

demanda. A capacidade de transmissão do enlace será compartilhada pacote por pacote somente entre usuários que tenham pacotes que precisam ser transmitidos pelo enlace. Tal compartilhamento de recursos por demanda (e não por alocação prévia) às vezes é denominado multiplexação estatística de recursos.

Embora tanto a comutação de pacotes quanto a comutação de circuitos predominem nas redes de telecomunicação de hoje, a tendência é, sem dúvida, a comutação de pacotes. Até mesmo muitas das atuais redes de telefonia de comutação de circuitos estão migrando lentamente para a comutação de pacotes. Em especial, redes telefônicas frequentemente usam comutação de pacotes na parte cara de uma chamada telefônica para o exterior, isto é, na parte que não é processada em território nacional.

1.3.2 Como os pacotes percorrem as redes de comutadores de pacotes?

Anteriormente dissemos que um roteador conduz um pacote que chega em um de seus enlaces de comunicação para outro enlace. Mas como o roteador determina o enlace que deve conduzir o pacote? Na verdade, isso é feito de diferentes maneiras por diferentes tipos de rede de computadores. Neste capítulo introdutório, descreveremos uma abordagem popular, a saber, a abordagem empregada pela Internet.

Na Internet, cada pacote que atravessa a rede contém o seu endereço de destino em seu cabeçalho. Como os endereços postais, esse endereço possui uma estrutura hierárquica. Quando um pacote chega à um roteador na rede, o roteador examina uma parte do endereço de destino do pacote e conduz o pacote a um roteador adjacente. Especificamente falando, cada roteador possui uma base de encaminhamento que mapeia o endereço de destino (ou partes desse endereço) para enlaces de saída. Quando um pacote chega ao roteador, este examina o endereço e busca sua base utilizando esse endereço de destino para encontrar o enlace de saída apropriado. O roteador, então, direciona o pacote ao enlace de saída.

Vimos que um roteador usa um endereço de destino do pacote para indexar uma base de encaminhamento e determinar o enlace de saída apropriado. Mas essa afirmação traz ainda outra questão: como as bases de encaminhamento se configuram? Elas são configuradas manualmente em cada roteador ou a Internet utiliza um procedimento mais automático? Essa questão será estudada mais profundamente no Capítulo 4. Mas para aguçar seu apetite, observe que a Internet possui um número especial de protocolos de roteamento que são utilizados para configurar automaticamente as bases de encaminhamento. Um protocolo de roteamento pode, por exemplo, determinar o caminho mais curto de cada roteador a cada destino e utilizar os resultados desse caminho para configurar as bases de encaminhamento nos roteadores.

O processo de roteamento fim a fim é semelhante a um motorista que não quer fazer uso do mapa, preferindo pedir informações. Por exemplo, suponha que Joe vai dirigir da Filadélfia para 156 Lakeside Drive, em Orlando, Flórida. Primeiro, Joe vai ao posto de gasolina de seu bairro e pergunta como chegar a 156 Lakeside Drive, em Orlando, Flórida. O frentista do posto extrai a palavra Flórida do endereço e diz que Joe precisa pegar a interestadual I-95 South, cuja entrada fica ao lado do posto. Ele também diz a Joe para pedir outras informações assim que chegar a Flórida. Então, Joe pega a I-95 South até chegar a Jacksonville, na Flórida, onde pede mais informações a outro frentista. Este extrai a palavra Orlando do endereço e diz a Joe para continuar na I-95 até Daytona Beach, e lá se informar novamente. Em Daytona Beach, outro frentista também extrai a palavra Orlando do endereço e pede para que ele pegue a I-4 diretamente para Orlando. Joe segue suas orientações e chega a uma saída para Orlando. Ele vai até outro posto de gasolina, e desta vez o atendente extrai a palavra Lakeside Drive do endereço e diz a ele qual estrada seguir para Lakeside Drive. Assim que Joe chega a Lakeside Drive, ele pergunta a uma criança de bicicleta como chegar a seu destino. A criança extrai o número 156 do endereço e aponta para a casa. Joe finalmente chega a seu destino final.

Na analogia acima, os frentistas dos postos e a criança na bicicleta são semelhantes aos roteadores. As bases de encaminhamento, que estão no cérebro deles, foram configuradas por anos de experiência.

Você gostaria de ver a rota fim a fim que os pacotes realizam na Internet? Convidamos você a colocar a mão na massa e interagir com o programa Traceroute, visitando o site <http://www.traceroute.org>. (Para detalhes sobre o Traceoute, veja a Seção 1.4.)

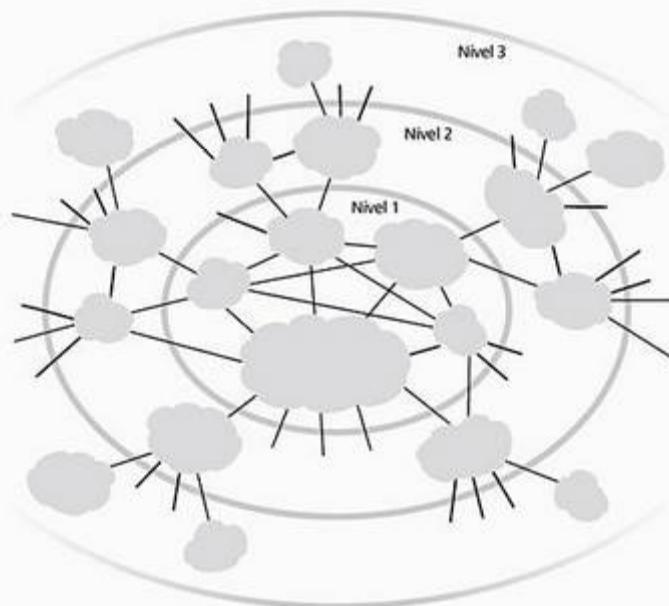
1.3.3 ISPs e backbones da Internet

Vimos anteriormente que sistemas finais (PCs de usuários, PDAs, servidores Web, servidores de correio eletrônico e assim por diante) conectam-se à Internet por meio de um provedor local. O provedor pode fornecer uma conectividade tanto com fio como sem fio, utilizando um conjunto de tecnologias de acesso, incluindo DSL, modem de cabo, FTTH, Wi-Fi, telefone celular e WiMAX. Observe que o provedor local não precisa ser uma operadora de telefonia ou uma empresa de TV a cabo: pode ser, por exemplo, uma universidade (que oferece acesso à Internet para os alunos, a equipe e o corpo docente) ou uma empresa (que oferece acesso para seus funcionários). Mas conectar usuários finais e provedores de conteúdo a redes de acesso é apenas uma pequena peça do quebra-cabeça que é conectar as centenas de milhões de usuários e centenas de milhares de redes que compõem a Internet. A Internet é uma *rede de redes* — entender essa frase é a chave para resolver esse jogo.

Na Internet pública, redes de acessos situadas na borda da Internet são conectadas ao restante da rede segundo uma hierarquia de níveis de ISPs, como mostra a Figura 1.15. Os ISPs de acesso estão no nível mais baixo dessa hierarquia. No topo dela está um número relativamente pequeno de ISPs denominados ISPs de nível 1. Sob muitos aspectos, um ISP de nível 1 é igual a qualquer rede — tem enlaces e roteadores e está conectado a outras redes. Mas, considerando-se outros aspectos, ISPs de nível 1 são especiais. As velocidades de seus enlaces muitas vezes alcançam 622 Mbps ou mais, tendo os maiores deles enlaces na faixa de 2,5 a 10 Gbps. Consequentemente, seus roteadores são capazes de transmitir pacotes a taxas extremamente altas. ISPs de nível 1 também apresentam as seguintes características:

- conectam-se diretamente a *cada* um dos outros ISPs de nível 1;
- conectam-se a um grande número de ISPs de nível 2 e a outras redes clientes;
- cobertura internacional.

Esses ISPs também são conhecidos como redes de backbone da Internet. Citamos, como exemplos, Sprint, MCI (anteriormente UUNet/WorldCom), AT&T, Level3, Qwest e Cable & Wireless. Em meados de 2002, a WorldCom era, de longe, o maior ISP de nível 1 existente — mais de duas vezes maior do que seu rival mais próximo, segundo diversas medições de tamanho [Telegraphy, 2002]. O interessante é que nenhum grupo san-



cional, oficialmente, seu status de nível 1. Como se costuma dizer, se você tiver de perguntar se é um membro de um grupo, provavelmente não é.

Um ISP de nível 2 normalmente tem alcance regional ou nacional e (o que é importante) conecta-se apenas a uns poucos ISPs de nível 1 (veja Figura 1.15).

Assim, para alcançar uma grande parcela da Internet global, um ISP de nível 2 tem de direcionar o tráfego por um dos ISPs de nível 1 com o qual está conectado. Um ISP de nível 2 é denominado um cliente do ISP de nível 1 com o qual está conectado, que, por sua vez, é denominado provedor de seu cliente. Muitas empresas de grande porte e instituições conectam suas redes corporativas diretamente a um provedor de nível 1 ou 2, tornando-se, desse modo, cliente daquele ISP. O provedor ISP cobra uma tarifa de seu cliente, que normalmente depende da taxa de transmissão do enlace que interliga ambos. Uma rede de nível 2 também pode preferir conectar-se diretamente a outras redes de mesmo nível, caso em que o tráfego pode fluir entre as duas sem ter de passar por uma rede de nível 1. Abaixo dos ISPs de nível 2 estão os de níveis mais baixos que se conectam à Internet por meio de um ou mais ISPs de nível 2 e, na parte mais baixa da hierarquia, estão os ISPs de acesso. Para complicar ainda mais as coisas, alguns provedores de nível 1 também são provedores de nível 2 (isto é, integrados verticalmente) e vendem acesso para Internet diretamente a usuários finais e provedores de conteúdo, bem como os ISPs de níveis mais baixos. Quando dois ISPs estão ligados diretamente um ao outro são denominados pares (peers) um do outro. Um estudo interessante [Subramanian, 2002] procura definir mais exatamente a estrutura em níveis da Internet estudando sua topologia em termos de relacionamentos cliente-provedor e entre parceiros (peer-peer). Para uma discussão mais clara sobre esses relacionamentos, veja Van der Berg, 2008.

Dentro da rede de um ISP, os pontos em que ele se conecta a outros ISPs (seja abaixo, acima ou no mesmo nível na hierarquia) são conhecidos como pontos de presença (*points of presence* — POPs). Um POP é simplesmente um grupo de um ou mais roteadores na rede do ISP com os quais roteadores em outros ISPs, ou em redes pertencentes a clientes do ISP, podem se conectar. Um provedor de nível 1 normalmente tem muitos POPs espalhados por diferentes localidades geográficas em sua rede e várias redes clientes e outros ISPs ligados a cada POP. Para se conectar ao POP de um provedor, uma rede cliente normalmente aluga um enlace de alta velocidade de um provedor de telecomunicações de terceiros e conecta um de seus roteadores diretamente a um roteador no POP do provedor. Dois ISPs de nível 1 também podem formar um par conectando um par de POPs, cada um proveniente de um dos dois ISPs. Além disso, dois ISPs podem ter vários pontos de emparelhamento conectando-se um ao outro em dois ou mais de pares de POPs.

Resumindo, a topologia da Internet é complexa, consistindo em dezenas de ISPs de níveis 1 e 2 e milhares de ISPs de níveis mais baixos. A cobertura dos ISPs é bastante diversificada; alguns abrangem vários continentes e oceanos e outros se limitam a pequenas regiões do mundo. Os ISPs de níveis mais baixos conectam-se a ISPs de níveis mais altos e estes (normalmente) se interconectam uns com os outros. Usuários e provedores de conteúdo são clientes de ISPs de níveis mais baixos e estes são clientes de ISPs de níveis mais altos.

1.4 Atraso, perda e vazão em redes de comutação de pacotes

Na Seção 1.1 dissemos que a Internet pode ser vista como uma infraestrutura que fornece serviços a aplicações distribuídas que são executadas nos sistemas finais. De modo ideal, gostaríamos que os serviços da Internet transferissem tantos dados quanto desejamos entre dois sistemas finais, instantaneamente, sem nenhuma perda. Infelizmente, esse é um objetivo elusivo, algo inalcançável. Ao contrário disso, as redes de computador, necessariamente, restringem a vazão (a quantidade de dados por segundo que podem ser transferidos) entre sistemas finais, apresentam atrasos entre sistemas finais e podem perder pacotes. Por um lado, infelizmente as leis físicas da realidade introduzem atraso e perda, bem como restringem a vazão. Por outro lado, como as redes de computadores possuem esses problemas, existem muitas questões fascinantes sobre como lidar com eles — mais do que questões suficientes para frequentar um curso de rede de computadores e motivar centenas de teses de doutorado! Nesta seção, começaremos a examinar e quantificar atraso, perda e vazão em redes de computadores.

1.4.1 Uma visão geral de atraso em redes de comutação de pacotes

Lembre-se de que um pacote começa em um sistema final (a origem), passa por uma série de roteadores e termina sua jornada em um outro sistema final (o destino). Quando um pacote viaja de um nó (sistema final ou roteador) ao nó subsequente (sistema final ou roteador), sofre, ao longo desse caminho, diversos tipos de atraso em *cada* nó existente no caminho. Os mais importantes deles são o atraso de processamento nodal, o atraso de fila, o atraso de transmissão e o atraso de propagação; juntos, eles se acumulam para formar o atraso nodal total. Para entender a fundo a comutação de pacotes e redes de computadores, é preciso entender a natureza e a importância desses atrasos.

Tipos de atraso

Vamos examinar esses atrasos no contexto da Figura 1.16. Como parte de sua rota fim a fim entre origem e destino, um pacote é enviado do nó anterior por meio do roteador A até o roteador B. Nossa meta é caracterizar o atraso nodal no roteador A. Note que este tem um enlace de saída que leva ao roteador B. Esse enlace é precedido de uma fila (também conhecida como buffer). Quando o pacote chega ao roteador A, vindo do nó anterior, o roteador examina o cabeçalho do pacote para determinar o enlace de saída apropriado e então o direciona ao enlace. Nesse exemplo, o enlace de saída para o pacote é o que leva ao roteador B. Um pacote pode ser transmitido por um enlace somente se não houver nenhum outro pacote sendo transmitido por ele e se não houver outros pacotes à sua frente na fila. Se o enlace estiver ocupado, ou com pacotes à espera, o pacote recém-chegado entrará na fila.

Atraso de processamento

O tempo requerido para examinar o cabeçalho do pacote e determinar para onde direcioná-lo é parte do atraso de processamento, que pode também incluir outros fatores, como o tempo necessário para verificar os erros em bits existentes no pacote que ocorreram durante a transmissão dos bits desde o nó anterior ao roteador A. Atrasos de processamento em roteadores de alta velocidade normalmente são da ordem de microssegundos, ou menos. Depois desse processamento nodal, o roteador direciona o pacote à fila que precede o enlace com o roteador B. (No Capítulo 4, estudaremos os detalhes da operação de um roteador.)

Atraso de fila

O pacote sofre um atraso de fila enquanto espera para ser transmitido no enlace. O tamanho desse atraso para um pacote específico dependerá da quantidade de outros pacotes que chegarem antes e que já estiverem na fila esperando pela transmissão. Se a fila estiver vazia, e nenhum outro pacote estiver sendo transmitido naquele momento, então o tempo de fila de nosso pacote será zero. Por outro lado, se o tráfego estiver pesado e houver muitos pacotes também esperando para ser transmitidos, o atraso de fila será longo. Em breve, veremos que o número de pacotes que um determinado pacote provavelmente encontrará ao chegar é uma função da intensi-

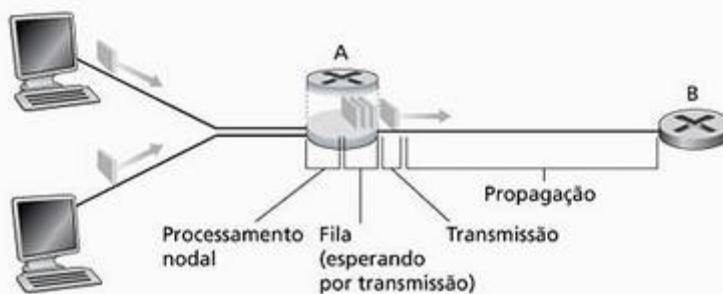


Figura 1.16 O atraso nodal no roteador A

dade e da natureza do tráfego que está chegando à fila. Na prática, atrasos de fila podem ser da ordem de micro a milissegundos.

Atraso de transmissão

Admitindo-se que pacotes são transmitidos segundo a estratégia de "o primeiro a chegar será o primeiro a ser processado", como é comum em redes de comutação de pacotes, nosso pacote somente poderá ser transmitido depois que todos aqueles que chegaram antes tenham sido enviados. Denominemos o tamanho do pacote como L bits e a velocidade de transmissão do enlace do roteador A ao roteador B como R bits/s. A velocidade R é determinada pela velocidade de transmissão do enlace ao roteador B. Por exemplo, para um enlace Ethernet de 10 Mbps, a velocidade é $R \cdot 10$ Mbps; para um enlace Ethernet de 100 Mbps, a velocidade é $R \cdot 100$ Mbps. O atraso de transmissão (também denominado atraso de armazenamento e reenvio, como discutimos na Seção 1.3) é L/R . Esta é a quantidade de tempo requerida para empurrar (isto é, transmitir) todos os bits do pacote para o enlace. Na prática, atrasos de transmissão são comumente da ordem de micro a milissegundos.

Atraso de propagação

Assim que é lançado no enlace, um bit precisa se propagar até o roteador B. O tempo necessário para propagar o bit desde o início do enlace até o roteador B é o atraso de propagação. O bit se propaga à velocidade de propagação do enlace, a qual depende do meio físico do enlace (isto é, fibra ótica, par de fios de cobre trançado e assim por diante) e está na faixa de

$$2 \cdot 10^8 \text{ m/s} \text{ a } 3 \cdot 10^8 \text{ m/s}$$

que é igual à velocidade da luz, ou um pouco menor. O atraso de propagação é a distância entre dois roteadores dividida pela velocidade de propagação. Isto é, o atraso de propagação é d/s , onde d é a distância entre o roteador A e o roteador B, e s é a velocidade de propagação do enlace. Assim que o último bit do pacote se propagar até o nó B, ele e todos os outros bits precedentes do pacote serão armazenados no roteador B. Então, o processo inteiro continua, agora com o roteador B executando a retransmissão. Em redes WAN, os atrasos de propagação são da ordem de milissegundos.

Comparação entre atrasos de transmissão e de propagação

Os principiantes na área de redes de computadores às vezes têm dificuldade para entender a diferença entre atraso de transmissão e atraso de propagação. A diferença é sutil, mas importante. O atraso de transmissão é a quantidade de tempo requerida para o roteador empurrar o pacote para fora; é uma função do comprimento do pacote e da taxa de transmissão do enlace, mas nada tem a ver com a distância entre os dois roteadores. O atraso de propagação, por outro lado, é o tempo que leva para um bit se propagar de um roteador até o seguinte; é uma função da distância entre os dois roteadores, mas nada tem a ver com o comprimento do pacote ou com a taxa de transmissão do enlace.

Podemos esclarecer melhor as noções de atrasos de transmissão e de propagação com uma analogia. Considere uma rodovia que tenha um posto de pedágio a cada 100 quilômetros, como mostrado na Figura 1.17. Imagine que os trechos da rodovia entre os postos de pedágio sejam enlaces e que os postos de pedágio sejam roteadores. Suponha que os carros trafeguem (isto é, se propaguem) pela rodovia a uma velocidade de 100 km/h (isto é, quando o carro sai de um posto de pedágio, acelera instantaneamente até 100 km/h e mantém essa velocidade entre os dois postos de pedágio). Agora, suponha que dez carros viajem em comboio, um atrás do outro, em ordem fixa. Imagine que cada carro seja um bit e que o comboio seja um pacote. Suponha ainda que cada posto de pedágio libere (isto é, transmita) um carro a cada 12 segundos, que seja tarde da noite e que os carros do comboio sejam os únicos na estrada. Por fim, suponha que, ao chegar a um posto de pedágio, o primeiro carro do comboio aguarde na entrada até que os outros nove cheguem e formem uma fila atrás dele. (Assim, o comboio inteiro deve ser 'armazenado' no posto de pedágio antes de começar a ser 'reenviado'.) O tempo necessário para que todo o comboio passe pelo posto de pedágio e volte à estrada é de $(10 \text{ carros}) / (5 \text{ carros/minuto}) = 2 \text{ minutos}$. Esse tempo é análogo ao atraso de transmissão em um roteador. O tempo necessário para um carro trafegar da

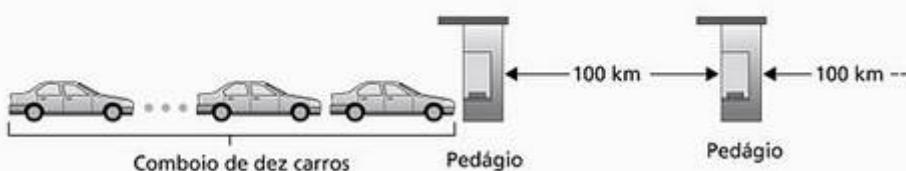


Figura 1.17 Analogia do comboio

saída de um posto de pedágio até o próximo posto de pedágio é de $(100 \text{ km})/(100 \text{ km/h}) = 1 \text{ hora}$. Esse tempo é análogo ao atraso de propagação. Portanto, o tempo decorrido entre o instante em que o comboio é 'armazenado' em frente a um posto de pedágio até o instante em que é 'armazenado' em frente ao seguinte é a soma do atraso de transmissão e do atraso de propagação — nesse exemplo, 62 minutos.

Vamos explorar um pouco mais essa analogia. O que aconteceria se o tempo de liberação do comboio no posto de pedágio fosse maior do que o tempo que um carro leva para trafegar entre dois postos? Por exemplo, suponha que os carros trafeguem a uma velocidade de 1.000 km/h e que o pedágio libere um carro por minuto. Então, o atraso de trânsito entre dois postos de pedágio é de 6 minutos e o tempo de liberação do comboio no posto de pedágio é de 10 minutos. Nesse caso, os primeiros carros do comboio chegarão ao segundo posto de pedágio antes que os últimos carros saiam do primeiro posto. Essa situação também acontece em redes de comutação de pacotes — os primeiros bits de um pacote podem chegar a um roteador enquanto muitos dos remanescentes ainda estão esperando para ser transmitidos pelo roteador precedente.

Se uma imagem vale mil palavras, então uma animação vale um milhão. O Companion Website apresenta um aplicativo interativo Java que ilustra e contrasta o atraso de transmissão e o atraso de propagação. Recomenda-se que o leitor visite esse aplicativo.

Se d_{proc} , d_{fila} , d_{trans} e d_{prop} forem, respectivamente, os atrasos de processamento, de fila, de transmissão e de propagação, então o atraso nodal total é dado por:

$$d_{\text{nodal}} = d_{\text{proc}} + d_{\text{fila}} + d_{\text{trans}} + d_{\text{prop}}$$

A contribuição desses componentes do atraso pode variar significativamente. Por exemplo, d_{prop} pode ser desprezível (por exemplo, dois microsegundos) para um enlace que conecta dois roteadores no mesmo campus universitário; contudo, é de centenas de milissegundos para dois roteadores interconectados por um enlace de satélite geoestacionário e pode ser o termo dominante no d_{nodal} . De maneira semelhante, d_{trans} pode variar de desprezível a significativo. Sua contribuição normalmente é desprezível para velocidades de transmissão de 10 Mbps e mais altas (por exemplo, em LANs); contudo, pode ser de centenas de milissegundos para grandes pacotes de Internet enviados por enlaces de modems discados de baixa velocidade. O atraso de processamento, d_{proc} , é quase sempre desprezível; no entanto, tem forte influência sobre a produtividade máxima de um roteador, que é a velocidade máxima com que ele pode encaminhar pacotes.

1.4.2 Atraso de fila e perda de pacote

O mais complicado e interessante componente do atraso nodal é o atraso de fila, d_{fila} . Realmente, o atraso de fila é tão importante e interessante em redes de computadores que milhares de artigos e numerosos livros já foram escritos sobre ele [Bertsekas, 1991; Daigle, 1991; Kleinrock, 1975, 1976; Ross, 1995]. Neste livro, faremos apenas uma discussão intuitiva, de alto nível, sobre o atraso de fila; o leitor mais curioso pode consultar alguns dos livros citados (ou até mesmo escrever uma tese sobre o assunto!). Diferentemente dos três outros atrasos (a saber, d_{proc} , d_{trans} e d_{prop}), o atraso de fila pode variar de pacote a pacote. Por exemplo, se dez pacotes chegarem a uma fila vazia ao mesmo tempo, o primeiro pacote transmitido não sofrerá nenhum atraso, ao passo que o último pacote sofrerá um atraso relativamente grande (enquanto espera que os outros nove pacotes sejam transmitidos). Por conseguinte, para se caracterizar um atraso de fila, normalmente são utilizadas medições estatísticas, tais como atraso de fila médio, variância do atraso de fila e a probabilidade de ele exceder um valor especificado.

Quando o atraso de fila é grande e quando é insignificante? A resposta a essa pergunta depende da velocidade de transmissão do enlace, da taxa com que o tráfego chega à fila e de sua natureza, isto é, se chega intermitentemente, em rajadas. Para entendermos melhor, vamos adotar a para representar a taxa média com que os pacotes chegam à fila (a é medida em pacotes/segundo). Lembre-se de que R é a taxa de transmissão, isto é, a taxa (em bits/segundo) com que os bits são retirados da fila. Suponha também, para simplificar, que todos os pacotes tenham L bits. Então, a taxa média com que os bits chegam à fila é La bits/s. Por fim, imagine que a fila seja muito longa, de modo que, essencialmente, possa conter um número infinito de bits. A razão La/R , denominada intensidade de tráfego, frequentemente desempenha um papel importante na estimativa do tamanho do atraso de fila. Se $La/R > 1$, então a velocidade média com que os bits chegam à fila excederá a velocidade com que eles podem ser transmitidos para fora da fila. Nessa situação desastrosa, a fila tenderá a aumentar sem limite e o atraso de fila tenderá ao infinito! Por conseguinte, uma das regras de ouro da engenharia de tráfego é: projete seu sistema de modo que a intensidade de tráfego não seja maior do que 1.

Agora, considere o caso em que $La/R \leq 1$. Aqui, a natureza do tráfego influencia o atraso de fila. Por exemplo, se pacotes chegarem periodicamente — isto é, se chegar um pacote a cada L/R segundos — então todos os pacotes chegarão a uma fila vazia e não haverá atraso. Por outro lado, se pacotes chegarem em rajadas, mas periodicamente, poderá haver um significativo atraso de fila médio. Por exemplo, suponha que N pacotes cheguem ao mesmo tempo a cada $(L/R)N$ segundos. Então, o primeiro pacote transmitido não sofrerá atraso de fila; o segundo pacote transmitido terá um atraso de fila de L/R segundos e, de modo mais geral, o enésimo pacote transmitido terá um atraso de fila de $(n - 1)L/R$ segundos. Deixamos como exercício para o leitor o cálculo do atraso de fila médio para esse exemplo.

Os dois exemplos de chegadas periódicas que acabamos de descrever são um tanto acadêmicos. Na realidade, o processo de chegada a uma fila é aleatório — isto é, não segue um padrão e os intervalos de tempo entre os pacotes são ao acaso. Nessa hipótese mais realista, a quantidade La/R normalmente não é suficiente para caracterizar por completo a estatística do atraso. Não obstante, é útil para entender intuitivamente a extensão do atraso de fila. Em especial, se a intensidade de tráfego for próxima de zero, então as chegadas de pacotes serão poucas e bem espaçadas e é improvável que um pacote que esteja chegando encontre outro na fila. Consequentemente, o atraso de fila médio será próximo de zero. Por outro lado, quando a intensidade de tráfego for próxima de 1, haverá intervalos de tempo em que a velocidade de chegada excederá a capacidade de transmissão (devido às variações na taxa de chegada do pacote) e uma fila será formada durante esses períodos de tempo; quando a taxa de chegada for menor do que a capacidade de transmissão, a extensão da fila diminuirá. Todavia, à medida que a intensidade de tráfego se aproxima de 1, o comprimento médio da fila fica cada vez maior. A dependência qualitativa entre o atraso de fila médio e a intensidade de tráfego é mostrada na Figura 1.18.

Um aspecto importante a observar na Figura 1.18 é que, quando a intensidade de tráfego se aproxima de 1, o atraso de fila médio aumenta rapidamente. Uma pequena porcentagem de aumento na intensidade resulta em

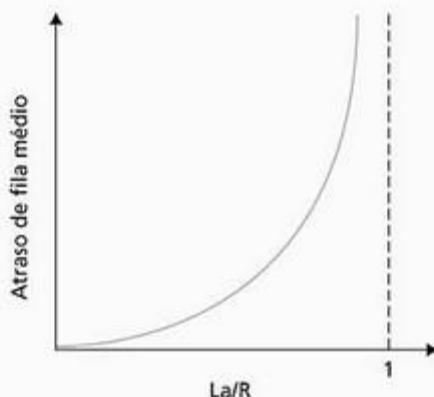


Figura 1.18 Dependência entre atraso de fila médio e intensidade de tráfego

um aumento muito maior no atraso, em termos de porcentagem. Talvez você já tenha percebido esse fenômeno na estrada. Se você dirige regularmente por uma estrada que normalmente está congestionada, o fato de ela estar sempre assim significa que a intensidade de tráfego é próxima de 1. Se algum evento causar um tráfego ligeiramente maior do que o usual, as demoras que você sofrerá poderão ser enormes.

Para compreender um pouco mais os atrasos de fila, visite o Companion Website, que apresenta um aplicativo Java interativo. Se você aumentar a taxa de chegada do pacote o suficiente de forma que a intensidade do tráfego exceda 1, você verá a fila aumentar ao longo do tempo.

Perda de pacote

Na discussão anterior, admitimos que a fila é capaz de conter um número infinito de pacotes. Na realidade, a capacidade da fila que precede um enlace é finita, embora a sua formação dependa bastante do projeto e do custo do comutador. Como a capacidade da fila é finita, na verdade os atrasos de pacote não se aproximam do infinito quando a intensidade de tráfego se aproxima de 1. O que realmente acontece é que um pacote pode chegar e encontrar uma fila cheia. Sem espaço disponível para armazená-lo, o roteador descartará esse pacote; isto é, ele será perdido. Esse excesso em uma fila pode ser observado novamente no aplicativo Java quando a intensidade do tráfego é maior do que 1. Do ponto de vista de um sistema final, uma perda de pacote é vista como um pacote que foi transmitido para o núcleo da rede, mas sem nunca ter emergido dele no destino. A fração de pacotes perdidos aumenta com o aumento da intensidade de tráfego. Por conseguinte, o desempenho em um nó é frequentemente medido não apenas em termos de atraso, mas também em termos da probabilidade de perda de pacotes. Como discutiremos nos capítulos subsequentes, um pacote perdido pode ser retransmitido fim a fim para garantir que todos os dados sejam finalmente transferidos da origem ao local de destino.

1.4.3 Atraso fim a fim

Até o momento, nossa discussão focalizou o atraso nodal, isto é, o atraso em um único roteador. Concluiremos essa discussão considerando brevemente o atraso da origem ao destino. Para entender esse conceito, suponha que haja $N - 1$ roteadores entre a máquina de origem e a máquina de destino. Imagine também que a rede não esteja congestionada (e, portanto, os atrasos de fila sejam desprezíveis), que o atraso de processamento em cada roteador e na máquina de origem seja d_{proc} , que a taxa de transmissão de saída de cada roteador e da máquina de origem seja R bits/s e que o atraso de propagação em cada enlace seja d_{prop} . Os atrasos nodais se acumulam e resultam em um atraso fim a fim

$$d_{fim\ a\ fim} = N (d_{proc} + d_{trans} + d_{prop})$$

onde, mais uma vez, $d_{trans} = L/R$ e L é o tamanho do pacote. Convidamos você a generalizar essa fórmula para o caso de atrasos heterogêneos nos nós e para o caso de um atraso de fila médio em cada nó.

Traceroute

Para perceber o que é realmente o atraso em uma rede de computadores, podemos utilizar o Traceroute, programa de diagnóstico que pode ser executado em qualquer máquina da Internet. Quando o usuário especifica um nome de hospedeiro de destino, o programa no hospedeiro de origem envia vários pacotes especiais em direção àquele destino. Ao seguir seu caminho até o destino, esses pacotes passam por uma série de roteadores. Um deles recebe um desses pacotes especiais e envia uma curta mensagem à origem. Essa mensagem contém o nome e o endereço do roteador.

Mais especificamente, suponha que haja $N - 1$ roteadores entre a origem e o destino. Então, a fonte enviará N pacotes especiais à rede e cada um deles estará endereçado ao destino final. Esses N pacotes especiais serão marcados de 1 a N , sendo a marca do primeiro pacote 1 e a do último, N . Assim que o enésimo roteador recebe o enésimo pacote com a marca n , não envia o pacote a seu destino, mas uma mensagem à origem. Quando o hospedeiro de destino recebe o pacote N , também envia uma mensagem à origem, que registra o tempo transcorrido entre o envio de um pacote e o recebimento da mensagem de retorno correspondente. A origem registra também o

nome e o endereço do roteador (ou do hospedeiro de destino) que retorna a mensagem. Dessa maneira, a origem pode reconstruir a rota tomada pelos pacotes que vão da origem ao destino e pode determinar os atrasos de ida e volta para todos os roteadores intervenientes. Na prática, o programa Traceroute repete o processo que acabamos de descrever três vezes, de modo que a fonte envia, na verdade, $3 \cdot N$ pacotes ao destino. O RFC 1393 descreve detalhadamente o Traceroute.

Eis um exemplo de resultado do programa Traceroute, no qual a rota traçada ia do hospedeiro de origem `gaia.cs.umass.edu` (na Universidade de Massachusetts) até `cis.poly.edu` (na Polytechnic University no Brooklyn). O resultado tem seis colunas: a primeira coluna é o valor n descrito acima, isto é, o número do roteador ao longo da rota; a segunda coluna é o nome do roteador; a terceira coluna é o endereço do roteador (na forma `xxx.xxx.xxx.xxx`); as últimas três colunas são os atrasos de ida e volta para três tentativas. Se a fonte receber menos do que três mensagens de qualquer roteador determinado (devido à perda de pacotes na rede), o Traceroute coloca um asterisco logo após o número do roteador e registra menos do que três tempos de duração de viagens de ida e volta para aquele roteador.

```
1 cs-gw (128.119.240.254) 1.009 ms 0.899 ms 0.993 ms
2 128.119.3.154 (128.119.3.154) 0.931 ms 0.441 ms 0.651 ms
3 border4-rt-gi-1-3.gw.umass.edu (128.119.2.194) 1.032 ms 0.484 ms 0.451 ms
4 acrl-ge-2-1-0.Boston.cw.net (208.172.51.129) 10.006 ms 8.150 ms 8.460 ms
5 agr4-loopback.NewYork.cw.net (206.24.194.104) 12.272 ms 14.344 ms 13.267 ms
6 acr2-loopback.NewYork.cw.net (206.24.194.62) 13.225 ms 12.292 ms 12.148 ms
7 pos10-2.core2.NewYork1.Level3.net (209.244.160.133) 12.218 ms 11.823 ms 11.793 ms
8 gige9-1-52.hsipaccess1.NewYork1.Level3.net (64.159.17.39) 13.081 ms 11.556 ms 13.297 ms
9 p0-0.polyu.bbnplanet.net (4.25.109.122) 12.716 ms 13.052 ms 12.786 ms
10 cis.poly.edu (128.238.32.126) 14.080 ms 13.035 ms 12.802 ms
```

No exemplo anterior há nove roteadores entre a origem e o destino. Quase todos eles têm um nome e todos têm endereço. Por exemplo, o nome do Roteador 3 é `border4-rt-gi-1-3.gw.umass.edu` e seu endereço é `128.119.2.194`. Examinando os dados apresentados para esse roteador, verificamos que na primeira das três tentativas, o atraso de ida e volta entre a origem e o roteador foi de 1,03 milissegundos. Os atrasos para as duas tentativas subsequentes foram 0,48 e 0,45 milissegundos e incluem todos os atrasos que acabamos de discutir, ou seja, atrasos de transmissão, de propagação, de processamento do roteador e de fila. Como o atraso de fila varia com o tempo, o atraso de ida e volta do pacote n enviado a um roteador n pode, às vezes, ser maior do que o do pacote $n+1$ enviado ao roteador $n+1$. Realmente, observamos esse fenômeno no exemplo acima: os atrasos do roteador 6 são maiores que os atrasos do roteador 7!

Você quer experimentar o Traceroute por conta própria? Recomendamos muito que visite o site <http://www.traceroute.org>, que provê uma interface Web para uma extensa lista de fontes para traçar rotas. Escolha uma fonte, forneça o nome de hospedeiro para qualquer destino e o programa Traceroute fará todo o trabalho. Existem muitos programas de software gratuitos que apresentam uma interface gráfica para o Traceroute; um dos nossos favoritos é o PingPlotter [PingPlotter, 2009].

Sistema final, aplicativo e outros atrasos

Além dos atrasos de processamento, transmissão e de propagação, os sistemas finais podem adicionar outros atrasos significativos. Os modems discados apresentam um atraso de modulação/codificação, que pode ser da ordem de dezenas de microssegundos. (Os atrasos de modulação/codificação para outras tecnologias de acesso — incluindo a Ethernet, o modem a cabo e a DSL — são menos significativos e normalmente desprezíveis.) Um sistema final que quer transmitir um pacote para uma mídia compartilhada (por exemplo, como em um cenário WiFi ou Ethernet) pode, intencionalmente, atrasar sua transmissão como parte de seu protocolo por compartilhar a mídia com outros sistemas finais; vamos analisar tais protocolos no Capítulo 5. Um outro importante atraso é o atraso de empacotamento de mídia, o qual está presente nos aplicativos VoIP (voz sobre IP). No VoIP, o

remetente deve primeiro carregar um pacote com voz digitalizada e codificada antes de transmitir o pacote para a Internet. A etapa de carregar um pacote — chamada de atraso de empacotamento — pode ser significativa e ter impacto sobre a qualidade visível pelo usuário de uma chamada VoIP. Esse assunto será explorado mais adiante nos exercícios de fixação no final deste capítulo.

1.4.4 Vazão nas redes de computadores

Além do atraso e da perda de pacotes, outra medida de desempenho importante em redes de computadores é a vazão fim a fim. Para definir vazão, considere a transferência de um arquivo grande do Hospedeiro A para o Hospedeiro B através de uma rede de computadores. Essa transferência pode ser, por exemplo, um clipe extenso de um parceiro para outro por meio do sistema de compartilhamento P2P. A vazão instantânea a qualquer momento é a taxa (em bits/s) em que o Hospedeiro B está recebendo o arquivo. (Muitos aplicativos, incluindo muitos sistemas de compartilhamento P2P, exibem a vazão instantânea durante os downloads na interface do usuário — talvez você já tenha observado isso!) Se o arquivo consistir em F bits e a transferência levar T segundos para o Hospedeiro B receber todos os F bits, então a vazão média da transferência do arquivo é F/T bits/s. Para algumas aplicações, como a telefonia via Internet, é desejável ter um atraso baixo e uma vazão instantânea acima de algum limiar (por exemplo, superior a 24 kbps para telefonia via Internet, e superior a 256 kbps para alguns aplicativos de vídeo em tempo real). Para outras aplicações, incluindo as de transferência de arquivo, o atraso não é importante, mas é recomendado ter a vazão mais alta possível.

Para obter uma visão mais detalhada, vamos analisar alguns exemplos. A Figura 1.19 (a) mostra dois sistemas finais, um servidor e um cliente, conectados por dois enlaces de comunicação e um roteador. Considere a vazão para uma transferência de arquivo do servidor para o cliente. Suponha que R_s seja a taxa do enlace entre o roteador e o cliente; e R_c seja a taxa do enlace entre o roteador e o cliente. Imagine que os únicos bits enviados na rede inteira sejam os do servidor para o cliente. Agora vem a pergunta, neste cenário ideal, qual é a vazão servidor-para-cliente? Para responder a ela, pense nos bits como um *líquido* e nos enlaces de comunicação como *canos*. Evidentemente, o servidor não pode enviar os bits através de seu enlace a uma taxa mais rápida do que R_s bps, e o roteador não pode encaminhar os bits a uma taxa mais rápida do que R_c bps. Se $R_s < R_c$, então os bits enviados pelo servidor irão “correr” diretamente pelo roteador e chegar no cliente a uma taxa de R_s bps. Se, por outro lado, $R_c < R_s$, então o roteador não poderá encaminhar os bits tão rapidamente quanto ele os recebe. Neste caso, os bits somente deixarão o roteador a uma taxa R_c , dando uma vazão fim a fim de R_c . (Observe também que se os bits continuarem a chegar no roteador a uma taxa R_s , e continuarem a deixar o roteador a uma taxa R_c , o acúmulo de bits no roteador esperando para transmissão ao cliente só aumentará — uma situação, na maioria das vezes, indesejável!) Assim, para essa rede simples de dois enlaces, a vazão é $\min\{R_s, R_c\}$, ou seja, é a taxa de transmissão do enlace de gargalo. Após determinar a vazão, agora podemos

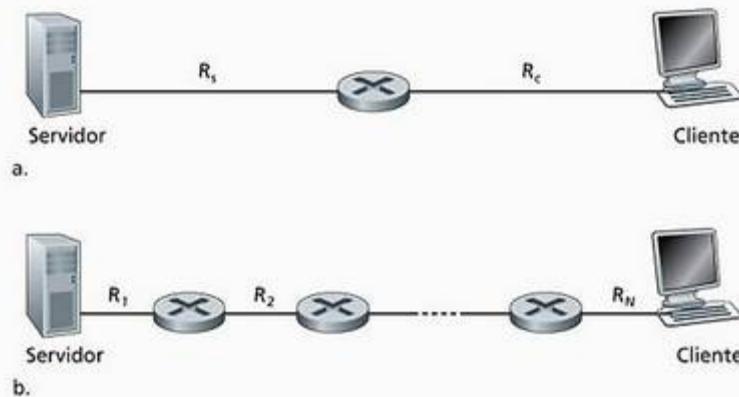


Figura 1.19 Vazão para uma transferência de arquivo do servidor ao cliente

aproximar o tempo que leva para transferir um arquivo grande de F bits do servidor ao cliente como $F/\min\{R_s, R_c\}$. Para um exemplo específico, suponha que você está fazendo o download de um arquivo MP3 de $F = 32$ milhões de bits, o servidor tem uma taxa de transmissão de $R_s = 2$ Mbps, e você tem um enlace de acesso de $R_c = 1$ Mbps. O tempo necessário para transferir o arquivo é, então, 32 segundos. Naturalmente essas expressões para tempo de vazão e de transferência são apenas aproximações, já que elas não contabilizam assuntos relacionados a protocolos e pacotes.

A Figura 1.19 (b) agora mostra uma rede com N enlaces entre o servidor e o cliente, com as taxas de transmissão de N enlaces sendo R_1, R_2, \dots, R_N . Aplicando a mesma análise da rede de dois enlaces, descobrimos que a vazão para uma transferência de arquivo do servidor ao cliente é $\min\{R_1, R_2, \dots, R_N\}$, a qual é novamente a taxa de transmissão do enlace de gargalo ao longo do caminho entre o servidor e o cliente.

Agora considere outro exemplo motivado pela Internet de hoje. A Figura 1.20 (a) mostra dois sistemas finais, um servidor e um cliente, conectados a uma rede de computadores. Considere a vazão para uma transferência de arquivo do servidor ao cliente. O servidor está conectado à rede com um enlace de acesso de taxa R_s e o cliente está conectado à rede com um enlace de acesso de R_c . Agora suponha que todos os enlaces no núcleo da rede de comunicação tenham taxas altas de transmissão, muito maiores do que R_s e R_c . Realmente, hoje, o núcleo da Internet está superabastecido com enlaces de alta velocidade que sofrem pouco congestionamento [Akella, 2003]. Suponha, também, que os únicos bits que estão sendo enviados em toda a rede sejam os do servidor para o cliente. Como o núcleo da rede de computadores é como um cano largo neste exemplo, a taxa a qual os bits correm da origem ao lugar de destino é novamente o mínimo de R_s e R_c , ou seja, vazão = $\min\{R_s, R_c\}$. Portanto, o fator coercitivo para vazão na Internet de hoje é, normalmente, o acesso à rede.

Para um exemplo final, considere a Figura 1.20 (b) na qual existem 10 servidores e 10 clientes conectados ao núcleo da rede de computadores. Neste exemplo, 10 downloads simultâneos estão sendo realizados, envolvendo 10 pares de clientes-servidores. Suponha que esses 10 downloads sejam o único tráfego na rede no momento presente. Como mostrado na figura, há um enlace no núcleo que é atravessado por todos os 10 downloads. Considere R a taxa de transmissão desse enlace. Suponha que todos os enlaces de acesso do servidor possuem a mesma taxa

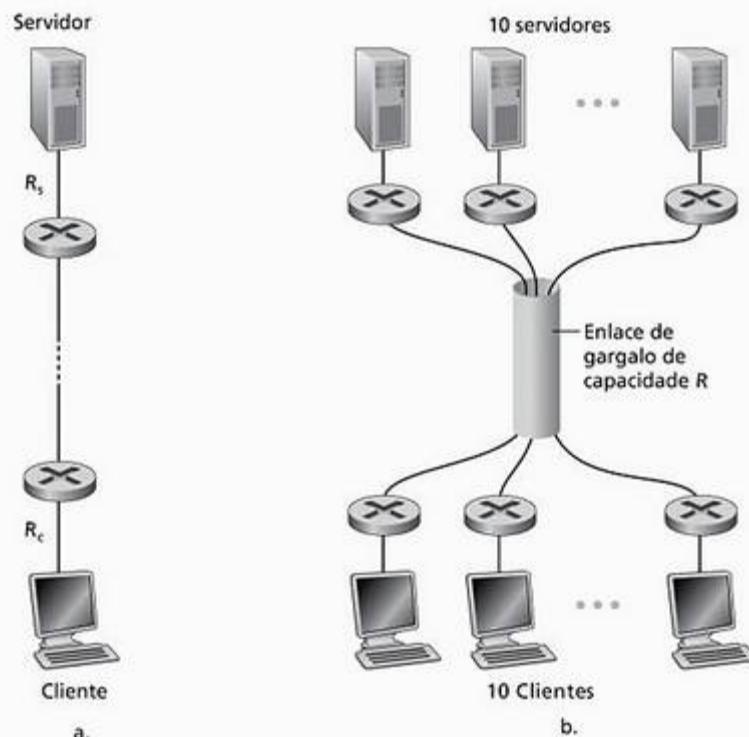


Figura 1.20 Vazão fim a fim: (a) O cliente baixa um arquivo do servidor; (b) 10 clientes fazem o download com 10 servidores

R_s , todos os enlaces de acesso do cliente possuem a mesma taxa R_c e a taxa de transmissão de todos os enlaces no núcleo — com exceção de um enlace comum de taxa R — sejam muito maiores do que R_s , R_c e R . Agora perguntamos, quais são as vazões dos downloads? Evidentemente, se a taxa do enlace comum, R , é grande — digamos que cem vezes maior do que R_s , R_c — então a vazão para cada download será novamente mÍn $\{R_s, R_c\}$. Mas e se a taxa do enlace comum for da mesma ordem que R_s e R_c ? Qual será a vazão neste caso? Vamos observar um exemplo específico. Suponha que $R_s = 2$ Mbps, $R_c = 1$ Mbps, $R = 5$ Mbps, e o enlace comum divide sua taxa de transmissão igualmente entre 10 downloads. Então, o gargalo para cada download não se encontra mais na rede de acesso, mas é o enlace compartilhado no núcleo, que somente fornece para cada download 500 kbps de vazão. Deste modo, a vazão fim a fim para cada download é agora reduzida a 500 kbps.

Os exemplos na Figura 1.19 e Figura 1.20 (a) mostram que a vazão depende das taxas de transmissão dos enlaces sobre as quais os dados correm. Vimos que quando não há tráfego interventor, a vazão pode simplesmente ser aproximada como a taxa de transmissão mínima ao longo do caminho entre a origem e o local de destino. O exemplo na Figura 1.20 (b) mostra que, de maneira geral, a vazão depende não somente das taxas de transmissão dos enlaces ao longo do caminho, mas também do tráfego interventor. Em especial, um enlace com uma alta taxa de transmissão pode, todavia, ser o enlace de gargalo para uma transferência de arquivo, caso muitos outros fluxos de dados estejam também passando por aquele enlace. Analisaremos mais detalhadamente vazão em redes de computadores nos exercícios de fixação e nos capítulos subsequentes.

1.5 Camadas de protocolo e seus modelos de serviço

Até aqui, nossa discussão demonstrou que a Internet é um sistema *extremamente* complicado e que possui muitos componentes: inúmeras aplicações e protocolos, vários tipos de sistemas finais e conexões entre eles, roteadores, além de vários tipos de meios físicos de enlace. Dada essa enorme complexidade, há alguma esperança de organizar a arquitetura de rede ou, ao menos, nossa discussão sobre ela? Felizmente, a resposta a ambas as perguntas é sim.

1.5.1 Arquitetura de camadas

Antes de tentarmos organizar nosso raciocínio sobre a arquitetura da Internet, vamos procurar uma analogia humana. Na verdade, lidamos com sistemas complexos o tempo todo em nosso dia a dia. Imagine se alguém pedisse que você descrevesse, por exemplo, o sistema de uma companhia aérea. Como você encontraria a estrutura para descrever esse sistema complexo que tem agências de emissão de passagens, pessoal para embarcar a bagagem para ficar no portão de embarque, pilotos, aviões, controle de tráfego aéreo e um sistema mundial de roteamento de aeronaves? Um modo de descrever esse sistema poderia ser apresentar a relação de uma série de ações que você realiza (ou que outros realizam para você) quando voa por uma empresa aérea. Você compra a passagem, despacha suas malas, dirige-se ao portão de embarque e, finalmente, entra no avião, que decola e segue uma rota até seu destino. Após a aterrissagem, você desembarca no portão designado e recupera suas malas. Se a viagem foi ruim, você reclama na agência que lhe vendeu a passagem (esforço em vão). Esse cenário é ilustrado na Figura 1.21.

Já podemos notar aqui algumas analogias com redes de computadores: você está sendo despachado da origem ao destino pela companhia aérea; um pacote é despachado da máquina de origem à máquina de destino na Internet. Mas essa não é exatamente a analogia que buscamos. Estamos tentando encontrar alguma estrutura na Figura 1.21. Observando essa figura, notamos que há uma função referente à passagem em cada ponta; há também uma função de bagagem para passageiros que já apresentaram a passagem e uma função de portão de embarque para passageiros que já apresentaram a passagem e despacharam a bagagem. Para passageiros que já passaram pelo portão de embarque (isto é, aqueles que já apresentaram a passagem, despacharam a bagagem e passaram pelo portão), há uma função de decolagem e de aterrissagem e, durante o voo, uma função de roteamento do avião. Isso sugere que podemos examinar a funcionalidade na Figura 1.21 na horizontal, como mostra a Figura 1.22.

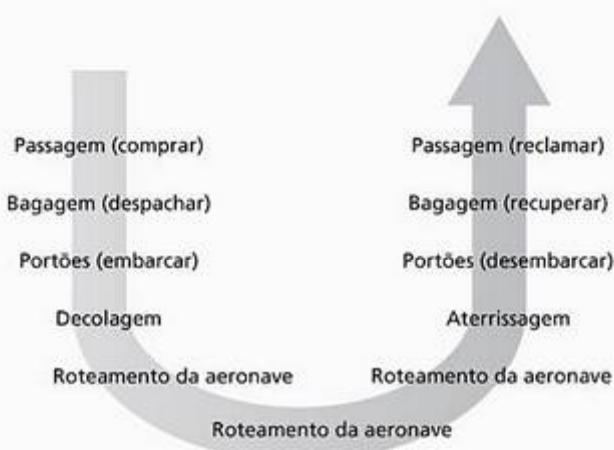


Figura 1.21 Uma viagem de avião: ações

A Figura 1.22 dividiu a funcionalidade da linha aérea em camadas, provendo uma estrutura com a qual podemos discutir a viagem aérea. Note que cada camada, combinada com as camadas abaixo dela, implementa alguma funcionalidade, algum serviço. Na camada da passagem aérea e abaixo dela, é realizada a transferência ‘balcão-de-linha-aérea-balcão-de-linha-aérea’ de um passageiro. Na camada de bagagem e abaixo dela, é realizada a transferência ‘despacho-de-bagagem–recuperação-de-bagagem’ de um passageiro e de suas malas. Note que a camada da bagagem provê esse serviço apenas para a pessoa que já apresentou a passagem. Na camada do portão, é realizada a transferência ‘portão-de-embarque-portão-de-desembarque’ de um passageiro e de sua bagagem. Na camada de decolagem/aterrissagem, é realizada a transferência ‘pista-a-pista’ de passageiros e de suas bagagens. Cada camada provê seu serviço (1) realizando certas ações dentro da camada (por exemplo, na camada do portão, embarcar e desembarcar pessoas de um avião) e (2) utilizando os serviços da camada imediatamente inferior (por exemplo, na camada do portão, aproveitando o serviço de transferência ‘pista-a-pista’ de passageiros da camada de decolagem/aterrissagem).

Uma arquitetura de camadas nos permite discutir uma parcela específica e bem definida de um sistema grande e complexo. Essa simplificação tem considerável valor intrínseco, pois provê modularidade fazendo com que fique muito mais fácil modificar a implementação do serviço prestado pela camada. Contanto que a camada forneça o mesmo serviço para a que está acima dela e use os mesmos serviços da camada abaixo dela, o restante do sistema permanece inalterado quando a sua implementação é modificada. (Note que modificar a implementação de um serviço é muito diferente de mudar o serviço em si!) Por exemplo, se as funções de portão fossem modificadas (digamos que passassem a embarcar e desembarcar passageiros por ordem de altura), o restante do sistema da linha aérea permaneceria inalterado, já que a camada do portão continuaria a prover a mesma função.

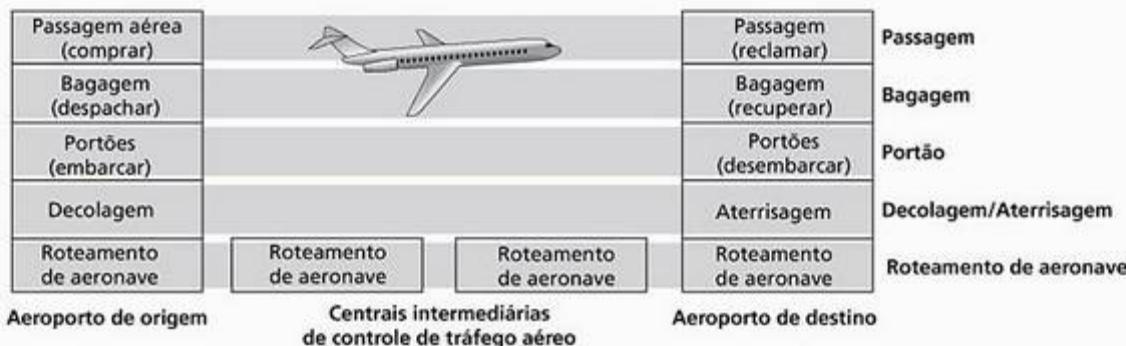


Figura 1.22 Camadas horizontais da funcionalidade de linha aérea

(embarcar e desembarcar passageiros); ela simplesmente implementaria aquela função de maneira diferente após a modificação. Para sistemas grandes e complexos que são atualizados constantemente, a capacidade de modificar a implementação de um serviço sem afetar outros componentes do sistema é outra vantagem importante da divisão em camadas.

Camadas de protocolo

Mas chega de linhas aéreas! Vamos agora voltar nossa atenção a protocolos de rede. Para prover uma estrutura para o projeto de protocolos de rede, projetistas de rede organizam protocolos — e o hardware e o software de rede que implementam os protocolos — em camadas. Cada protocolo pertence a uma das camadas, exatamente como cada função na arquitetura de linha aérea da Figura 1.22 pertence a uma camada. Novamente estamos interessados nos serviços que uma camada oferece à camada acima dela — denominado modelo de serviço de uma camada. Exatamente como no nosso exemplo da linha aérea, cada camada provê seu serviço (1) executando certas ações dentro da camada e (2) utilizando os serviços da camada diretamente abaixo dela. Por exemplo, os serviços providos pela camada n podem incluir entrega confiável de mensagens de uma extremidade da rede à outra, que pode ser implementada utilizando um serviço não confiável de entrega de mensagem fim a fim da camada $n - 1$ e adicionando funcionalidade da camada n para detectar e retransmitir mensagens perdidas.

Uma camada de protocolo pode ser implementada em software, em hardware, ou em uma combinação dos dois. Protocolos de camada de aplicação como HTTP e SMTP — quase sempre são implementados em software em sistemas finais; o mesmo acontece com protocolos de camada de transporte. Como a camada física e as camadas de enlace de dados são responsáveis pelo manuseio da comunicação por um enlace específico, normalmente são implementadas em uma placa de interface de rede (por exemplo, placas de interface Ethernet ou Wi-Fi) associadas a um determinado enlace. A camada de rede quase sempre é uma implementação mista de hardware e software. Note também que, exatamente como as funções na arquitetura em camadas da linha aérea eram distribuídas entre os vários aeroportos e centrais de controle de tráfego aéreo que compunham o sistema, também um protocolo de camada n é distribuído entre os sistemas finais, comutadores de pacote e outros componentes que formam a rede. Isto é, há sempre uma parcela de um protocolo de camada n em cada um desses componentes de rede.

O sistema de camadas de protocolos tem vantagens conceituais e estruturais. Como vimos, a divisão em camadas proporciona um modo estruturado de discutir componentes de sistemas. A modularidade facilita a atualização de componentes de sistema. Devemos mencionar, no entanto, que alguns pesquisadores e engenheiros de rede se opõem veementemente ao sistema de camadas [Wakeman, 1992]. Uma desvantagem potencial desse sistema é que uma camada pode duplicar a funcionalidade de uma camada inferior. Por exemplo, muitas pilhas de protocolos oferecem serviço de recuperação de erros na camada de enlace e também fim a fim. Uma segunda desvantagem potencial é que a funcionalidade em uma camada pode necessitar de informações (por exemplo, um valor de carimbo de tempo) que estão presentes somente em uma outra camada, o que infringe o objetivo de separação de camadas.

Quando tomados em conjunto, os protocolos das várias camadas são denominados pilha de protocolos, que é formada por cinco camadas: física, de enlace, de rede, de transporte e de aplicação, como mostra a Figura 1.23 (a). Se você verificar o sumário, verá que organizamos este livro utilizando as camadas da pilha de protocolos da Internet. Fazemos uma abordagem descendente, primeiro abordando as camadas de aplicação e, então, os processos.

Camada de aplicação

A camada de aplicação é onde residem aplicações de rede e seus protocolos. Ela inclui muitos protocolos, tais como o protocolo HTTP (que provê requisição e transferência de documentos pela Web), o SMTP (que provê transferência de mensagens de correio eletrônico) e o FTP (que provê a transferência de arquivos entre dois sistemas finais). Veremos que certas funções de rede, como a tradução de nomes fáceis de entender dados a sistemas finais da Internet (por exemplo, `gaia.cs.umass.edu`) para um endereço de rede de 32 bits, também são executadas com a ajuda de um protocolo de camada de aplicação, no caso, o sistema de nomes de domínio (*domain name system* — DNS). Veremos no Capítulo 2 que é muito fácil criar nossos próprios novos protocolos de camada de aplicação.

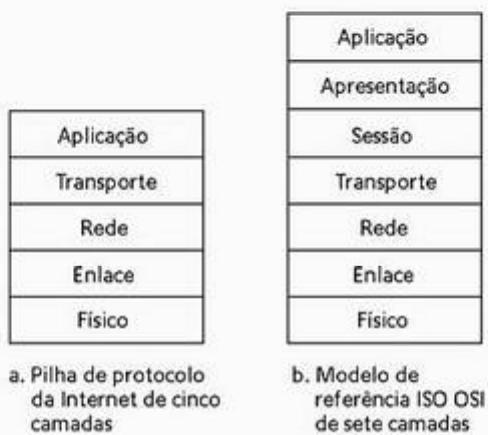


Figura 1.23 A pilha de protocolo da Internet (a) e o modelo de referência OSI (b)

Um protocolo de camada de aplicação é distribuído por diversos sistemas finais, sendo que a aplicação em um sistema final utiliza o protocolo para trocar pacotes de informação com a aplicação em outro sistema final. Denominaremos esse pacote de informação na camada de aplicação de mensagem.

Camada de transporte

A camada de transporte da Internet transporta mensagens da camada de aplicação entre os lados do cliente e servidor de uma aplicação. Há dois protocolos de transporte na Internet: TCP e UDP, e qualquer um deles pode levar mensagens de camada de aplicação. O TCP provê serviços orientados para conexão para suas aplicações. Alguns desses serviços são a entrega garantida de mensagens da camada de aplicação ao destino e controle de fluxo (isto é, compatibilização das velocidades do remetente e do receptor). O TCP também fragmenta mensagens longas em segmentos mais curtos e provê mecanismo de controle de congestionamento, de modo que uma origem regula sua velocidade de transmissão quando a rede está congestionada. O protocolo UDP provê serviço não orientado para conexão a suas aplicações. Este é um serviço econômico que fornece segurança, sem controle de fluxo e de congestionamento. Neste livro, um pacote de camada de transporte será denominado segmento.

Camada de rede

A camada de rede da Internet é responsável pela movimentação, de uma máquina para outra, de pacotes de camada de rede conhecidos como *datagramas*. O protocolo de camada de transporte da Internet (TCP ou UDP) em uma máquina de origem passa um segmento de camada de transporte e um endereço de destino à camada de rede, exatamente como você passaria ao serviço de correios uma carta com um endereço de destinatário. A camada de rede então provê o serviço de entrega do segmento à camada de transporte na máquina destinatária.

A camada de rede da Internet tem dois componentes principais. Um deles é um protocolo que define os campos no datagrama, bem como o modo como os sistemas finais e os roteadores agem nesses campos. Este é o famoso protocolo IP. Existe somente um único protocolo IP, e todos os componentes da Internet que têm uma camada de rede devem executar esse protocolo. O outro componente importante é o protocolo de roteamento que determina as rotas que os datagramas seguem entre origens e destinos. A Internet tem muitos protocolos de roteamento. Como vimos na Seção 1.3, a Internet é uma rede de redes e, dentro de uma delas, o administrador pode executar qualquer protocolo de roteamento que queira. Embora a camada de rede contenha o protocolo IP e também numerosos protocolos de roteamento, ela quase sempre é denominada simplesmente camada IP, refletindo o fato de que ele é o elemento fundamental que mantém a integridade da Internet.

Camada de enlace

A camada de rede da Internet roteia um datagrama por meio de uma série de roteadores entre a origem e o destino. Para levar um pacote de um nó (sistema final ou comutador de pacotes) ao nó seguinte na rota, a camada de rede depende dos serviços da camada de enlace. Em especial, em cada nó, a camada de rede passa o datagrama para a camada de enlace, que o entrega, ao longo da rota, ao nó seguinte, no qual o datagrama é passado da camada de enlace para a rede.

Os serviços prestados pela camada de enlace dependem do protocolo específico empregado no enlace. Alguns protocolos de camada de enlace proveem entrega garantida entre enlaces, isto é, desde o nó transmissor, passando por um único enlace, até o nó receptor. Note que esse serviço confiável de entrega é diferente do serviço de entrega garantida do TCP, que provê serviço de entrega garantida de um sistema final a outro. Exemplos de protocolos de camadas de enlace são Ethernet, WiFi e PPP (*point-to-point protocol* — protocolo ponto-a-ponto). Como datagramas normalmente precisam transitar por diversos enlaces para irem da origem ao destino, serão manuseados por diferentes protocolos de camada de enlace em diferentes enlaces ao longo de sua rota, podendo ser manuseados por Ethernet em um enlace e por PPP no seguinte. A camada de rede receberá um serviço diferente de cada um dos variados protocolos de camada de enlace. Neste livro, pacotes de camada de enlace serão denominados quadros.

Camada física

Enquanto a tarefa da camada de enlace é movimentar quadros inteiros de um elemento da rede até um elemento adjacente, a da camada física é movimentar os *bits individuais* que estão dentro do quadro de um nó para o seguinte. Os protocolos nessa camada novamente dependem do enlace e, além disso, dependem do próprio meio de transmissão do enlace (por exemplo, fios de cobre trançado ou fibra ótica monomodal). Por exemplo, a Ethernet tem muitos protocolos de camada física: um para par de fios de cobre trançado, outro para cabo coaxial, um outro para fibra e assim por diante. Em cada caso, o bit é movimentado pelo enlace de um modo diferente.

O modelo OSI

Após discutir detalhadamente a pilha de protocolo da Internet, devemos mencionar que essa não é a única pilha de protocolo existente. Particularmente, no final dos anos 1970, a Organização Internacional para Padronização (ISO) propôs que as redes de computadores fossem organizadas em, aproximadamente, sete camadas, denominadas modelo de Interconexão de Sistemas Abertos (OSI) [ISO, 2009]. O modelo OSI tomou forma quando os protocolos que iriam se tornar protocolos da Internet estavam em amadurecimento; e eram um dos muitos conjuntos de protocolos em desenvolvimento; na verdade, os inventores do modelo OSI original provavelmente não tinham a Internet em mente ao criá-lo. No entanto, no final dos anos 1970, muitos cursos universitários e de treinamento obtiveram conhecimentos sobre a exigência do ISO e organizaram cursos voltados para o modelo de sete camadas. Em razão de seu impacto precoce na educação de redes, o modelo de sete camadas continua presente em alguns livros sobre redes e em cursos de treinamento.

As sete camadas do modelo de referência OSI, mostradas na Figura 1.23 (b), são: camada de aplicação, camada de apresentação, camada de sessão, camada de transporte, camada de rede, camada de enlace e camada física. A função de cinco dessas camadas é a mesma que seus componentes da Internet. Desse modo, vamos considerar as duas camadas adicionais presentes no modelo de referência OSI — a camada de apresentação e a camada de sessão. O papel da camada de apresentação é prover serviços que permitam que as aplicações de comunicação interpretem o significado dos dados trocados. Entre esses serviços estão a compressão de dados e a codificação de dados (o que não precisa de explicação), assim como a descrição de dados (que, como veremos no Capítulo 9, livram as aplicações da preocupação com o formato interno onde os dados estão sendo descritos/armazenados — formato esse que podem se diferenciar de um computador para o outro). A camada de sessão provê a delimitação e sincronização da troca de dados, incluindo os meios de construir um esquema de pontos de verificação e de recuperação.

O fato de a Internet ser desprovida de duas camadas encontradas no modelo de referência OSI faz surgir duas questões: os serviços fornecidos por essas camadas são irrelevantes? E se uma aplicação precisar de um desses

serviços? A resposta da Internet para essas perguntas é a mesma — depende do criador da aplicação. Cabe ao criador da aplicação decidir se um serviço é importante, e se o serviço *for* importante, cabe ao criador da aplicação desenvolver essa função para ela.

1.5.2 Mensagens, segmentos, datagramas e quadros

A Figura 1.24 apresenta o caminho físico que os dados percorrem: para baixo na pilha de protocolos de um sistema final emissor, para cima e para baixo nas pilhas de protocolos de um comutador de camada de enlace interveniente e de um roteador e então para cima na pilha de protocolos do sistema final receptor. Como discutiremos mais adiante neste livro, ambos, comutadores de camada de enlace e roteadores, são comutadores de pacotes. De modo semelhante a sistemas finais, roteadores e comutadores de camada de enlace organizam seu hardware e software de rede em camadas. Mas roteadores e comutadores de camada de enlace não implementam *todas* as camadas da pilha de protocolos; normalmente implementam apenas as camadas de baixo. Como mostra a Figura 1.24, comutadores de camada de enlace implementam as camadas 1 e 2; roteadores implementam as camadas 1, 2 e 3. Isso significa, por exemplo, que roteadores da Internet são capazes de implementar o protocolo IP (da camada 3), mas comutadores de camada de enlace não. Veremos mais adiante que, embora não reconheçam endereços IP, comutadores de camada de enlace são capazes de reconhecer endereços de camada 2, tais como endereços da Ethernet. Note que sistemas finais implementam todas as cinco camadas, o que é consistente com a noção de que a arquitetura da Internet concentra sua complexidade na periferia da rede.

A Figura 1.24 também ilustra o importante conceito de encapsulamento. Uma mensagem de camada de aplicação na máquina emissor (M na Figura 1.24) é passada para a camada de transporte. No caso mais simples, esta pega a mensagem e anexa informações adicionais (denominadas informações de cabeçalho de camada de transporte, H_t na Figura 1.24) que serão usadas pela camada de transporte do lado receptor. A mensagem de camada de aplicação e as informações de cabeçalho da camada de transporte, juntas, constituem o segmento da camada de transporte, que encapsula a mensagem da camada de aplicação. As informações adicionadas podem

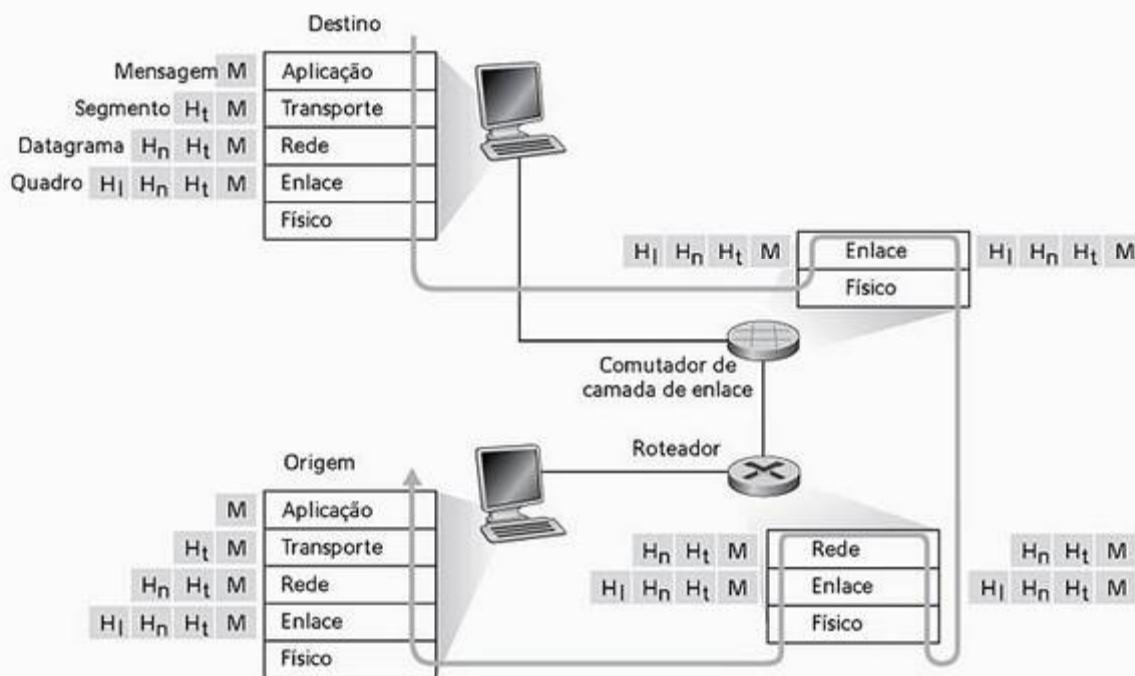


Figura 1.24 Hospedeiros, roteadores e comutadores de camada de enlace; cada um contém um conjunto diferente de camadas, refletindo suas diferenças em funcionalidade

incluir dados que habilitem a camada de transporte do lado do receptor a entregar a mensagem à aplicação apropriada, além de bits de detecção de erro que permitem que o receptor determine se os bits da mensagem foram modificados em trânsito. A camada de transporte então passa o segmento à camada de rede, que adiciona informações de cabeçalho de camada de rede (H_n na Figura 1.24), como endereços de sistemas finais de origem e de destino, criando um datagrama de camada de rede. Este é então passado para a camada de enlace, que (é claro!), adicionará suas próprias informações de cabeçalho e criará um quadro de camada de enlace. Assim, vemos que, em cada camada, um pacote possui dois tipos de campos: campos de cabeçalho e um campo de carga útil. A carga útil é normalmente um pacote da camada acima.

Uma analogia útil que podemos usar aqui é o envio de um memorando entre escritórios de uma empresa pelo correio de uma filial corporativa a outra. Suponha que Alice, que está em uma filial, queira enviar um memorando a Bob, que está na outra filial. O memorando *representa* uma mensagem da camada de aplicação. Alice coloca o memorando em um envelope de correspondência interna em cuja face são escritos o nome e o departamento de Bob. O envelope de correspondência interna representa o segmento da camada de aplicação — contém as informações de cabeçalho (o nome de Bob e seu departamento) e encapsula a mensagem de camada de aplicação (o memorando). Quando a central de correspondência do escritório emissor recebe o envelope, ele é colocado dentro de outro envelope adequado para envio pelo correio. A central de correspondência emissora também escreve o endereço postal do remetente e do destinatário no envelope postal. Neste ponto, o envelope postal é análogo ao datagrama — encapsula o segmento de camada de transporte (o envelope de correspondência interna), que por sua vez encapsula a mensagem original (o memorando). O correio entrega o envelope postal à central de correspondência do escritório destinatário. Nesse local, o processo de reencapsulamento se inicia. A central de correspondência retira o memorando e o encaminha a Bob. Este, finalmente, abre o envelope e retira o memorando. O processo de encapsulamento pode ser mais complexo do que o descrito acima. Por exemplo, uma mensagem grande pode ser dividida em vários segmentos de camada de transporte (que também podem ser divididos em vários datagramas de camada de rede). Na extremidade receptora, cada segmento deve ser reconstruído a partir dos datagramas que o compõem.

1.6 Redes sob ameaça

A Internet se tornou essencial para muitas instituições hoje, incluindo empresas grandes e pequenas, universidades e órgãos do governo. Muitas pessoas também contam com a Internet para suas atividades profissionais, sociais e pessoais. Mas atrás de toda essa utilidade e entusiasmo, existe o lado negro, um lado no qual “vilões” tentam causar problemas em nosso cotidiano danificando nossos computadores conectados à Internet, violando nossa privacidade e tornando inoperantes os serviços da Internet dos quais dependemos [Skoudis, 2006].

A área de segurança de rede abrange como esses vilões podem ameaçar as redes de computadores e como nós, futuros especialistas no assunto, podemos defender a rede contra essas ameaças ou, melhor ainda, criar novas arquiteturas imunes a tais ameaças em primeiro lugar. Dadas a frequência e a variedade das ameaças existentes, bem como o perigo de novos e mais destrutivos futuros ataques, a segurança de rede se tornou, recentemente, um assunto principal na área de redes de computadores. Um dos objetivos deste livro é trazer as questões de segurança de rede para primeiro plano. Iniciaremos nossa discussão sobre redes de computadores nesta seção, com uma breve descrição de alguns dos ataques mais predominantes e prejudiciais na Internet de hoje. Então, à medida que retratarmos em detalhes as diversas tecnologias de redes de computadores e protocolos nos capítulos seguintes, analisaremos as diversas questões relacionadas à segurança associadas a essas tecnologias e protocolos. Finalmente, no Capítulo 8, munidos com nosso know-how em redes de computadores e protocolos da Internet recém-adquirido, estudaremos a fundo como as redes de computadores podem ser defendidas contra ameaças, ou projetadas e operadas para impossibilitá-las.

Visto que ainda não temos o know-how em rede de computadores e em protocolos da Internet, começaremos com uma análise de alguns dos atuais problemas mais predominantes relacionados à segurança. Isto irá aguçar nosso apetite para mais discussões importantes nos capítulos futuros. Começamos com a pergunta, o que pode dar errado? Como as redes de computadores são frágeis? Quais são alguns dos tipos de ameaças mais predominantes hoje?

Os vilões podem colocar "malware" em seu hospedeiro através da Internet

Conectamos aparelhos à Internet porque queremos receber/enviar dados de/para a Internet. Isso inclui todos os tipos de recursos vantajosos, como páginas da Web, mensagens de e-mail, MP3, chamadas telefônicas, vídeo em tempo real, resultados de mecanismo de busca etc. Porém, infelizmente, junto com esses recursos vantajosos aparecem os maliciosos — conhecidos conjuntamente como malware — que também podem entrar e infectar nossos aparelhos. Uma vez que o malware infecta nosso aparelho, ele é capaz de fazer coisas tortuosas, como apagar nossos arquivos; instalar spyware que coleta nossas informações particulares, como nosso número de cartão de crédito, senhas e combinação de teclas, e as envia (através da Internet, é claro!) de volta aos vilões. Nosso hospedeiro comprometido pode estar, também, envolvido em uma rede de milhares de aparelhos comprometidos, conhecidos como "botnet", o qual é controlado e influenciado pelos vilões para distribuição de spams ou ataques de recusa de serviço distribuídos (que serão brevemente discutidos) contra hospedeiros direcionados.

Muitos malwares existentes hoje são autorreprodutivos: uma vez que infecta um hospedeiro, a partir deste, ele faz a busca por entradas em outros hospedeiros através da Internet, e a partir de hospedeiros recém-infectados, ele faz a busca por entrada em mais hospedeiros. Desta maneira, o malware autorreprodutivo pode se disseminar rapidamente. Por exemplo, o número de aparelhos infectados pelo worm 2003 Saphire/Slammer dobrou a cada 8,5 segundos nos primeiros minutos após seu ataque, infectando mais de 90 por cento dos hospedeiros frágeis em 10 minutos [Moore, 2003]. O malware pode se espalhar na forma de vírus, worms ou cavalo de Troia [Skoudis, 2004]. Os vírus são malwares que necessitam de uma interação do usuário para infectar seu aparelho. O exemplo clássico é um anexo de e-mail contendo um código executável malicioso. Se o usuário receber e abrir tal anexo, o malware será executado em seu aparelho. Geralmente, tais vírus de e-mail se autorreproduzem: uma vez executado, o vírus pode enviar uma mensagem idêntica, com um anexo malicioso idêntico, para, por exemplo, todos os contatos da lista de endereços do usuário. Worms (como o Slammer) são malwares capazes de entrar em um aparelho sem qualquer interação do usuário. Por exemplo, um usuário pode estar executando uma aplicação de rede frágil para a qual um atacante pode enviar um malware. Em alguns casos, sem a intervenção do usuário, a aplicação pode aceitar o malware da Internet e executá-lo, criando um worm. Este, no aparelho recém-infectado, então, varre a Internet em busca de outros hospedeiros que estejam executando a mesma aplicação de rede frágil. Ao encontrar outros hospedeiros frágeis, ele envia uma cópia de si mesmo para eles. Por fim, um cavalo de Troia é uma parte oculta de um software funcional. Hoje, o malware é persuasivo e é caro para se criar uma proteção. À medida que trabalhar com este livro, sugerimos que pense na seguinte questão: O que os projetistas de computadores podem fazer para proteger os aparelhos que utilizam a Internet de ameaças de malware?

Os vilões podem atacar servidores e infraestrutura de redes

Um amplo grupo de ameaças à segurança pode ser classificado como ataques de recusa de serviços (DoS). Como o nome sugere, um ataque DoS torna uma rede, hospedeiro ou outra parte da infraestrutura inutilizável por usuários verdadeiros. Servidores da Web, de e-mails e DNS (discutidos no Capítulo 2), e redes institucionais podem estar sujeitos aos ataques DoS. Na Internet, esses ataques são extremamente comuns, com milhares deles ocorrendo todo ano [Moore, 2001; Mirkovic, 2005]. A maioria dos ataques DoS na Internet podem ser divididos em três categorias:

Ataque de vulnerabilidade. Envolve o envio de mensagens perfeitas a uma aplicação vulnerável ou a um sistema operacional sendo executado em um hospedeiro direcionado. Se a sequência correta de pacotes é enviada a uma aplicação vulnerável ou a um sistema operacional, o serviço pode parar ou, pior, o hospedeiro pode pisar.

Inundação na largura de banda. O atacante envia um grande número de pacotes ao hospedeiro direcionado — tantos pacotes que o enlace de acesso do alvo se entope, impedindo os pacotes legítimos de alcançarem o servidor.

Inundação na conexão. O atacante estabelece um grande número de conexões TCP semiabertas ou abertas (as conexões TCP são discutidas no Capítulo 3) no hospedeiro-alvo. O hospedeiro pode ficar tão atolado com essas conexões falsas que para de aceitar conexões legítimas.

Vamos agora explorar mais detalhadamente o ataque de inundação na largura de banda. Lembrando de nossa análise sobre atraso e perda na seção 1.4.2, é evidente que se o servidor possui uma taxa de acesso R bps, então o atacante precisará enviar tráfego a uma taxa de, aproximadamente, R bps para causar dano. Se R for muito grande, uma fonte de ataque única pode não ser capaz de gerar tráfego suficiente para prejudicar o servidor. Além disso, se todo o tráfego provier de uma fonte única, um roteador upstream pode conseguir detectar o ataque e bloquear todo o tráfego da fonte antes que ele se aproxime do servidor. Em um ataque DoS distribuído (DDoS), ilustrado na Figura 1.25, o atacante controla múltiplas fontes que sobrecarregam o alvo. Com tal abordagem, a taxa de tráfego agregada por todas as fontes controladas precisa ser, aproximadamente, R para incapacitar o serviço. Os ataques DDoS que potencializam botnets com centenas de hospedeiros comprometidos são uma ocorrência comum hoje em dia [Mirkovic, 2005]. Os ataques DDoS são muito mais difíceis de detectar e de prevenir do que um ataque DoS de um único hospedeiro.

Os vilões podem analisar pacotes

Muitos usuários hoje acessam à Internet por meio de aparelhos sem fio, como laptops conectados à tecnologia Wi-Fi ou aparelhos portáteis com conexões à Internet via telefone celular (abordado no Capítulo 6). Enquanto o acesso onipresente à Internet é extremamente conveniente e disponibiliza novas aplicações sensacionais aos usuários móveis, ele também cria uma grande vulnerabilidade de segurança — posicionando um receptor passivo nas proximidades do transmissor sem fio, o receptor pode obter uma cópia de cada pacote transmitido! Esses pacotes podem conter todo tipo de informações confidenciais, incluindo senhas, número de inscrição da previdência social, segredos comerciais e mensagens pessoais. Um receptor passivo que grava uma cópia de cada pacote que passa é denominado analisador de pacote.

Os analisadores também podem estar distribuídos em ambientes de conexão com fio. Nesses ambientes, como em muitas Ethernet LANs, um analisador de pacote pode obter cópias de todos os pacotes enviados pela LAN. Como descrito na Seção 1.2, as tecnologias de acesso a cabo também transmitem pacotes e são, dessa forma, vulneráveis à análise. Além disso, um vilão que quer ganhar acesso ao roteador de acesso de uma instituição ou enlace de acesso para a Internet pode instalar um analisador que faça uma cópia de cada pacote que vai para/de a empresa. Os pacotes farejados podem, então, ser analisados off-line em busca de informações confidenciais.

O software para analisar pacotes está disponível gratuitamente em diversos sites da Internet e em produtos comerciais. Professores que ministram um curso de redes passam exercícios que envolvem a composição de um programa de reconstrução de dados da camada de aplicação e um programa analisador de pacotes.

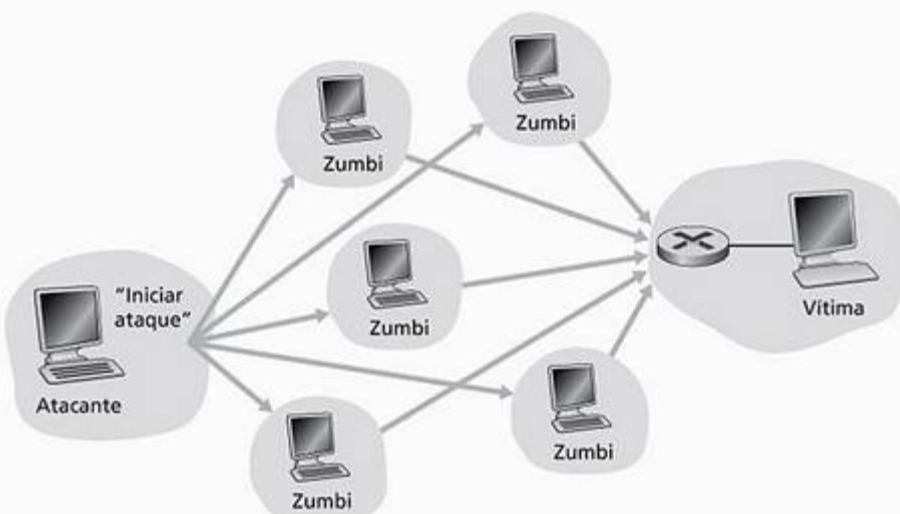


Figura 1.25 Um ataque de recusa de serviço distribuído (DDoS)

Como os analisadores de pacote são passivos — ou seja, eles não introduzem pacotes no canal — eles são difíceis de detectar. Portanto, quando enviamos pacotes para um canal sem fio, devemos aceitar a possibilidade de que alguém possa estar copiando nossos pacotes. Como você deve ter imaginado, uma das melhores defesas contra a análise de pacote envolve a criptografia, que será analisada no Capítulo 8, já que se aplica à segurança de rede.

Os vilões podem se passar por alguém de sua confiança

Por incrível que pareça, é extremamente fácil (você saberá como conforme ler este livro!) criar um pacote com um endereço de fonte arbitrário, conteúdo de pacote e um endereço de destino e, então, transmitir esse pacote feito à mão para a Internet, que, obedientemente, o encaminhará para seu destino. Imagine que um receptor inocente (digamos um roteador da Internet) que recebe tal pacote, acredita que a fonte (falsa) seja confiável e então executa um comando integrado aos conteúdos do pacote (digamos que modifica sua base de encaminhamento). A habilidade de introduzir pacotes na Internet com uma fonte falsa de endereço é conhecida como IP spoofing, e é uma das muitas maneiras pelas quais o usuário pode se passar por outro.

Para resolver esse problema, precisaremos de uma comprovação da fonte, ou seja, um mecanismo que nos permitirá determinar com certeza se uma mensagem se origina de onde pensamos. Mais uma vez, sugerimos que pense em como isso pode ser feito em aplicações de rede e protocolos à medida que avança sua leitura pelos capítulos deste livro. Exploraremos mais mecanismos para comprovação da fonte no Capítulo 8.

Os vilões podem alterar ou excluir mensagens

Encerramos esta breve análise de ataques na rede descrevendo os ataques *man-in-the-middle* (homem no meio). Nessa categoria de ataques, o vilão está infiltrado no percurso da comunicação entre duas entidades comunicantes. Vamos chamar essas entidades de Alice e Bob, que podem ser seres humanos reais ou entidades de rede como dois roteadores ou dois servidores de e-mail. O vilão pode ser, por exemplo, um roteador comprometido no percurso da comunicação, ou um módulo de software residente em um dos hospedeiros finais em uma camada inferior da pilha de protocolo. No ataque *man-in-the-middle*, o vilão não somente possui a possibilidade de analisar todos os pacotes que passam entre Bob e Alice, como também pode introduzir, alterar ou excluir pacotes. No jargão da segurança de rede, um ataque *man-in-the-middle* pode comprometer a integridade dos dados enviados entre Alice e Bob. Assim como veremos no Capítulo 8, os mecanismos que oferecem sigilo (proteção contra análise) e autenticação da origem (permitindo que o receptor verifique com certeza o gerador da mensagem) não necessariamente oferecem integridade dos dados.

Ao encerrar esta seção, deve-se considerar como a Internet se tornou um local inseguro, em primeiro lugar. A resposta resumidamente é que a Internet foi, a princípio, criada para ser assim, baseada no modelo de “um grupo de usuários de confiança mútua ligados a uma rede transparente” [Blumenthal, 2001] — um modelo no qual (por definição) não há necessidade de segurança. Muitos aspectos da arquitetura inicial da Internet refletem profundamente essa noção de confiança mútua. Por exemplo, a capacidade de um usuário enviar um pacote a qualquer outro é mais uma falha do que um recurso solicitado/concedido, e acredita-se piamente na identidade do usuário em vez de ela ser confirmada automaticamente.

Mas a Internet de hoje certamente não envolve “usuários de confiança mútua”. Contudo, os usuários atuais ainda precisam se comunicar mesmo quando não confiam um no outro, podem se comunicar anonimamente, podem se comunicar indiretamente através de terceiros (por exemplo, Web caches, que serão estudados no Capítulo 2, ou agentes móveis para assistência, que serão estudados no Capítulo 6), e podem desconfiar do hardware, software e até mesmo do ar pelo qual eles se comunicam. Temos agora muitos desafios relacionados à segurança perante nós à medida que prosseguimos com o livro: devemos procurar por proteção contra a análise, disfarce da origem, ataques *man-in-the-middle*, ataques DDoS, malware e mais. Devemos manter em mente que a comunicação entre usuários de confiança mútua é mais uma exceção do que uma regra.

Seja bem-vindo ao mundo da moderna rede de computadores!

1.7 História das redes de computadores e da Internet

Da seção 1.1 à 1.6, apresentamos um panorama da tecnologia de redes de computadores e da Internet. Agora, você já deve saber o suficiente para impressionar sua família e amigos. Contudo, se realmente quiser ser o maior sucesso do próximo coquetel, você deve rechear seu discurso com pérolas da fascinante história da Internet [Segaller, 1998].

1.7.1 Desenvolvimento da comutação de pacotes: 1961-1972

Os primeiros passos da disciplina de redes de computadores e da Internet podem ser traçados desde o início da década de 1960, quando a rede telefônica era a rede de comunicação dominante no mundo inteiro. Lembre-se de que na Seção 1.3 dissemos que a rede de telefonia usa comutação de circuitos para transmitir informações entre uma origem e um destino — uma escolha acertada, já que a voz é transmitida a uma taxa constante entre a origem e o destino. Dada a importância cada vez maior (e o alto custo) dos computadores no início da década de 1960 e o advento de computadores com multiprogramação (*time-sharing*), nada seria mais natural (agora que temos uma visão perfeita do passado) do que considerar a questão de como interligar computadores para que pudessem ser compartilhados entre usuários distribuídos em localizações geográficas diferentes. O tráfego gerado por esses usuários provavelmente era intermitente, por *rajadas* — períodos de atividade, como o envio de um comando a um computador remoto, seguidos de períodos de inatividade, como a espera por uma resposta ou o exame de uma resposta recebida.

Três grupos de pesquisa ao redor do mundo, sem que nenhum tivesse conhecimento do trabalho do outro [Leiner, 1998], começaram a inventar a comutação de pacotes como uma alternativa poderosa e eficiente à de circuitos. O primeiro trabalho publicado sobre técnicas de comutação de pacotes foi o de Leonard Kleinrock [Kleinrock, 1961, 1964], que, naquela época, era um doutorando do MIT. Usando a teoria de filas, o trabalho de Kleinrock demonstrou, com elegância, a eficácia da abordagem da comutação de pacotes para fontes de tráfego intermitentes (*em rajadas*). Em 1964, Paul Baran [Baran, 1964], do Rand Institute, começou a investigar a utilização de comutação de pacotes na transmissão segura de voz pelas redes militares, ao mesmo tempo que Donald Davies e Roger Scantlebury desenvolviam suas ideias sobre esse assunto no National Physical Laboratory, na Inglaterra.

Os trabalhos desenvolvidos no MIT, no Rand Institute e no National Physical Laboratory foram os alicerces do que hoje é a Internet. Mas a Internet também tem uma longa história de atitudes do tipo “construir e demonstrar”, que também data do início da década de 1960. J. C. R. Licklider [DEC, 1990] e Lawrence Roberts, ambos colegas de Kleinrock no MIT, foram adiante e lideraram o programa de ciência de computadores na ARPA (Advanced Research Projects Agency — Agência de Projetos de Pesquisa Avançada), nos Estados Unidos. Roberts publicou um plano geral para a ARPAnet [Roberts, 1967], a primeira rede de computadores por comutação de pacotes e uma ancestral direta da Internet pública de hoje. Os primeiros comutadores de pacotes eram conhecidos como processadores de mensagens de interface (*interface message processors* — IMPs), e o contrato para a fabricação desses comutadores foi entregue à empresa BBN. Em 1969, no Dia do Trabalho nos Estados Unidos, foi instalado o primeiro IMP na UCLA (Universidade da Califórnia em Los Angeles) sob a supervisão de Kleinrock. Logo em seguida foram instalados três IMPs adicionais no Stanford Research Institute (SRI), na Universidade da Califórnia em Santa Bárbara e na Universidade de Utah (Figura 1.26).

O incipiente precursor da Internet tinha quatro nós no final de 1969. Kleinrock recorda que a primeiríssima utilização da rede foi fazer um login remoto entre a UCLA e o SRI, derrubando o sistema [Kleinrock, 2004].

Em 1972, a ARPAnet tinha aproximadamente 15 nós e foi apresentada publicamente pela primeira vez por Robert Kahn na Conferência Internacional sobre Comunicação por Computadores (International Conference on Computer Communications) daquele ano. O primeiro protocolo fim a fim entre sistemas finais da ARPAnet, conhecido como protocolo de controle de rede (*network-control protocol* — NCP), estava concluído [RFC 001] e a partir desse momento a escrita de aplicações tornou-se possível. Em 1972, Ray Tomlinson, da BBN, escreveu o primeiro programa de e-mail.

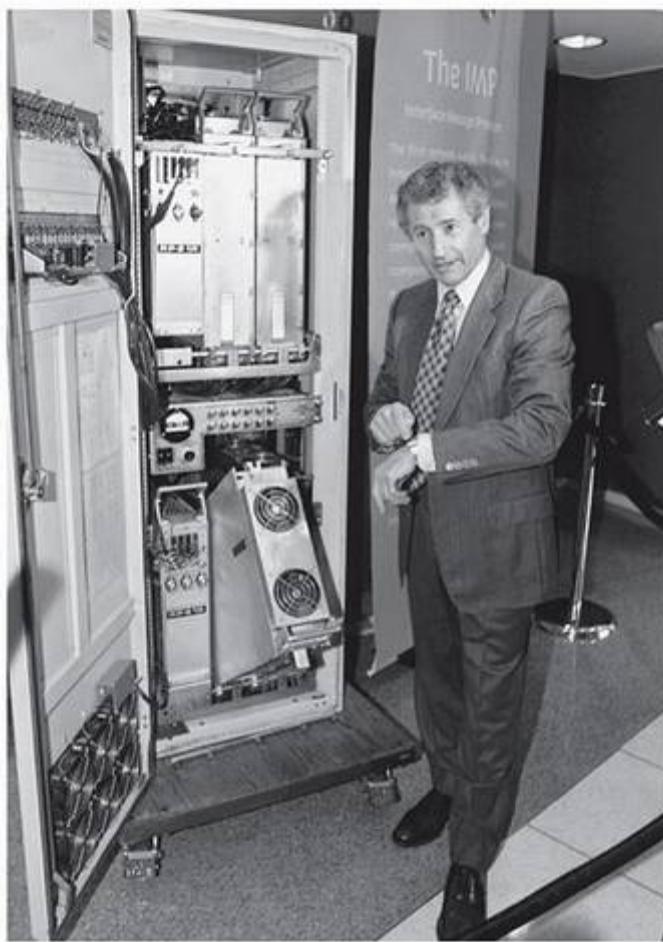


Figura 1.26 Um dos primeiros processadores de mensagens de interface (IMP) e L. Kleinrock. (Mark J. Terrill, AP/Wide World Photos)

1.7.2 Redes proprietárias e trabalho em rede: 1972-1980

A ARPAnet inicial era um rede isolada, fechada. Para se comunicar com uma máquina da ARPAnet, era preciso estar ligado a um outro IMP dessa rede. Do início a meados de 1970, surgiram novas redes independentes de comutação de pacotes:

- ALOHAnet, uma rede de micro-ondas ligando universidades das ilhas do Havaí [Abramson, 1970], bem como as redes de pacotes por satélite [RFC 829] e por rádio [Kahn, 1978] da DARPA [Kahn, 1978];
- Telenet, uma rede comercial de comutação de pacotes da BBN fundamentada na tecnologia ARPAnet;
- Cyclades, uma rede de comutação de pacotes pioneira na França, montada por Louis Pouzin [Think, 2002];
- redes de tempo compartilhado como a Tymnet e a rede GE Information Services, entre outras que surgiram no final da década de 1960 e início da década de 1970 [Schwartz, 1977];
- rede SNA da IBM (1969-1974), cujo trabalho comparava-se ao da ARPAnet [Schwartz, 1977].

O número de redes estava crescendo. Hoje, com perfeita visão do passado, podemos perceber que aquela era a hora certa para desenvolver uma arquitetura abrangente para conectar redes. O trabalho pioneiro de interconexão de redes, sob o patrocínio da DARPA (Defense Advanced Research Projects Agency — Agência de Projetos de Pesquisa Avançada de Defesa), criou em essência *uma rede de redes* e foi realizado por Vinton Cerf e Robert Kahn [Cerf, 1974]; o termo *internettiong* foi cunhado para descrever esse trabalho.

Esses princípios de arquitetura foram incorporados ao TCP. As primeiras versões desse protocolo, contudo, eram muito diferentes do TCP de hoje. Aquelas versões combinavam uma entrega sequencial confiável de dados via retransmissão por sistema final (que ainda faz parte do TCP de hoje) com funções de envio (que hoje são desempenhadas pelo IP). As primeiras experiências com o TCP, combinadas com o reconhecimento da importância de um serviço de transporte fim a fim não confiável, sem controle de fluxo, para aplicações como voz em pacotes, levaram à separação entre IP e TCP e ao desenvolvimento do protocolo UDP. Os três protocolos fundamentais da Internet que temos hoje — TCP, UDP e IP — estavam conceitualmente disponíveis no final da década de 1970.

Além das pesquisas sobre a Internet realizadas pela DARPA, muitas outras atividades importantes relacionadas ao trabalho em rede estavam em curso. No Havaí, Norman Abramson estava desenvolvendo a ALOHAnet, uma rede de pacotes por rádio que permitia que vários lugares remotos das ilhas havaianas se comunicassem entre si. O ALOHA [Abramson, 1970] foi o primeiro protocolo de acesso múltiplo que permitiu que usuários distribuídos em diferentes localizações geográficas compartilhassem um único meio de comunicação broadcast (uma frequência de rádio). O trabalho de Abramson sobre protocolo de múltiplo acesso foi aprimorado por Metcalfe e Boggs com o desenvolvimento do protocolo Ethernet [Metcalfe, 1976] para redes compartilhadas de transmissão broadcast por fio; veja a Figura 1.27.

O interessante é que o protocolo Ethernet de Metcalfe e Boggs foi motivado pela necessidade de conectar vários PCs, impressoras e discos compartilhados [Perkins, 1994]. Há 25 anos, bem antes da revolução do PC e da explosão das redes, Metcalfe e Boggs estavam lançando as bases para as LANs de PCs de hoje. A tecnologia Ethernet representou uma etapa importante para o trabalho em redes interconectadas. Cada rede Ethernet local era, em si, uma rede, e, à medida que o número de LANs aumentava, a necessidade de interconectar essas redes foi se tornando cada vez mais importante. Discutiremos detalhadamente a tecnologia Ethernet, ALOHA e outras tecnologias de LAN no Capítulo 5.

1.7.3 Proliferação de redes: 1980-1990

Ao final da década de 1970, aproximadamente 200 máquinas estavam conectadas à ARPAnet. Ao final da década de 1980, o número de máquinas ligadas à Internet pública, uma confederação de redes muito parecida com a Internet de hoje, alcançaria cem mil. A década de 1980 seria uma época de formidável crescimento.

Grande parte daquele crescimento foi consequência de vários esforços distintos para criar redes de computadores para interligar universidades. A BITNET processava e-mails e fazia transferência de arquivos entre diversas universidades do nordeste dos Estados Unidos. A CSNET (computer science network — rede da ciência de com-

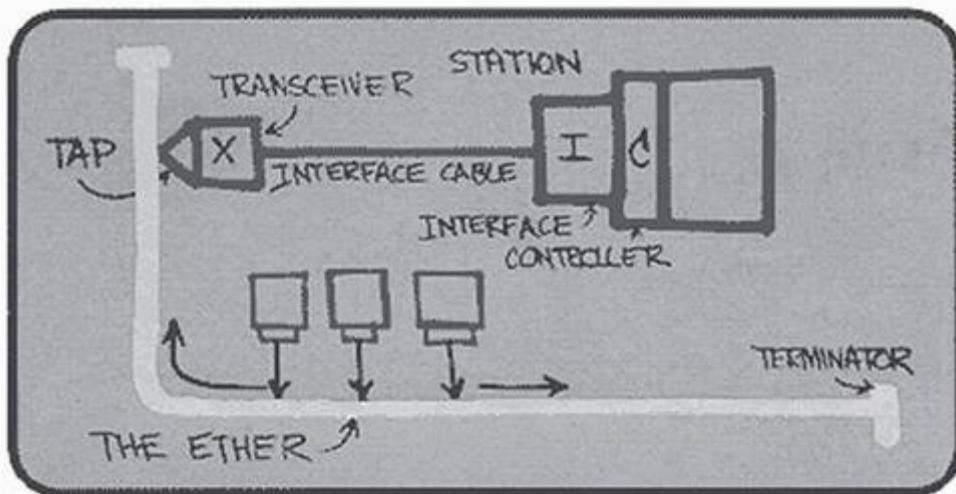


Figura 1.27 A concepção original de Metcalfe para a Ethernet

putadores) foi formada para interligar pesquisadores de universidades que não tinham acesso à ARPAnet. Em 1986, foi criada a NSFNET para prover acesso a centros de supercomputação patrocinados pela NSF. Partindo de uma velocidade inicial de 56 kbps, ao final da década o backbone da NSFNET estaria funcionando a 1,5 Mbps e servindo como backbone primário para a interligação de redes regionais.

Na comunidade da ARPAnet, já estavam sendo encaixados muitos dos componentes finais da arquitetura da Internet de hoje. No dia 1º de janeiro de 1983, o TCP/IP foi adotado oficialmente como o novo padrão de protocolo de máquinas para a ARPAnet (em substituição ao protocolo NCP). Devido à importância do evento, o dia da transição do NCP para o TCP/IP [RFC 801] foi marcado com antecedência — a partir daquele dia todas as máquinas tiveram de adotar o TCP/IP. No final da década de 1980, foram agregadas importantes extensões ao TCP para implementação do controle de congestionamento baseado em hospedeiros [Jacobson, 1988]. Também foi desenvolvido o sistema de nomes de domínios (DNS) utilizado para mapear nomes da Internet fáceis de entender (por exemplo, *gaia.cs.umass.edu*) para seus endereços IP de 32 bits [RFC 1034]. Paralelamente ao desenvolvimento da ARPAnet (que em sua maior parte deve-se aos Estados Unidos), no início da década de 1980 os franceses lançaram o projeto Minitel, um plano ambicioso para levar as redes de dados para todos os lares. Patrocinado pelo governo francês, o sistema Minitel consistia em uma rede pública de comutação de pacotes (baseada no conjunto de protocolos X.25, que usava circuitos virtuais), servidores Minitel e terminais baratos com modems de baixa velocidade embutidos. O Minitel transformou-se em um enorme sucesso em 1984, quando o governo francês forneceu, gratuitamente, um terminal para toda residência francesa que quisesse. O sistema Minitel incluía sites de livre acesso — como o da lista telefônica — e também sites particulares, que cobravam uma taxa de cada usuário baseada no tempo de utilização. No seu auge, em meados de 1990, o Minitel oferecia mais de 20 mil serviços, que iam desde home banking até bancos de dados especializados para pesquisa. Era usado por mais de 20 por cento da população da França, gerava receita de mais de um bilhão de dólares por ano e criou dez mil empregos. Estava presente em grande parte dos lares franceses dez anos antes de a maioria dos norte-americanos ouvir falar de Internet.

1.7.4 A explosão da Internet: a década de 1990

A década de 1990 estreou com vários eventos que simbolizaram a evolução contínua e a comercialização iminente da Internet. A ARPAnet, a progenitora da Internet, deixou de existir. Durante a década de 1980, a MILNET e a Defense Data Network (Rede de Dados de Defesa) cresceram e passaram a carregar a maior parte do tráfego do Departamento de Defesa dos Estados Unidos e a NSFNET começou a servir como uma rede de backbone conectando redes regionais nos Estados Unidos com nacionais no exterior. Em 1991, a NSFNET extinguia as restrições que impunha à sua utilização com finalidades comerciais, mas, em 1995, perderia seu mandato quando o tráfego de backbone da Internet passou a ser carregado por provedores de serviços de Internet.

O principal evento da década de 1990, no entanto, foi o surgimento da World Wide Web, que levou a Internet para os lares e as empresas de milhões de pessoas no mundo inteiro. A Web serviu também como plataforma para a habilitação e a disponibilização de centenas de novas aplicações, inclusive negociação de ações e serviços bancários on-line, serviços multimídia em tempo real e serviços de recuperação de informações. Para um breve histórico dos primórdios da Web, consulte [W3C, 1995].

A Web foi inventada no CERN (European Center for Nuclear Physics — Centro Europeu para Física Nuclear) por Tim Berners-Lee entre 1989 e 1991 [Berners-Lee, 1989], com base em ideias originadas de trabalhos anteriores sobre hipertexto realizados por Bush [Bush, 1945], na década de 1940, e por Ted Nelson [Ziff-Davis, 1998], na década de 1960. Berners-Lee e seus companheiros desenvolveram versões iniciais de HTML, HTTP, um servidor para a Web e um browser — os quatro componentes fundamentais da Web. Os browsers originais do CERN ofereciam apenas uma interface de linha de comando. Perto do final de 1992 havia aproximadamente 200 servidores Web em operação, e esse conjunto de servidores era apenas uma amostra do que estava por vir. Nessa época, vários pesquisadores estavam desenvolvendo browsers da Web com interfaces GUI (graphical user interface — interface gráfica de usuário), entre eles Marc Andreessen, que liderou o desenvolvimento do popular browser Mosaic. Em 1994, Marc Andreessen e Jim Clark formaram a Mosaic Communications, que mais

tarde se transformou na Netscape Communications Corporation [Cusumano, 1998; Quittner, 1998]. Em 1995, estudantes universitários estavam usando browsers Mosaic e Netscape para navegar na Web diariamente. Nessa época, empresas — grandes e pequenas — começaram a operar servidores Web e a realizar transações comerciais pela Web. Em 1996, a Microsoft começou a fabricar browsers, dando início à guerra dos browsers entre Netscape e Microsoft, vencida pela última alguns anos mais tarde.

A segunda metade da década de 1990 foi um período de tremendo crescimento e inovação para a Internet, com grandes corporações e milhares de novas empresas criando produtos e serviços para a Internet. O correio eletrônico pela Internet (e-mail) continuou a evoluir com leitores ricos em recursos provendo agendas de endereços, anexos, hot links e transporte de multimídia. No final do milênio a Internet dava suporte a centenas de aplicações populares, entre elas quatro de enorme sucesso:

- e-mail, incluindo anexos e correio eletrônico com acesso pela Web;
- a Web, incluindo navegação pela Web e comércio pela Internet;
- serviço de mensagem instantânea, com listas de contato, cujo pioneiro foi o ICQ;
- compartilhamento peer-to-peer de arquivos MP3, cujo pioneiro foi o Napster.

O interessante é que as duas primeiras dessas aplicações de sucesso arrasador vieram da comunidade de pesquisas, ao passo que as duas últimas foram criadas por alguns jovens empreendedores.

No período de 1995 a 2001, a Internet realizou uma viagem vertiginosa nos mercados financeiros. Antes mesmo de se mostrarem lucrativas, centenas de novas empresas da Internet faziam suas ofertas públicas iniciais de ações e começavam a ser negociadas em bolsas de valores. Muitas empresas eram avaliadas em bilhões de dólares sem ter nenhum fluxo significativo de receita. As ações da Internet sofreram uma queda também vertiginosa em 2000-2001, e muitas novas empresas fecharam. Não obstante, várias empresas surgiram como grandes vencedoras no mundo da Internet (mesmo que os preços de suas ações tivessem sofrido com aquela queda), entre elas Microsoft, Cisco, Yahoo, e-Bay, Google e Amazon.

1.7.5 Desenvolvimentos recentes

A inovação na área de redes de computadores continua a passos largos. Há progressos em todas as frentes, incluindo desenvolvimento de novas aplicações, distribuição de conteúdo, telefonia por Internet, velocidades de transmissão mais altas em LANs e roteadores mais rápidos. Mas três desenvolvimentos merecem atenção especial: a proliferação de redes de acesso de alta velocidade (incluindo acesso sem fio), a segurança e as redes P2P.

Como discutimos na Seção 1.2, a penetração cada vez maior do acesso residencial de banda larga à Internet via modem a cabo e DSL montou o cenário para uma profusão de novas aplicações multimídia, entre elas voz e vídeo sobre IP [Skype, 2009], compartilhamento de vídeo [Youtube, 2009] e televisão sobre IP [PPLive, 2009]. A crescente onipresença de redes Wi-Fi públicas de alta velocidade (11 Mbps e maiores) e de acesso de média velocidade (centenas de kbps) à Internet por redes de telefonia celular não está apenas possibilitando conexão constante, mas também habilitando um novo conjunto muito interessante de serviços específicos para localizações determinadas. Abordaremos redes sem fio e mobilidade no Capítulo 6.

Em seguida a uma série de ataques de recusa de serviço em importantes servidores Web no final da década de 1990 e à proliferação de ataques de worms (por exemplo, o Blaster) que infectam sistemas finais e emperram a rede com tráfego excessivo, a segurança da rede tornou-se uma questão extremamente importante. Esses ataques resultaram no desenvolvimento de sistemas de detecção de intrusão capazes de prevenir ataques com antecedência, na utilização de firewalls para filtrar tráfego indesejado antes que entre na rede. Abordaremos vários tópicos importantes relacionados à segurança no Capítulo 8.

A última inovação que queremos destacar são as redes P2P. Uma aplicação de rede P2P explora os recursos de computadores de usuários — armazenagem, conteúdo, ciclos de CPU e presença humana — e tem significativa autonomia em relação a servidores centrais. A conectividade dos computadores de usuários (isto é, dos pares, ou peers) normalmente é intermitente. Há alguns anos, houve diversas histórias de sucesso com o P2P, incluindo o compartilha-

mento de arquivo P2P (Napster, Kazaa, Gnutella, eDonkey, Lime Wire etc.), distribuição de arquivos (BitTorrent), Voz sobre IP (Skype) e IPTV (PPLive, ppStream). Muitas dessas aplicações P2P serão discutidas no Capítulo 2.

1.8 Resumo

Neste capítulo, abordamos uma quantidade imensa de assuntos. Examinamos as várias peças de hardware e software que compõem a Internet, em especial, e redes de computadores, em geral. Começamos pela periferia da rede, examinando sistemas finais e aplicações, além do serviço de transporte fornecido às aplicações que executam nos sistemas finais. Também vimos as tecnologias de camada de enlace e meio físico normalmente encontrados na rede de acesso. Em seguida, mergulhamos no interior da rede e chegamos ao seu núcleo, identificando comutação de pacotes e comutação de circuitos como as duas abordagens básicas do transporte de dados por uma rede de telecomunicações, expondo os pontos fortes e fracos de cada uma delas. Examinamos, então, as partes inferiores (do ponto de vista da arquitetura) da rede — as tecnologias de camada de enlace e os meios físicos comumente encontrados na rede de acesso. Examinamos também a estrutura da Internet global e aprendemos que ela é uma rede de redes. Vimos que a estrutura hierárquica da Internet, composta de ISPs de níveis mais altos e mais baixos, permitiu que ela se expandisse e incluisse milhares de redes.

Na segunda parte deste capítulo introdutório, abordamos diversos tópicos fundamentais da área de redes de computadores. Primeiramente examinamos as causas de atrasos e perdas de pacotes em uma rede de comutação de pacotes. Desenvolvemos modelos quantitativos simples de atrasos de transmissão, de propagação e de fila; esses modelos de atrasos serão muito usados nos problemas propostos em todo o livro. Em seguida examinamos camadas de protocolo e modelos de serviço, princípios fundamentais de arquitetura de redes aos quais voltaremos a nos referir neste livro. Analisamos também alguns dos ataques mais comuns na Internet atualmente. Terminamos nossa introdução sobre redes com uma breve história das redes de computadores. O primeiro capítulo constitui um minicurso sobre redes de computadores.

Portanto, percorremos realmente um extraordinário caminho neste primeiro capítulo! Se você estiver um pouco assustado, não se preocupe. Abordaremos todas essas ideias em detalhes nos capítulos seguintes (é uma promessa, e não uma ameaça!). Por enquanto, esperamos que, ao encerrar este capítulo, você tenha adquirido uma noção, ainda que incipiente, das peças que formam uma rede, um domínio ainda em desenvolvimento do vocabulário (não se acanhe de voltar aqui para consulta) e um desejo cada vez maior de aprender mais sobre redes. Esta é a tarefa que nos espera no restante deste livro.

O guia deste livro

Antes de iniciarmos qualquer viagem, sempre é bom consultar um guia para nos familiarizar com as estradas principais e desvios que encontraremos pela frente. O destino final da viagem que estamos prestes a empreender é um entendimento profundo do como, do quê e do porquê das redes de computadores. Nossa guia é a sequência de capítulos deste livro:

1. Redes de computadores e a Internet
2. Camada de aplicação
3. Camada de transporte
4. Camada de rede
5. Camada de enlace e redes locais (LANs)
6. Sem fio e redes móveis
7. Redes multimídia
8. Segurança em redes de computadores
9. Gerenciamento de rede

Os capítulos 2 a 5 são os quatro capítulos centrais deste livro. Note que esses capítulos estão organizados segundo as quatro camadas superiores da pilha de cinco camadas de protocolos da Internet, com um capítulo para

cada camada. Note também que nossa jornada começará no topo da pilha de protocolos da Internet, a saber, a camada de aplicação, e prosseguirá daí para baixo. O princípio racional que orienta essa jornada de cima para baixo é que, entendidas as aplicações, podemos compreender os serviços de rede necessários para dar suporte a elas. Então, poderemos examinar, um por um, os vários modos como esses serviços poderiam ser implementados por uma arquitetura de rede. Assim, o estudo das aplicações logo no início dá motivação para o restante do livro.

A segunda metade deste livro — capítulos 6 a 9 — aborda quatro tópicos extremamente importantes (e de certa maneira independentes) de redes modernas. No Capítulo 6, examinamos tecnologia sem fio e mobilidade, incluindo LANs sem fio (Wi-Fi, WiMAX e Bluetooth), redes de telefonia celular (GSM) e mobilidade (nas redes IP e GSM). No Capítulo 7 (Redes multimídia), examinamos aplicações de áudio e vídeo, como telefone por Internet, videoconferência e recepção de mídia armazenada. Examinamos também como uma rede de comutação de pacotes pode ser projetada para prover serviço de qualidade consistente para aplicações de áudio e vídeo. No Capítulo 8 (Segurança em redes de computadores), analisamos, primeiramente, os fundamentos da criptografia e da segurança de redes e, em seguida, de que modo a teoria básica está sendo aplicada a um amplo leque de contextos da Internet. No último capítulo (Gerenciamento de redes), examinamos as questões fundamentais do gerenciamento de redes, bem como os protocolos primários da Internet utilizados para esse fim.



Exercícios de fixação

Capítulo 1 Questões de revisão

Seção 1.1

- Qual é a diferença entre um hospedeiro e um sistema final? Cite os tipos de sistemas finais. Um servidor Web é um sistema final?
- A palavra protocolo é muito usada para descrever relações diplomáticas. Dê um exemplo de um protocolo diplomático.

Seção 1.2

- O que é um programa cliente? O que é um programa servidor? Um programa servidor requisita e recebe serviços de um programa cliente?
- Cite seis tecnologias de acesso. Classifique cada uma delas nas categorias acesso residencial, acesso corporativo ou acesso móvel.
- A taxa de transmissão HFC é dedicada ou é compartilhada entre usuários? É possível haver colisões na direção provedor-usuário de um canal HFC? Por quê?
- Cite seis tecnologias de acesso residencial disponíveis em sua cidade. Para cada tipo de acesso, apresente a taxa downstream, a taxa upstream e o preço mensal anunciados.
- Qual é a taxa de transmissão de LANs Ethernet? Para uma dada taxa de transmissão, cada usuário da LAN pode transmitir continuamente a essa taxa?

- Cite alguns meios físicos utilizados para instalar a Ethernet.
- Modems discados, HFC e ADSL são usados para acesso residencial. Para cada uma dessas tecnologias de acesso, cite uma faixa de taxas de transmissão e comente se a largura de banda é compartilhada ou dedicada.
- Descreva as tecnologias de acesso sem fio mais populares atualmente. Faça uma comparação entre elas.

Seção 1.3

- Qual é a vantagem de uma rede de comutação de circuitos em relação a uma de comutação de pacotes? Quais são as vantagens da TDM sobre a FDM em uma rede de comutação de circuitos?
- Por que se afirma que a comutação de pacotes emprega multiplexação estatística? Compare a multiplexação estatística com a multiplexação que ocorre em TDM.
- Suponha que exista exatamente um comutador de pacotes entre um computador de origem e um de destino. As taxas de transmissão entre a máquina de origem e o comutador e entre este e a máquina de destino são R_1 e R_2 , respectivamente. Admitindo que um roteador use comutação de pacotes do tipo armazena e reenvia, qual é o atraso total fim a fim para enviar um pacote de comprimento L ? (Desconsidere

- formação de fila, atraso de propagação e atraso de processamento).
- 14.** Qual é principal diferença que distingue ISPs de nível 1 e de nível 2?
- Suponha que usuários compartilhem um enlace de 2 Mbps e que cada usuário transmita continuamente a 1 Mbps, mas cada um deles transmite apenas 20 por cento do tempo. (Veja a discussão sobre multiplexação estatística na Seção 1.3).
- Quando a comutação de circuitos é utilizada, quantos usuários podem usar o enlace?
 - Para o restante deste problema, suponha que seja utilizada a comutação de pacotes. Por que não haverá atraso de fila antes de um enlace se dois ou menos usuários transmitirem ao mesmo tempo? Por que haverá atraso de fila se três usuários transmitirem ao mesmo tempo?
 - Determine a probabilidade de um dado usuário estar transmitindo.
 - Suponha agora que haja três usuários. Determine a probabilidade de, a qualquer momento, os três usuários transmitirem simultaneamente. Determine a fração de tempo durante o qual a fila cresce.
- 20.** Suponha que o sistema final A queira enviar um arquivo grande para o sistema B. Em um nível muito alto, descreva como o sistema A cria pacotes a partir do arquivo. Quando um desses arquivos chegar ao comutador de pacote, quais informações no pacote o comutador utiliza para determinar o enlace através do qual o pacote é encaminhado? Por que a comutação de pacote na Internet é análoga a dirigir de uma cidade para outra pedindo informações ao longo do caminho?
- 21.** Visite o applet "Queuing and Loss" no Companion Website. Qual é a taxa de emissão máxima e a taxa de emissão mínima? Com essas taxas, qual é a intensidade do tráfego? Execute o applet com essas taxas e determine o tempo que leva a ocorrência de uma perda de pacote. Repita o procedimento uma segunda vez e determine novamente o tempo de ocorrência para a perda de pacote. Os resultados são diferentes? Por quê? Por que não?

Seção 1.4

- 16.** Considere o envio de um pacote de uma máquina de origem a uma de destino por uma rota fixa. Relacione os componentes do atraso que formam o atraso fim a fim. Quais deles são constantes e quais são variáveis?
- 17.** Visite o applet "Transmission versus Propagation Delay" no Companion Website. Entre as taxas, o atraso de propagação e os tamanhos de pacote disponíveis, determine uma combinação para a qual o emissor termine de transmitir antes que o primeiro bit do pacote chegue ao receptor.
- 18.** Quanto tempo um pacote de 1.000 bytes leva para se propagar através de um enlace de 2.500 km de distância, com uma velocidade de propagação de $2.5 \cdot 10^8$ m/s e uma taxa de transmissão de 2 Mbps? Geralmente, quanto tempo um pacote de comprimento L leva para se propagar através de um enlace de distância d , velocidade de propagação s , e taxa de transmissão de R bps? Esse atraso depende do comprimento do pacote? Esse atraso depende da taxa de transmissão?
- 19.** Suponha que o Hospedeiro A queira enviar um arquivo grande para o Hospedeiro B. O percurso do Hospedeiro A para o Hospedeiro B possui três enlaces, de taxas $R_1 = 500$ kbps, $R_2 = 2$ Mbps, e $R_3 = 1$ Mbps.

a. Considerando que não haja nenhum outro tráfego na rede, qual é a vazão para a transferência de arquivo?

- b. Suponha que o arquivo tenha 4 milhões de bytes. Dividindo o tamanho do arquivo pela vazão, quanto tempo levará a transferência para o Hospedeiro B?
- c. Repita os itens "a" e "b", mas agora com R_2 reduzido a 100 kbps.

- 22.** Cite cinco tarefas que uma camada pode executar. É possível que uma (ou mais) dessas tarefas seja(m) realizada(s) por duas (ou mais) camadas?
- 23.** Quais são as cinco camadas da pilha de protocolo da Internet? Quais as principais responsabilidades de cada uma dessas camadas?
- 24.** O que é uma mensagem de camada de aplicação? Um segmento de camada de transporte? Um datagrama de camada de rede? Um quadro de camada de enlace?
- 25.** Que camadas da pilha do protocolo da Internet um roteador implementa? Que camadas um comutador de camada de enlace implementa? Que camadas um sistema final implementa?

Seção 1.5

- 26.** Qual é a diferença entre um vírus, um worm e um cavalo de Troia?
- 27.** Descreva como pode ser criado uma botnet e como ela pode ser utilizada no ataque DDoS.

Seção 1.6

28. Suponha que Alice e Bob estejam enviando pacotes um para o outro através de uma rede de computadores e que Trudy se posicione na rede para que ela consiga capturar todos os pacotes enviados por Alice

e enviar o que quiser para Bob; ela também consegue capturar todos os pacotes enviados por Bob e enviar o que quiser para Alice. Cite algumas atitudes maliciosas que Trudy pode fazer a partir de sua posição.

Problemas

1. Projete e descreva um protocolo de nível de aplicação para ser usado entre um caixa automático e o computador central de um banco. Esse protocolo deve permitir verificação do cartão e da senha de um usuário, consulta do saldo de sua conta (que é mantido no computador central) e saque de dinheiro da conta corrente (isto é, entrega de dinheiro ao usuário). As entidades do protocolo devem estar preparadas a resolver o caso comum em que não há dinheiro suficiente na conta do usuário para cobrir o saque. Faça uma especificação de seu protocolo relacionando as mensagens trocadas e as ações realizadas pelo caixa automático ou pelo computador central do banco na transmissão e recepção de mensagens. Esquematize a operação de seu protocolo para o caso de um saque simples sem erros, usando um diagrama semelhante ao da Figura 1.2. Descreva explicitamente o que seu protocolo espera do serviço de transporte fim a fim.
2. Considere uma aplicação que transmita dados a uma taxa constante (por exemplo, a origem gera uma unidade de dados de N bits a cada k unidades de tempo, onde k é pequeno e fixo). Considere também que, quando essa aplicação começa, continuará em funcionamento por um período de tempo relativamente longo. Responda às seguintes perguntas, dando uma breve justificativa para suas respostas:
 - a. O que seria mais apropriado para essa aplicação: uma rede de comutação de circuitos ou uma rede de comutação de pacotes? Por quê?
 - b. Suponha que seja usada uma rede de comutação de pacotes e que o único tráfego dessa rede venha de aplicações como a descrita anteriormente. Além disso, admita que a soma das velocidades de dados da aplicação seja menor do que a capacidade de cada um dos enlaces. Será necessário algum tipo de controle de congestionamento? Por quê?
3. Considere a rede de comutação de circuitos da Figura 1.12. Lembre-se de que há n circuitos em cada enlace.
 - a. Qual é o número máximo de conexões simultâneas que podem estar em curso a qualquer instante nessa rede?
4. Considere novamente a analogia do comboio de carros da Seção 1.4. Admita uma velocidade de propagação de 100 km/h.
 - a. Suponha que o comboio viaje 150 km, começando em frente ao primeiro dos postos de pedágio, passando por um segundo e terminando após um terceiro. Qual é o atraso fim a fim?
 - b. Repita o item 'a' admitindo agora que haja sete carros no comboio em vez de dez.
5. Este problema elementar começa a explorar atrasos de propagação e de transmissão, dois conceitos centrais em redes de computadores. Considere dois computadores, A e B, conectados por um único enlace de taxa R bps. Suponha que esses computadores estejam separados por m metros e que a velocidade de propagação ao longo do enlace seja de s metros/segundo. O computador A tem de enviar um pacote de L bits ao computador B.
 - a. Expressse o atraso de propagação, d_{prop} , em termos de m e s .
 - b. Determine o tempo de transmissão do pacote, d_{trans} , em termos de L e R .
 - c. Ignorando os atrasos de processamento e de fila, obtenha uma expressão para o atraso fim a fim.
 - d. Suponha que o computador A comece a transmitir o pacote no instante $t = 0$. No instante $t = d_{trans}$, onde estará o último bit do pacote?
 - e. Suponha que d_{prop} seja maior do que d_{trans} . Onde estará o primeiro bit do pacote no instante $t = d_{trans}$?
 - f. Suponha que d_{prop} seja menor do que d_{trans} . Onde estará o primeiro bit do pacote no instante $t = d_{trans}$?
 - g. Suponha $s = 2,5 \cdot 10^8$, $L = 100$ bits e $R = 56$ kbps. Encontre a distância m de forma que d_{prop} seja igual a d_{trans} .
6. Neste problema, consideramos o envio de voz em tempo real do computador A para o computador

- B por meio de uma rede de comutação de pacotes (VoIP). O computador A converte voz analógica para uma cadeia digital de bits de 64 kbps e, em seguida, agrupa os bits em pacotes de 56 bytes. Há apenas um enlace entre os computadores A e B; sua taxa de transmissão é de 2 Mbps e seu atraso de propagação, de 10 milissegundos. Assim que o computador A recolhe um pacote, ele o envia ao computador B. Quando recebe um pacote completo, o computador B converte os bits do pacote em um sinal analógico. Quanto tempo decorre entre o momento em que um bit é criado (a partir do sinal analógico no computador A) e o momento em que ele é decodificado (como parte do sinal analógico no computador B)?
7. Suponha que usuários compartilhem um enlace de 3 Mbps e que cada usuário precise de 150 kbps para transmitir, mas que transmita apenas durante 10 por cento do tempo. (Veja a discussão sobre multiplexação estatística na Seção 1.3.)
- Quando é utilizada comutação de circuitos, quantos usuários podem ter suporte?
 - Suponha que haja 120 usuários. Determine a probabilidade que, a um tempo dado, exatamente n usuários estejam transmitindo simultaneamente. (Dica: Use a distribuição binomial.)
 - Determine a probabilidade de haver 21 ou mais usuários transmitindo simultaneamente.
8. Considere a discussão na Seção 1.3 sobre multiplexação estatística, na qual é dado um exemplo com um enlace de 1 Mbps. Quando em atividade, os usuários estão gerando dados a uma taxa de 100 kbps; mas a probabilidade de estarem em atividade, gerando dados, é de $p = 0,1$. Suponha que o enlace de 1 Mbps seja substituído por um enlace de 1 Gbps.
- Qual é o número máximo de usuários, N , que pode ser suportado simultaneamente por comutação de pacotes?
 - Agora considere comutação de circuitos e um número M de usuários. Elabore uma fórmula (em termos de p , M , N) para a probabilidade de que mais de N usuários estejam enviando dados.
9. Considere um pacote de comprimento L que se inicia no sistema final A e percorre três enlaces até um sistema final de destino. Esses três enlaces estão conectados por dois comutadores de pacotes. Suponha que d_i , s_i e R_i representem o comprimento, a velocidade de propagação e a taxa de transmissão do enlace i , sendo $i = 1, 2, 3$. O comutador de pacote atrasa cada pacote por d_{proc} . Considerando que não haja nenhum atraso de fila, em relação a d_i , s_i e R_i , ($i = 1, 2, 3$) e L , qual é o atraso fim a fim total para o pacote? Suponha agora que o pacote tenha 1.500 bytes, a velocidade de propagação de ambos enlaces seja $2,5 \cdot 10^8$ m/s, as taxas de transmissão dos três enlaces sejam 2 Mbps, o atraso de processamento do comutador de pacote seja de 3 milissegundos, o comprimento do primeiro enlace seja 5.000 km, o comprimento do segundo seja 4.000 km e o último seja 1.000 km. Dados esses valores, qual é o atraso fim a fim?
10. No problema anterior, suponha que $R_1 = R_2 = R_3 = R$ e $d_{proc} = 0$. Suponha que o comutador de pacote não armazena e envia pacotes, mas transmite imediatamente cada bit recebido antes de esperar o pacote chegar. Qual é o atraso fim a fim?
11. Um comutador de pacote recebe um pacote e determina o enlace de saída pelo qual o pacote deve ser enviado. Quando o pacote chega, um outro já está sendo transmitido nesse enlace de saída e outros quatro já estão esperando para serem transmitidos. Os pacotes são transmitidos em ordem de chegada. Suponha que todos os pacotes tenham 1.500 bytes e que a taxa do enlace seja 2 Mbps. Qual é o atraso de fila para o pacote? Em um amplo sentido, qual é o atraso de fila quando todos os pacotes possuem comprimento L , o enlace possui uma taxa de transmissão $R \times$ bits do pacote sendo enviado já foram transmitidos e N pacotes já estão na fila?
12. Suponha que N pacotes cheguem simultaneamente ao enlace no qual não há pacotes sendo transmitidos e nem pacotes enfileirados. Cada pacote tem L de comprimento e é transmitido à taxa R . Qual é o atraso médio para os N pacotes?
13. Considere o atraso de fila em um buffer de roteador (antes de um enlace de saída). Suponha que todos os pacotes tenham L bits, que a taxa de transmissão seja de R bps e que N pacotes cheguem simultaneamente ao buffer a cada LN/R segundos. Determine o atraso de fila médio para um pacote. (Dica: o atraso de fila para o primeiro pacote é zero; para o segundo pacote, L/R ; para o terceiro pacote, $2L/R$. O pacote de ordem N já terá sido transmitido quando o segundo lote de pacotes chegar.)
14. Considere o atraso de fila em um buffer de roteador, sendo I a intensidade de tráfego; isto é, $I = La/R$. Suponha que o atraso de fila tome a forma de IL/R ($1 - I$) para $I < 1$.
- Deduza uma fórmula para o atraso total, isto é, para o atraso de fila mais o atraso de transmissão.
 - Faça um gráfico do atraso total como uma função de L/R .
15. Sendo a a taxa de pacotes que chegam a um enlace em pacotes/s, e μ a taxa de transmissão de enlaces em pacotes/s, baseado na fórmula do atraso total (isto é, o atraso de fila mais o atraso de transmissão) do problema anterior, deduza uma fórmula para o atraso total em relação a a e μ .

16. Considere um buffer de roteador antes de um enlace de saída. Neste problema, você usará a fórmula de Little, uma famosa fórmula da teoria das filas. Sendo N o número médio de pacotes no buffer mais o pacote sendo transmitido, a a taxa de pacotes que chegam no enlace, e d o atraso total médio (isto é, o atraso de fila mais o atraso de transmissão) sofrido pelo pacote. Dada a fórmula de Little $N = a \cdot d$, suponha que, na média, o buffer contenha 10 pacotes, o atraso de fila de pacote médio seja 10 milissegundos e a taxa de transmissão do enlace seja 100 pacotes/s. Utilizando tal fórmula, qual é a taxa média de chegada de pacote, considerando que não há perda de pacote?
17. a. Generalize a fórmula para o atraso fim a fim na Seção 1.4.3 para taxas de processamento, atrasos de propagação e taxa de transmissão heterogêneos.
 b. Repita o item "a", mas suponha também que haja um atraso de fila médio d_{fila} em cada nó.
18. Execute o programa Traceroute para verificar a rota entre uma origem e um destino, no mesmo continente, para três horários diferentes do dia.
 a. Determine a média e o desvio padrão dos atrasos de ida e volta para cada um dos três horários.
 b. Determine o número de roteadores no caminho para cada um dos três. Os caminhos mudaram em algum dos horários?
 c. Tente identificar o número de redes ISPs pelas quais o pacote do Traceroute passa entre origem e destino. Roteadores com nomes semelhantes e/ou endereços IP semelhantes devem ser considerados como parte do mesmo ISP. Em suas respostas, os maiores atrasos ocorrem nas interfaces de formação de pares entre ISPs adjacentes?
 d. Faça o mesmo para uma origem e um destino em continentes diferentes. Compare os resultados dentro do mesmo continente com os resultados entre continentes diferentes.
19. Considere o exemplo de vazão correspondente à Figura 1.20 (b). Agora imagine que haja M pares de cliente-servidor em vez de 10. R_s , R_c e R representam as taxas do enlace do servidor, enlaces do cliente e enlace da rede. Suponha que os outros enlaces possuam capacidade abundante e que não haja outro tráfego na rede além daquele gerado pelos M pares de cliente-servidor. Deduza uma expressão geral para a vazão em relação a R_s , R_c , R e M .
20. Considere a Figura 1.19 (b). Agora suponha que haja M percursos entre o servidor e o cliente. Nenhum dos dois percursos compartilham qualquer enlace. O percurso k ($k = 1, \dots, M$) consiste em N enlaces com taxas de transmissão $R_1^k, R_2^k, \dots, R_N^k$. Se o servidor pode usar somente um percurso para enviar dados ao cliente, qual é a vazão máxima que ele pode atingir? Se o servidor pode usar todos os M percursos para enviar dados, qual é a vazão máxima que ele pode atingir?
21. Considere a Figura 1.19 (b). Suponha que cada enlace entre o servidor e o cliente possua uma probabilidade de perda de pacote p , e que as probabilidades de perda de pacote para esses enlaces sejam independentes. Qual é a probabilidade de um pacote (enviado pelo servidor) ser recebido com sucesso pelo receptor? Se o pacote se perder no percurso do servidor para o cliente, então o servidor retransmitirá o pacote. Na média, quantas vezes o servidor retransmitirá o pacote para que o cliente o receba com sucesso?
22. Considere a Figura 1.19 (a). Suponha que o enlace de gargalo ao longo do percurso do servidor para o cliente seja o primeiro com a taxa R_s bps. Imagine que enviamos um par de pacotes um após o outro do servidor para o cliente, e que não haja nenhum outro tráfego nesse percurso. Imagine também que cada pacote de tamanho L bits e os dois enlaces tenham o mesmo atraso de propagação d_{prop} .
 a. Qual é o tempo entre chegadas ao destino? Isto é, quanto tempo transcorre de quando o último bit do primeiro pacote chega até o último bit do segundo pacote chegar?
 b. Agora suponha que o segundo enlace seja o de gargalo (isto é, $R_c < R_s$). É possível que o segundo pacote entre na fila de entrada do segundo enlace? Explique. Agora imagine que o servidor envie o segundo pacote T segundos após enviar o primeiro. Quão grande deve ser T para garantir que não haja nenhuma fila antes do segundo enlace? Explique.
23. Suponha que você queira enviar, urgentemente, 40 terabytes de dados de Boston para Los Angeles. Você tem disponível um enlace dedicado de 100 Mbps para transferência de dados. Você escolheria transmitir os dados por meio desse enlace ou usar o serviço de entrega 24 horas FedEx? Explique.
24. Suponha que dois computadores, A e B, estejam separados por uma distância de 20 mil quilômetros e conectados por um enlace direto de $R = 2$ Mbps. Suponha que a velocidade de propagação pelo enlace seja de $2,5 \cdot 10^8$ metros por segundo.
 a. Calcule o produto largura de banda-atraso $R \cdot d_{prop}$.
 b. Considere o envio de um arquivo de 800 mil bits do computador A para o computador B. Suponha que o arquivo seja enviado continuamente, como se fosse uma única grande mensagem. Qual é o número máximo de bits que estará no enlace a qualquer dado instante?

- c. Interprete o produto largura de banda-atraso.
- d. Qual é o comprimento (em metros) de um bit no enlace? É maior do que a de um campo de futebol?
- e. Derive uma expressão geral para o comprimento de um bit em termos da velocidade de propagação s , da velocidade de transmissão R e do comprimento do enlace m .
25. Com referência ao problema 24, suponha que possamos modificar R . Para qual valor de R o comprimento de um bit será o mesmo que o comprimento do enlace?
26. Considere o problema 24, mas agora com um enlace de $R = 1$ Gbps.
- Calcule o produto largura de banda-atraso, $R \cdot d_{\text{prop}}$.
 - Considere o envio de um arquivo de 800 mil bits do computador A para o computador B. Suponha que o arquivo seja enviado continuamente, como se fosse uma única grande mensagem. Qual será o número máximo de bits que estará no enlace a qualquer dado instante?
 - Qual é o comprimento (em metros) de um bit no enlace?
27. Novamente com referência ao problema 24.
- Quanto tempo demora para enviar o arquivo, admitindo que ele seja enviado continuamente?
 - Suponha agora que o arquivo seja fragmentado em 20 pacotes e que cada pacote contenha 40 mil bits. Suponha que cada pacote seja verificado pelo receptor e que o tempo de transmissão de uma verificação de pacote seja desprezível. Finalmente, admita que o emissor não possa enviar um pacote até que o anterior tenha sido reconhecido. Quanto tempo demorará para enviar o arquivo?
 - Compare os resultados de 'a' e 'b'.
- Suponha que haja um enlace de microondas de 10 Mbps entre um satélite geoestacionário e sua estação-base na Terra. A cada minuto o satélite tira uma foto digital e a envia à estação-base. Admita uma velocidade de propagação de $2,4 \cdot 10^8$ metros por segundo.
- Qual é o atraso de propagação do enlace?
 - Qual é o produto largura de banda-atraso, $R \cdot d_{\text{prop}}$?
 - Seja x o tamanho da foto. Qual é o valor mínimo de x para que o enlace de micro-ondas transmita continuamente?
28. Considere a analogia da viagem aérea que utilizamos em nossa discussão sobre camadas na Seção 1.7, e a adição de cabeçalhos a unidades de dados de protocolo enquanto passam por sua pilha. Existe uma noção equivalente de adição de informações de cabeçalho à movimentação de passageiros e suas malas pela pilha de protocolos da linha aérea?
29. Em redes modernas de comutação de pacotes, a máquina de origem segmenta mensagens longas de camada de aplicação (por exemplo, uma imagem ou um arquivo de música) em pacotes menores e os envia pela rede. A máquina destinatária, então, monta novamente os pacotes restaurando a mensagem original. Denominamos esse processo segmentação de mensagem. A Figura 1.28 ilustra o transporte fim a fim de uma mensagem com e sem segmentação. Considere que uma mensagem de $8 \cdot 10^6$ bits de comprimento tenha de ser enviada da origem ao destino na Figura 1.28. Suponha que a velocidade de cada enlace da figura seja 2 Mbps. Ignore atrasos de propagação, de fila e de processamento.
- Considere o envio da mensagem da origem ao destino sem segmentação. Quanto tempo essa mensagem levará para ir da máquina de origem até o primeiro comutador de pacotes? Tendo em mente que cada comutador usa comutação de pacotes do tipo armazena-e-reenvia, qual é o tempo total para levar a mensagem da máquina de origem à máquina de destino?
 - Agora suponha que a mensagem seja segmentada em 4 mil pacotes, cada um com 2.000 bits de comprimento. Quanto tempo demorará para o primeiro pacote ir da máquina de origem até o primeiro comutador? Quando o primeiro pacote está sendo enviado do primeiro ao segundo comutador, o segundo pacote está sendo enviado da máquina de origem ao primeiro comutador. Em que instante o segundo pacote terá sido completamente recebido no primeiro comutador?
 - Quanto tempo demorará para movimentar o arquivo da máquina de origem até a máquina de destino quando é usada segmentação de mensagem? Compare este resultado com sua resposta na parte 'a' e comente.
 - Discuta as desvantagens da segmentação de mensagem.
30. Experimente o applet "Message Segmentation apresentado" no site Web deste livro. Os atrasos no applet correspondem aos atrasos obtidos no problema anterior? Como os atrasos de propagação no enlace afetam o atraso total fim a fim na comutação de pacotes (com segmentação de mensagem) e na comutação de mensagens?

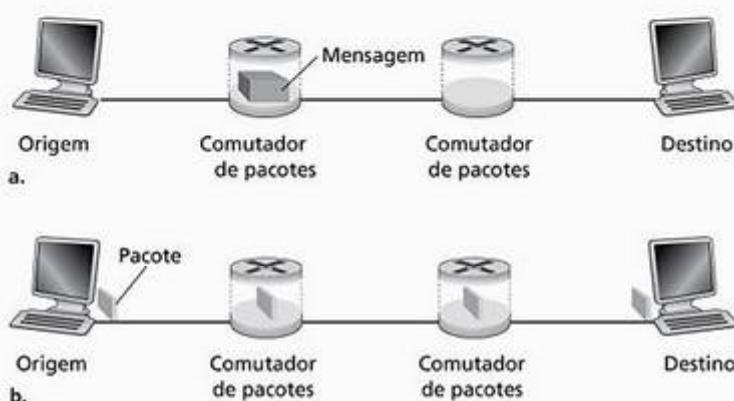


Figura 1.28 Transporte fim a fim de mensagem: a) sem segmentação de mensagem; (b) com segmentação de mensagem

31. Considere o envio de um arquivo grande de F bits do computador A para o computador B. Há dois enlaces (e um comutador) entre eles e os enlaces não estão congestionados (isto é, não há atrasos de fila). O computador A fragmenta o arquivo em segmentos de S bits

cada e adiciona 80 bits de cabeçalho a cada segmento, formando pacotes de $L = 40 + S$ bits. Cada enlace tem uma taxa de transmissão de R bps. Qual é o valor de S que minimiza o atraso para levar o arquivo de A para B? Desconsidere o atraso de propagação.



Questões dissertativas

- Que tipos de serviços de telefone celular sem fio estão disponíveis em sua área?
- Usando tecnologia de LAN sem fio 802.11, elabore o projeto de uma rede doméstica para sua casa ou para a casa de seus pais. Relacione os modelos de produtos específicos para essa rede doméstica juntamente com seus custos.
- Descreva os serviços Skype PC para PC. Experimente o serviço de vídeo do Skype PC para PC e relate a experiência.
- O Skype oferece um serviço que permite realizar chamadas telefônicas de um computador para um computador comum. Isso significa que a chamada deve passar pela Internet e pela rede telefônica. Discuta como esse processo deve ser feito.
- O que é Short Message Service? Em quais países/continentes esse serviço é comum? É possível enviar uma mensagem SMS de um site Web para um telefone portátil?
- O que é recepção de vídeo armazenado? Quais são os sites Web mais populares que oferecem esse serviço hoje?
- O que é recepção de vídeo em tempo real por P2P? Quais são os sites Web mais populares que oferecem esse serviço hoje?
- Descubra cinco empresas que oferecem serviços de compartilhamento de arquivos P2P. Cite os tipos de arquivos (isto é, conteúdo) que cada empresa processa.
- Quem inventou o ICQ, o primeiro serviço de mensagem instantânea? Quando foi inventado e que idade tinham seus inventores? Quem inventou o Napster? Quando foi inventado e que idade tinham seus inventores?
- Quais são as semelhanças e diferenças entre as tecnologias Wi-Fi e 3G de acesso sem fio à Internet? Quais são as taxas de bits dos dois serviços? Quais são os custos? Discuta roaming e acesso de qualquer lugar.
- Por que o serviço original de compartilhamento de arquivo P2P Napster deixou de existir? O que é a RIAA e que providências está tomando para limitar o compartilhamento de arquivos P2P de conteúdo protegido por direitos autorais? Qual é a diferença entre violação de direitos autorais direta e indireta?
- O que é BitTorrent? No que ele difere de um serviço de compartilhamento de arquivo P2P, como eDonkey, LimeWire ou Kazaa?
- Você acha que daqui a dez anos redes de computadores ainda compartilharão amplamente arquivos protegidos por direitos autorais? Por que sim ou por que não? Justifique.



Wireshark Lab

“Conte-me e eu esquecerei. Mostre-me e eu lembrarei. Envolva-me e eu entenderei.”

Provérbio chinês

A compreensão de protocolos de rede pode ser muito mais profunda se os virmos em ação e interagirmos com eles — observando a sequência de mensagens trocadas entre duas entidades de protocolo, pesquisando detalhes de sua operação, fazendo com que eles executem determinadas ações e observando essas ações e suas consequências. Isso pode ser feito em cenários simulados ou em um ambiente real de rede, tal como a Internet. Os applets Java apresentados (em inglês) no site deste livro adotam a primeira abordagem. Nos Wireshark labs adotaremos a última. Você executará aplicações de rede em vários cenários utilizando seu computador no escritório, em casa ou em um laboratório e observará também os protocolos de rede interagindo e trocando mensagens com entidades de protocolo que estão executando em outros lugares da Internet. Assim, você e seu computador serão partes integrantes desses laboratórios ao vivo e você observará — e aprenderá — fazendo.

A ferramenta básica para observar as mensagens trocadas entre entidades de protocolos em execução é denominada analisador de pacotes. Como o nome sugere, um analisador de pacotes recebe passivamente mensagens enviadas e recebidas por seu computador; também exibe o conteúdo dos vários campos de protocolo das mensagens que captura. Uma tela do analisador de pacotes Wireshark é mostrada na Figura 1.29. O Wireshark é um analisador de pacotes gratuito que funciona em computadores com sistemas operacionais Windows, Linux/Unix e Mac. Por todo o livro, você encontrará Wireshark labs que o habilitarão a explorar vários dos protocolos estudados em cada capítulo. Neste primeiro Wireshark lab, você obterá e instalará uma cópia do programa, acessará um site Web e examinará as mensagens de protocolo trocadas entre seu browser e o servidor Web.

Você encontrará detalhes completos, em inglês, sobre este primeiro Wireshark Lab (incluindo instruções sobre como obter e instalar o programa) no site www.aw.com/kurose_br.

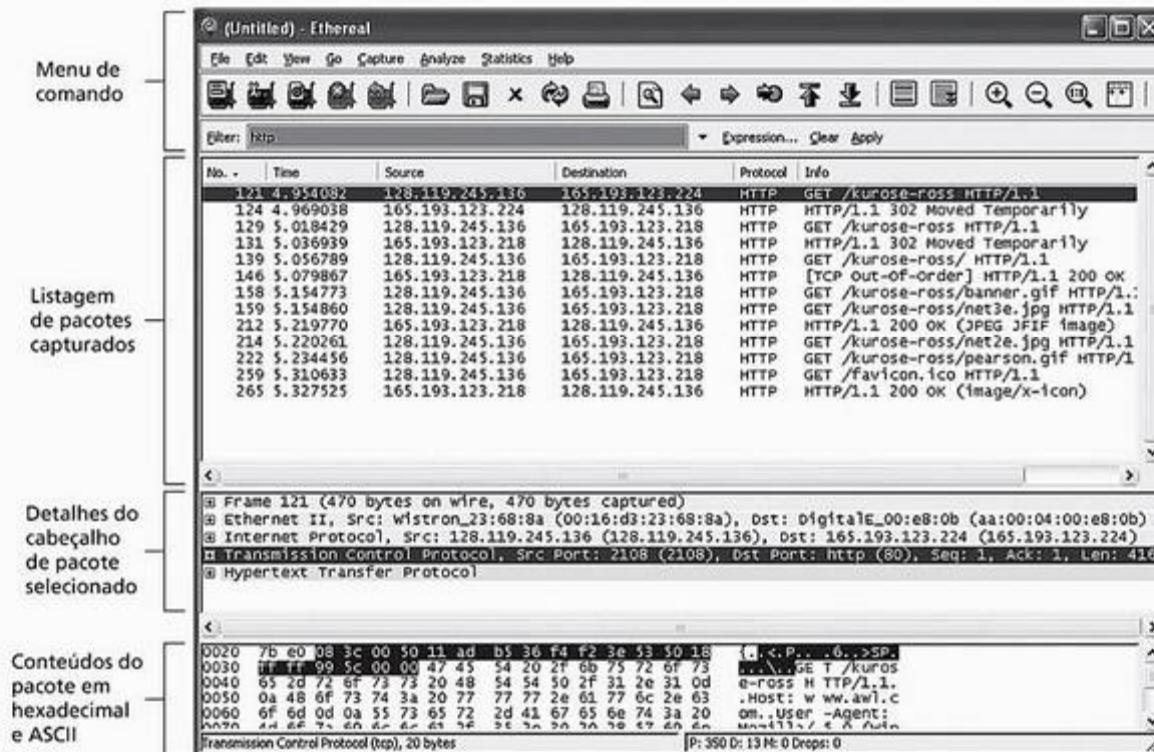


Figura 1.29 Uma amostra de tela do programa Wireshark

Entrevista

Leonard Kleinrock

Leonard Kleinrock é professor de ciência da computação da Universidade da Califórnia em Los Angeles. Em 1969, seu computador na UCLA se tornou o primeiro nó da Internet. Ele criou os princípios da comutação de pacotes em 1961, que se tornou a tecnologia básica da Internet. Leonard também é presidente e fundador da Nomadix, Inc., uma companhia cuja tecnologia oferece maior acessibilidade a serviços de Internet banda larga. Ele é bacharel em engenharia elétrica pela City College of New York (CCNY) e mestre e doutor em engenharia elétrica pelo Instituto de Tecnologia de Massachusetts (MIT).



O que o fez se decidir pela especialização em tecnologia de redes/Internet?

Como doutorando do MIT em 1959, percebi que a maioria dos meus colegas de turma estava fazendo pesquisas nas áreas de teoria da informação e de teoria da codificação. Havia no MIT o grande pesquisador Claude Shannon, que já tinha proposto estudos nessas áreas e resolvido a maior parte dos problemas importantes. Os problemas que restaram para pesquisar eram difíceis e de menor importância. Portanto, decidi propor uma nova área na qual até então ninguém tinha pensado. Lembre-se de que no MIT, eu estava cercado de computadores, e era evidente para mim que brevemente aquelas máquinas teriam de se comunicar umas com as outras. Na época, não havia nenhum meio eficaz de fazer isso; portanto, decidi desenvolver a tecnologia que permitiria a criação de redes de dados eficientes.

Qual foi seu primeiro emprego no setor de computação? O que implicava?

Frequentei o curso noturno de graduação em engenharia elétrica da CCNY de 1951 a 1957. Durante o dia, trabalhei inicialmente como técnico e depois como engenheiro em uma pequena empresa de eletrônica industrial chamada Photobell. Enquanto trabalhava lá, introduzi tecnologia digital na linha de produtos da empresa. Essencialmente, estávamos usando equipamentos fotoelétricos para detectar a presença de certos itens (caixas, pessoas etc.) e a utilização de um circuito conhecido na época como *multivibrador biestável* era exatamente o tipo de tecnologia de que precisávamos para levar o processamento digital a esse campo da detecção. Acontece que esses circuitos são os blocos de construção básicos dos computadores e vieram a ser conhecidos como *flip-flops* ou *comutadores* na linguagem coloquial de hoje.

O que passou por sua cabeça quando enviou a primeira mensagem computador a computador (da UCLA para o Stanford Research Institute)?

Francamente, eu não fazia a menor ideia da importância daquele acontecimento. Não havíamos preparado uma mensagem de significância histórica, como muitos criadores do passado o fizeram (Samuel Morse com "Que obra fez Deus.", ou Alexandre Graham Bell, com "Watson, venha cá! Preciso de você.", ou Neal Armstrong com "Este é um pequeno passo para o homem, mas um grande salto para a humanidade".) Esses caras eram inteligentes! Eles entendiam de meios de comunicação e relações públicas. Nossa objetivo foi nos conectar ao computador do SRI. Então digitamos a letra "L", que foi aceita corretamente, digitamos a letra "o", que foi aceita, e depois digitamos a letra "g", que fez o computador hospedeiro pifar! Então, nossa mensagem acabou sendo curta e, talvez, a mais profética de todas, ou seja, "Lo!", como em "*Lo and behold*" ("*Pasmem!*").

Anteriormente, naquele mesmo ano, fui citado em um comunicado de imprensa da UCLA por ter dito que, logo que a rede estivesse pronta e em funcionamento, seria possível ter acesso a utilidades de outros computadores a partir de nossa casa e escritório tão facilmente quanto tínhamos acesso à eletricidade e ao telefone. Portanto,

a visão que eu tinha da Internet naquela época era que ela seria onipresente, estaria sempre em funcionamento e sempre disponível, que qualquer pessoa que possuisse qualquer equipamento poderia se conectar com ela de qualquer lugar e que ela seria invisível. Mas jamais imaginei que minha mãe, aos 99 anos de idade, usaria a Internet — como ela realmente usou!

Em sua opinião, qual é o futuro das redes?

O fácil é predizer a infraestrutura por si mesma. Eu antecipo que vemos uma implantação considerável de computação nômade, aparelhos móveis e espaços inteligentes. Realmente, a disponibilidade de computação portátil, de alto desempenho, acessível e leve e de aparelho de comunicação (mais a onipresença da Internet) nos permitiu tornar nômades.

A computação nômade refere-se à tecnologia que permite aos usuários finais, que viajam de um lugar para o outro, ganhar acesso aos serviços da Internet de maneira transparente, não importando para onde vão e qual aparelho possuem ou ganham acesso. O difícil é predizer as aplicações e serviços, que nos surpreenderam consistentemente de formas dramáticas (e-mail, tecnologias de busca, a rede de alcance mundial, blogs, redes sociais, geração de usuários e compartilhamento de música, fotos, vídeos etc.). Estamos na margem de uma nova categoria de aplicações móveis, inovadoras e surpreendentes presentes em nossos aparelhos portáteis.

O passo seguinte vai nos capacitar a sair do mundo misterioso do ciberespaço para o mundo físico dos espaços inteligentes. A tecnologia dará vida a nossos ambientes (mesas, paredes, veículos, relógios e cintos, entre outros) por meio de atuadores, sensores, lógica, processamento, armazenagem, câmeras, microfones, alto-falantes, painéis e comunicação. Essa tecnologia embutida permitirá que nosso ambiente forneça os serviços IP que quisermos. Quando eu entrar em uma sala, ela saberá que entrei. Poderei me comunicar com meu ambiente naturalmente, como se estivesse falando o meu idioma nativo; minhas solicitações gerarão respostas apresentadas como páginas Web em painéis de parede, por meus óculos, por voz, por hologramas e assim por diante.

Analizando um panorama mais longínquo, vejo um futuro para as redes que inclui componentes fundamentais que ainda virão. Vejo agentes inteligentes de software distribuídos por toda a rede cuja função é fazer mineração de dados, agir sobre esses dados, observar tendências e adaptar e realizar tarefas dinamicamente. Vejo tráfego de rede consideravelmente maior gerado não tanto por seres humanos, mas por esses equipamentos embutidos e agentes inteligentes de software. Vejo grandes conjuntos de sistemas auto-organizáveis controlando essa rede imensa e veloz. Vejo quantidades enormes de informações zunindo por essa rede instantaneamente e passando por extraordinários processamentos e filtragens. A Internet será, essencialmente, um sistema nervoso de presença global. Vejo tudo isso e mais enquanto entramos de cabeça no século XXI.

Que pessoas o inspiraram profissionalmente?

Quem mais me inspirou foi Claude Shannon, do MIT, um brilhante pesquisador que tinha a capacidade de relacionar suas ideias matemáticas com o mundo físico de modo muitíssimo intuitivo. Ele fazia parte da banca examinadora de minha tese de doutorado.

Você pode dar algum conselho aos estudantes que estão ingressando na área de redes/Internet?

A Internet, e tudo o que ela habilita, é uma vasta fronteira nova, cheia de desafios surpreendentes. Há espaço para grandes inovações. Não fiquem limitados à tecnologia existente hoje. Soltem sua imaginação e pensem no que poderia acontecer e transformem isso em realidade.



Capítulo 2

Camada de aplicação



Aplicações são a razão de ser de uma rede de computadores. Se não fosse possível inventar aplicações úteis, não haveria necessidade de projetar protocolos de rede para suportá-las. Nos últimos 40 anos, foram criadas numerosas aplicações de rede engenhosas e maravilhosas. Entre elas estão as aplicações clássicas de texto, que se tornaram populares na década de 1970 e 1980: correio eletrônico, acesso a computadores remotos, transferência de arquivo, grupos de discussão e bate-papo e também uma aplicação que alcançou estrondoso sucesso em meados da década de 1990: a World Wide Web, abrangendo a navegação na Web, busca e o comércio eletrônico. Duas aplicações de enorme sucesso também surgiram no final do milênio — mensagem instantânea com lista de amigos e compartilhamento P2P de arquivos, assim como muitas aplicações de áudio e vídeo, incluindo a telefonia por Internet, transmissão e compartilhamento de vídeo, rádio via Internet e televisão sobre o protocolo IP (IPTV). Além disso, a penetração crescente de acesso residencial banda larga e a onipresença de acesso sem fio estão preparando o terreno para aplicações mais modernas e interessantes no futuro.

Neste capítulo estudamos os aspectos conceituais e de implementação de aplicações de rede. Começamos definindo conceitos fundamentais de camada de aplicação, incluindo serviços de rede exigidos por aplicações, clientes e servidores, processos e interfaces de camada de transporte. Examinamos detalhadamente várias aplicações de rede, entre elas a Web, e-mail, DNS, compartilhamento de arquivos P2P e telefonia por Internet P2P.

Em seguida, abordamos desenvolvimento de aplicação de rede por TCP e também por UDP. Em particular, estudamos o API socket e examinamos algumas aplicações cliente-servidor simples em Java. Apresentamos também vários exercícios divertidos e interessantes de programação de aplicações no final do capítulo.

A camada de aplicação é um lugar particularmente bom para iniciarmos o estudo de protocolos. É terreno familiar, pois conhecemos muitas das aplicações que dependem dos protocolos que estudaremos. Ela nos dará uma boa ideia do que são protocolos e nos apresentará muitos assuntos que encontraremos novamente quando estudarmos protocolos de camadas de transporte, de rede e de enlace.

2.1 Princípios de aplicações de rede

Suponha que você tenha uma grande ideia para uma nova aplicação de rede. Essa aplicação será, talvez, um grande serviço para a humanidade, ou agradará a seu professor, ou fará de você um homem (ou mulher) rico(a), ou simplesmente será divertido desenvolvê-la. Seja qual for sua motivação, vamos examinar agora como transformar a ideia em uma aplicação do mundo real.

O cerne do desenvolvimento de aplicação de rede é escrever programas que rodem em sistemas finais diferentes e se comuniquem entre si pela rede. Por exemplo, na aplicação Web há dois programas distintos que se comunicam um com o outro: o programa do browser, que roda na máquina do usuário (computador de mesa, laptop, PDA, telefone celular e assim por diante); e o programa do servidor Web, que roda na máquina do servidor Web. Outro exemplo é um sistema de compartilhamento de arquivos P2P no qual há um programa em cada máquina que participa da comunidade de compartilhamento de arquivos. Nesse caso, os programas de cada máquina podem ser semelhantes ou idênticos.

Portanto, ao desenvolver sua nova aplicação, você precisará escrever um software que rode em vários sistemas finais. Esse software poderia ser criado, por exemplo, em C, Java ou Python. Importante: você não precisará escrever programas que executem nos elementos do núcleo de rede, como roteadores e switches. Mesmo se quisesse, você não poderia escrever programas para esses elementos. Como aprendemos no Capítulo 1 e mostramos na Figura 1.24, equipamentos de núcleo de rede não funcionam na camada de aplicação, mas em camadas mais baixas, especificamente na de rede e abaixo dela. Esse projeto básico — a saber, confinar o software de aplicação nos sistemas finais, como mostra a Figura 2.1, facilitou o desenvolvimento e a proliferação rápidos de uma vasta gama de aplicações de Internet.

2.1.1 Arquiteturas de aplicação de rede

Antes de mergulhar na codificação do software, você deverá elaborar um plano geral para a arquitetura da sua aplicação. Tenha sempre em mente que a arquitetura de uma aplicação é distintamente diferente da arquitetura de rede (por exemplo, a arquitetura em cinco camadas da Internet que discutimos no Capítulo 1). Do ponto de vista do profissional que desenvolve a aplicação, a arquitetura de rede é fixa e provê um conjunto específico de serviços às aplicações. Por outro lado, a arquitetura da aplicação é projetada pelo desenvolvedor e determina como a aplicação é organizada nos vários sistemas finais. Ao escolher a arquitetura da aplicação, o desenvolvedor provavelmente aproveitará uma das duas arquiteturas mais utilizadas em aplicações modernas de rede: a arquitetura cliente-servidor ou a arquitetura P2P.

Em uma arquitetura cliente-servidor há um hospedeiro sempre em funcionamento, denominado *servidor*, que atende a requisições de muitos outros hospedeiros, denominados *clientes*. Estes podem estar em funcionamento às vezes ou sempre. Um exemplo clássico é a aplicação Web na qual um servidor Web que está sempre em funcionamento atende a requisições de browsers de hospedeiros clientes. Quando recebe uma requisição de um objeto de um hospedeiro cliente, um servidor Web responde enviando o objeto requisitado a ele. Observe que, na arquitetura cliente-servidor, os clientes não se comunicam diretamente uns com os outros; por exemplo, na aplicação Web, dois browsers não se comunicam diretamente. Outra característica da arquitetura cliente-servidor é que o servidor tem um endereço fixo, bem conhecido, denominado endereço IP (que discutiremos em breve). Devido a essa característica do servidor e devido ao fato de ele estar sempre em funcionamento, um cliente sempre pode contatá-lo, enviando um pacote ao endereço do servidor. Algumas das aplicações mais conhecidas que empregam a arquitetura cliente-servidor são Web, FTP, Telnet e e-mail. Essa arquitetura cliente-servidor é mostrada na Figura 2.2(a).

Em aplicações cliente-servidor, muitas vezes acontece de um único hospedeiro servidor ser incapaz de atender a todas as requisições de seus clientes. Por exemplo, um site Web popular pode ficar rapidamente saturado se tiver apenas um servidor para atender a todas as requisições. Por essa razão, um grande conjunto de hospedeiros — às vezes coletivamente chamado ***data center*** — frequentemente é usado para criar um servidor virtual poderoso em arquiteturas cliente-servidor. Os serviços de aplicação baseados na arquitetura cliente-servidor são geralmente de infraestrutura intensiva, uma vez que requerem que os provedores de serviço comprem, instalem e preservem o *server farm*. Além disso, os provedores de serviço devem pagar as despesas de interconexão recorrente e largura de banda para enviar e receber dados para e da Internet. Serviços populares como mecanismos de busca (por exemplo, o Google), comércio via Internet (por exemplo, Amazon e e-Bay), e-mail baseado na Web (por exemplo, Yahoo Mail), rede social (por exemplo, MySpace e Facebook) e compartilhamento de vídeo (por exemplo, YouTube) exigem muita infraestrutura e são de fornecimento dispendioso.

Em uma **arquitetura P2P**, há uma confiança mínima (ou nenhuma) nos servidores sempre em funcionamento. Em vez disso, a aplicação utiliza a comunicação direta entre pares de hospedeiros conectados

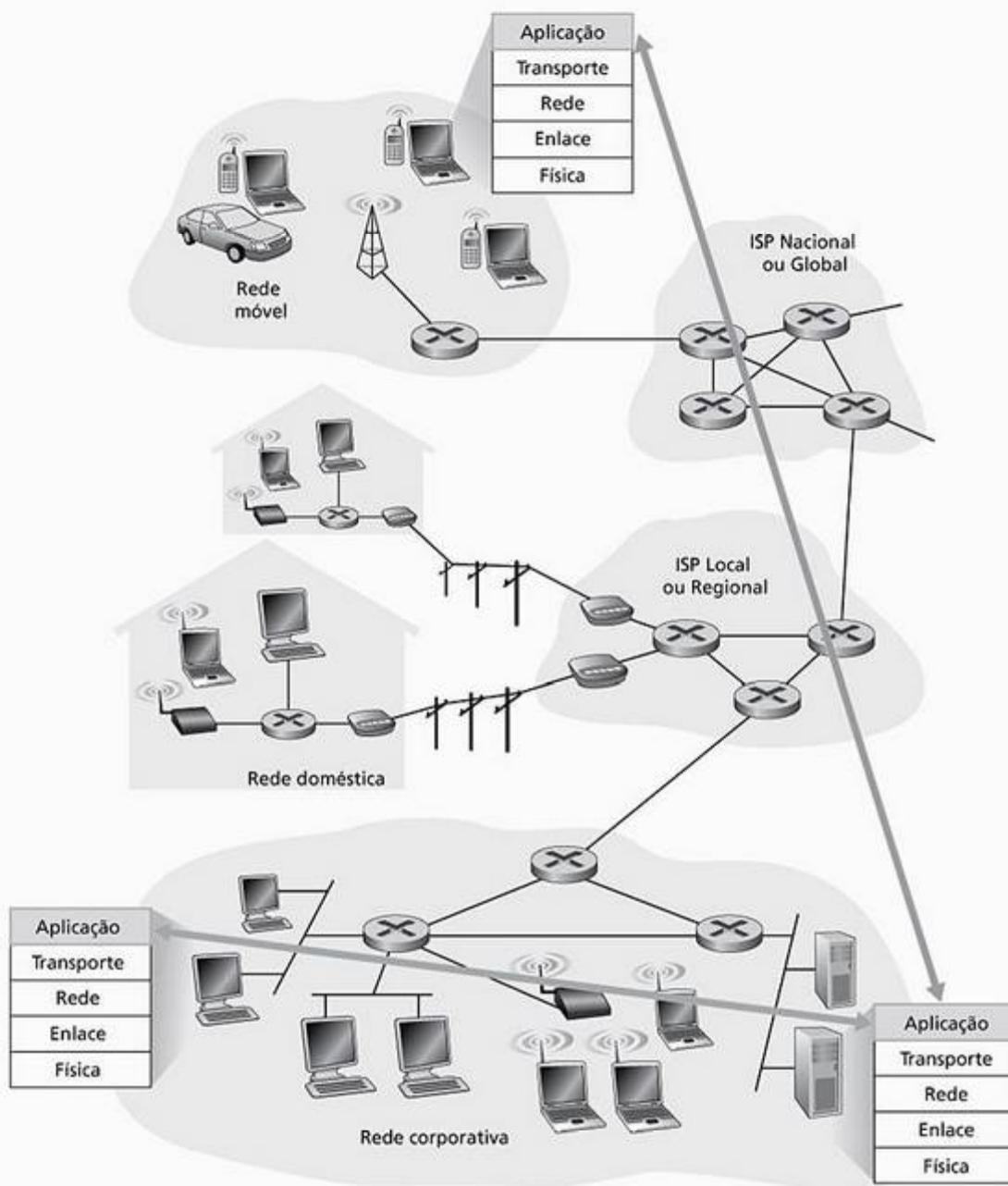


Figura 2.1 A comunicação de uma aplicação de rede ocorre entre sistemas finais na camada de aplicação

alternadamente, denominados *pares*. Os *pares* não são de propriedade dos provedores de serviço, mas são controlados por usuários de computadores de mesa e laptops, sendo que a maioria dos pares se aloja em residências, universidades e escritórios. Como os pares se comunicam sem passar por nenhum servidor dedicado, a arquitetura é denominada par-a-par (*par-to-par* — P2P). Muitas das aplicações de hoje mais populares e de intenso tráfego são baseadas nas arquiteturas P2P, incluindo distribuição de arquivos (por exemplo, BitTorrent), compartilhamento de arquivo (por exemplo, eMule e LimeWire), telefonia por Internet (por exemplo, Skype) e IPTV (por exemplo, PPStream). Essa arquitetura está ilustrada na Figura 2.2 (b). Mencionamos que algumas aplicações possuem arquiteturas híbridas, combinando elementos cliente-servidor e P2P. Por exemplo, para muitas aplicações de mensagem instantânea, os servidores costumam

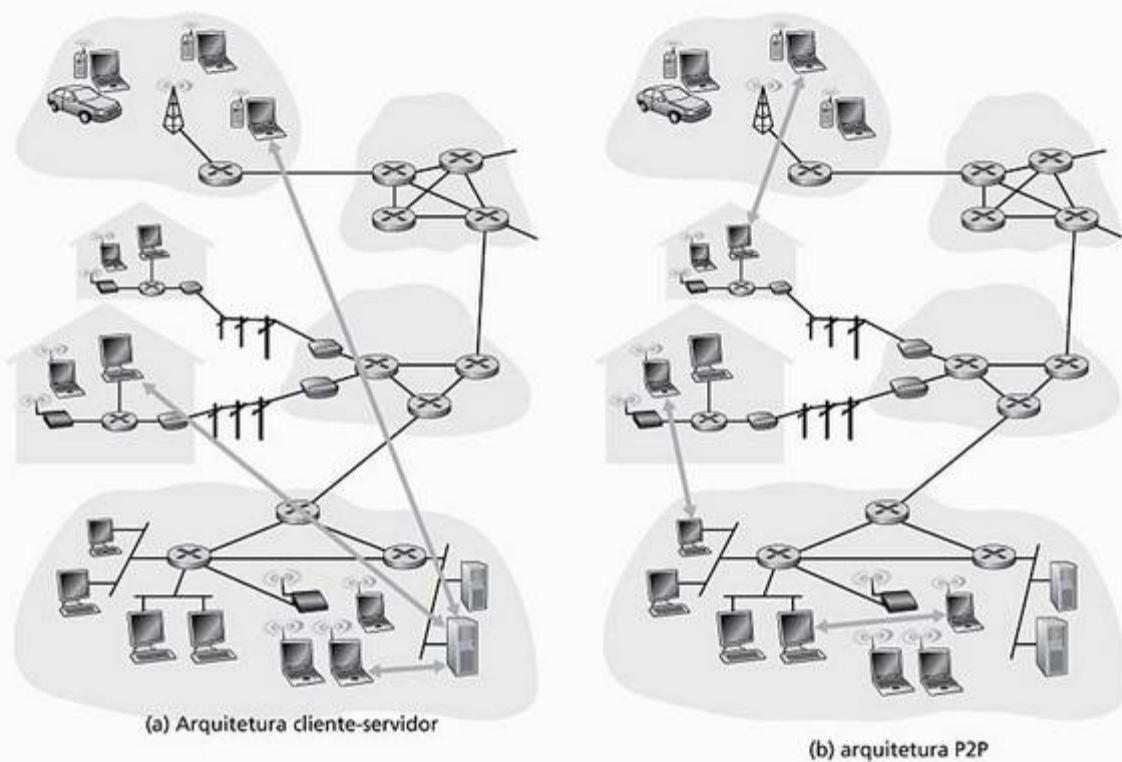


Figura 2.2 (a) Arquitetura cliente-servidor; (b) arquitetura P2P

rastrear o endereço IP dos usuários, mas as mensagens de usuário para usuário são enviadas diretamente entre os hospedeiros do usuário (sem passar por servidores intermediários). Uma das características mais fortes da arquitetura P2P é sua autoescalabilidade. Por exemplo, em uma aplicação de compartilhamento de arquivo P2P, embora cada par gere uma carga de trabalho solicitando arquivos, cada par também acrescenta capacidade de serviço ao sistema distribuindo arquivos a outros pares. As arquiteturas P2P também possuem uma boa relação custo-benefício, visto que normalmente não requerem uma infraestrutura de servidor significativa e uma largura de banda de servidor. A fim de reduzir os custos, os provedores de serviço (MSN, Yahoo etc.) estão cada vez mais interessados em utilizar arquiteturas P2P para suas aplicações [Chuang, 2007]. Entretanto, as futuras aplicações P2P estão diante de três principais desafios:

1. **ISP Amigável.** A maioria dos ISPs residenciais (incluindo o DSL e os ISPs a cabo) foram dimensionados para uso de largura de banda assimétrica, ou seja, para muito mais tráfego de entrada do que de saída. Mas a transmissão de vídeo P2P e as aplicações de distribuição de vídeo transferem o tráfego de saída dos servidores para ISPs residenciais, colocando, assim, uma pressão significativa nos ISPs. As futuras aplicações P2P precisam ser criadas para que sejam amigáveis aos ISPs [Xie, 2008].
2. **Segurança.** Em razão de sua natureza altamente distribuída e exposta, as aplicações P2P podem ser um desafio para proteger [Doucer, 2002; Yu, 2006; Liang, 2006; Naoumov, 2006; Dhungel, 2008].
3. **Incentivos.** O sucesso das futuras aplicações P2P também depende de usuários participativos para oferecer largura de banda, armazenamento e recursos da computação às aplicações, um projeto desafiador de incentivo [Feldman, 2005; Piatek, 2008; Aperjis, 2008].

2.1.2 Comunicação entre processos

Antes de construir sua aplicação de rede, você também precisará ter um entendimento básico de como programas que rodam em vários sistemas finais comunicam-se entre si. No jargão de sistemas

operacionais, na verdade não são programas, mas **processos que se comunicam**. Um processo pode ser imaginado como um programa que está rodando dentro de um sistema final. Quando os processos estão rodando no mesmo sistema final, eles comunicam-se entre si usando comunicação interprocessos, cujas regras são determinadas pelo sistema operacional do sistema final. Porém, neste livro, não estamos interessados em como se comunicam processos que estão no mesmo hospedeiro, mas em como se comunicam processos que rodam em sistemas finais *diferentes* (com sistemas operacionais potencialmente diferentes).

Eles se comunicam pela troca de mensagens por meio da rede de computadores. Um processo originador cria e envia mensagens para a rede; um processo destinatário recebe-as e possivelmente responde, devolvendo outras. A Figura 2.1 mostra que processos se comunicam usando a camada de aplicação da pilha de cinco camadas da arquitetura.

Processos clientes e processos servidores

Uma aplicação de rede consiste em pares de processos que enviam mensagens uns para os outros por meio de uma rede. Por exemplo, na aplicação Web, o processo browser de um cliente troca mensagens com o processo de um servidor Web. Em um sistema de compartilhamento de arquivos P2P, um arquivo é transferido de um processo que está em um par para outro que está em outro par. Para cada par de processos comunicantes normalmente rotulamos um dos dois processos de cliente e o outro, de servidor. Na Web, um browser é um processo cliente e um servidor Web é um processo servidor. No compartilhamento de arquivos P2P, o par que está enviando o arquivo é rotulado de cliente e o que está recebendo, de servidor.

Talvez você já tenha observado que, em algumas aplicações, tal como compartilhamento de arquivos P2P, um processo pode ser ambos, cliente e servidor. Realmente, um processo em um sistema de compartilhamento de arquivos P2P pode carregar e descarregar arquivos. Mesmo assim, no contexto de qualquer dada sessão entre um par de processos, ainda podemos rotular um processo de cliente e o outro de servidor. Definimos os processos cliente e servidor como segue:

*No contexto de uma sessão de comunicação entre um par de processos, o processo que inicia a comunicação (isto é, o primeiro a contatar o outro no início da sessão) é rotulado de **cliente**. O processo que espera ser contatado para iniciar a sessão é o **servidor**.*

Na Web, um processo do browser inicia o contato com um processo do servidor Web; por conseguinte, o processo do browser é o cliente e o processo do servidor Web é o servidor. No compartilhamento de arquivos P2P, quando o Par A solicita ao Par B o envio de um arquivo específico, o Par A é o cliente enquanto o Par B é o servidor no contexto dessa sessão específica de comunicação. Quando não houver possibilidade de confusão, às vezes usaremos também a terminologia “lado cliente e lado servidor de uma aplicação”. No final deste capítulo examinaremos passo a passo um código simples para ambos os lados de aplicações de rede: o lado cliente e o lado servidor.

A interface entre o processo e a rede de computadores

Como dissemos anteriormente, a maioria das aplicações consiste em pares de processos comunicantes, sendo que os dois processos de cada par enviam mensagens um para o outro. Qualquer mensagem enviada de um processo para um outro tem de passar pela rede subjacente. Um processo envia mensagens para a rede e recebe mensagens dela através de uma interface de software denominada socket. Vamos considerar uma analogia que nos auxiliará a entender processos e sockets. Um processo é análogo a uma casa e seu socket, à porta da casa. Quando um processo quer enviar uma mensagem a um outro processo em outro hospedeiro, ele empurra a mensagem pela porta (socket). Esse processo emissor admite que exista uma infraestrutura de transporte do outro lado de sua porta que transportará a mensagem pela rede até a porta do processo destinatário. Ao chegar ao hospedeiro destinatário, a mensagem passa através da porta (socket) do processo receptor, que então executa alguma ação sobre a mensagem.

A Figura 2.3 ilustra a comunicação por socket entre dois processos que se comunicam pela Internet. (A Figura 2.3 admite que o protocolo de transporte subjacente usado pelos processos é o TCP.) Como mostra essa figura, um socket é a interface entre a camada de aplicação e a de transporte dentro de uma máquina. É também denominado interface de programação da aplicação (*application programming interface — API*) entre a aplicação e a rede, visto que o socket é a interface de programação pela qual as aplicações de rede são inseridas na Internet. O desenvolvedor da aplicação controla tudo o que existe no lado da camada de aplicação do socket, mas tem pouco controle do lado da camada de transporte do socket. Os únicos controles que o desenvolvedor da aplicação tem do lado da camada de transporte são: (1) a escolha do protocolo de transporte e (2), talvez, a capacidade de determinar alguns parâmetros da camada de transporte, tais como tamanho máximo de buffer e de segmentos (a serem abordados no Capítulo 3). Uma vez escolhido um protocolo de transporte, (se houver escolha) o desenvolvedor constrói a aplicação usando os serviços da camada de transporte oferecidos por esse protocolo. Examinaremos sockets mais detalhadamente nas seções 2.7 e 2.8.

2.1.3 Serviços de transporte disponíveis para aplicações

Lembre-se de que um socket é a interface entre o processo da aplicação e o protocolo de camada de transporte. A aplicação do lado remetente envia mensagens através do socket. Do outro lado do socket, o protocolo de camada de transporte tem a responsabilidade de levar as mensagens pela rede até a “porta” do socket destinatário. Muitas redes, inclusive a Internet, oferecem mais de um protocolo de camada de transporte. Ao desenvolver uma aplicação, você deve escolher um dos protocolos de camada de transporte disponíveis. Como fazer essa escolha? O mais provável é que você avalie os serviços providos pelos protocolos de camada de transporte disponíveis e escolha o protocolo que melhor atenda às necessidades de sua aplicação. A situação é semelhante a escolher trem ou avião como meio de transporte entre duas cidades. Você tem de escolher um ou outro, e cada modalidade de transporte oferece serviços diferentes. Por exemplo, o trem oferece a facilidade da partida e da chegada no centro da cidade, ao passo que o avião oferece menor tempo de viagem.

Quais são os serviços que um protocolo da camada de transporte pode oferecer às aplicações que o chamem? Podemos classificar, de maneira geral, os possíveis serviços segundo quatro dimensões: transferência confiável de dados, vazão, temporização e segurança.

Transferência confiável de dados

Como discutido no Capítulo 1, os pacotes podem se perder dentro de uma rede de computador. Um pacote pode, por exemplo, exceder um buffer em um roteador, ou ser descartado por um hospedeiro ou um roteador

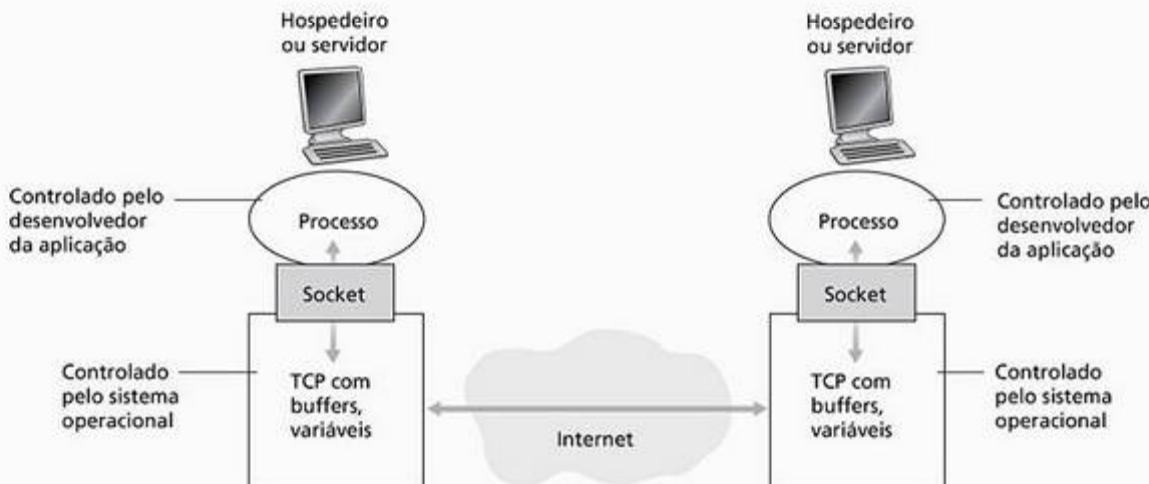


Figura 2.3 Processos de aplicação, sockets e protocolo de transporte subjacente

após alguns de seus bits terem sido corrompidos. Para muitas aplicações — como correio eletrônico, transferência de arquivo, acesso remoto, transferências de documentos da Web e aplicações financeiras — a perda de dados pode ter consequências devastadoras (no último caso, para o banco e para o cliente!). Assim, para suportar essas aplicações, algo deve ser feito para garantir que os dados enviados por uma extremidade da aplicação sejam enviados correta e completamente para a outra extremidade da aplicação. Se um protocolo fornecer um serviço de recebimento de dados garantidos, ele fornecerá uma transferência confiável de dados. Um importante serviço que o protocolo da camada de transporte pode oferecer para uma aplicação é a transferência confiável de dados processo a processo. Quando um protocolo de transporte oferece esse serviço, o processo remetente pode passar seus dados para um socket e saber com absoluta confiança que os dados chegarão sem erro ao processo destinatário.

Quando um protocolo da camada de transporte não oferece uma transferência confiável de dados, os dados enviados pelo processo remetente podem nunca chegar ao processo destinatário. Isso pode ser aceitável para aplicações tolerantes a perda, aplicações de multimídia como áudio/vídeo em tempo real ou áudio/vídeo armazenado, que podem tolerar uma quantidade de perda de dados. Nessas aplicações multimídia, dados perdidos podem resultar em uma pequena falha durante a execução do áudio/vídeo — o que não é um prejuízo crucial.

Vazão

No Capítulo 1 apresentamos o conceito de vazão disponível, que, no contexto de sessão da comunicação entre dois processos ao longo de um caminho de rede, é a taxa à qual o processo remetente pode enviar bits ao processo destinatário. Como outras sessões compartilharão a largura de banda no caminho da rede e estão indo e voltando, a vazão disponível pode oscilar com o tempo. Essas observações levam a outro serviço natural que um protocolo da camada de transporte pode oferecer, ou seja, uma vazão disponível garantida a uma taxa específica. Com tal serviço, a aplicação pode solicitar uma vazão garantida de r bits/s, e o protocolo de transporte garante, então, que a vazão disponível seja sempre r bits/s, pelo menos. Tal serviço de vazão garantida aparece em muitas aplicações.

Por exemplo, se uma aplicação de telefonia por Internet codifica voz a 32 bits/s, ela precisa enviar dados para a rede e fazer com que sejam entregues na aplicação receptora a essa mesma taxa. Se o protocolo de transporte não puder fornecer essa vazão, a aplicação precisará codificar a uma taxa menor (e receber vazão suficiente para sustentar essa taxa de codificação mais baixa) ou então desistir, já que receber metade da vazão de que precisa de nada adianta para essa **aplicação** de telefonia por Internet. Aplicações que possuam necessidade de vazão são conhecidas como aplicações sensíveis à largura de banda. Muitas aplicações de multimídia existentes são sensíveis à largura de banda, embora algumas poderão usar técnicas adaptativas de codificação para codificar a uma taxa que corresponda à vazão disponível na ocasião. Embora aplicações sensíveis à largura de banda possuam necessidades específicas de vazão, as **aplicações elásticas** podem fazer uso de qualquer quantidade mínima ou máxima que por acaso esteja disponível. Correio eletrônico, transferência de arquivos e transferências Web são todas aplicações elásticas. Evidentemente, quanto mais vazão, melhor. Há um ditado que diz que “dinheiro nunca é demais”; nesse caso, podemos dizer que vazão nunca é demais!

Temporização

Um protocolo da camada de transporte pode também oferecer garantias de temporização. Como na garantia de vazão, as garantias de temporização podem surgir em diversos aspectos e modos. Podemos citar como exemplo o fato de que cada bit que o remetente insere no socket chega ao socket destinatário em menos de 100 milissegundos depois. Esse serviço seria atrativo para aplicações interativas em tempo real, como a telefonia por Internet, ambientes virtuais, teleconferência e jogos multijogadores, que exigem restrições de temporização no envio de dados para garantir eficácia. (Veja Capítulo 7, [Gauthier, 1999; Ramjee, 1994].) Longos atrasos na telefonia por Internet, por exemplo, tendem a resultar em pausas artificiais na conversação; em um jogo multiusuário ou ambiente virtual interativo, um longo atraso entre realizar uma ação e ver a reação do ambiente (por exemplo, a reação de um outro jogador na outra extremidade de uma conexão fim a fim) faz com que a aplicação pareça menos realista. Para aplicações que não são em tempo real, é sempre preferível um atraso menor a um maior, mas não há nenhuma limitação estrita aos atrasos fim a fim.

Segurança

Por fim, um protocolo de transporte pode oferecer uma aplicação com um ou mais serviços de segurança. Por exemplo, no hospedeiro remetente, um protocolo de transporte pode codificar todos os dados transmitidos pelo processo remetente e, no hospedeiro destinatário, o protocolo da camada de transporte pode codificar os dados antes de enviá-los ao processo destinatário. Tal serviço pode oferecer sigilo entre os dois processos, mesmo que os dados sejam, de algum modo, observados entre os processos remetente e destinatário. Um protocolo de transporte pode, além do sigilo, fornecer outros serviços de segurança, incluindo integridade dos dados e autenticação do ponto terminal, assuntos que serão abordados em detalhes no Capítulo 8.

2.1.4 Serviços de transporte provisórios pela Internet

Até aqui, consideramos serviços de transportes que uma rede de computadores poderia oferecer em geral. Vamos agora nos aprofundar mais no assunto e analisar o tipo de suporte de aplicação provido pela Internet. A Internet (e, em um amplo sentido, as redes TCP/IP) disponibiliza dois protocolos de transporte para aplicações, o UDP e o TCP. Quando você (como um criador de aplicação) cria uma nova aplicação de rede para a Internet, uma das primeiras decisões a ser tomada é usar o UDP ou o TCP. Cada um desses protocolos oferece um conjunto diferente de serviços para as aplicações solicitantes. A Figura 2.4 mostra os requisitos do serviço para algumas aplicações selecionadas.

Serviços do TCP

O modelo de serviço TCP inclui um serviço orientado para conexão e um serviço confiável de transferência de dados. Quando uma aplicação solicita o TCP como seu protocolo de transporte, recebe dele ambos os serviços.

Serviço orientado para conexão: o TCP faz com que o cliente e o servidor troquem informações de controle de camada de transporte antes que as mensagens de camada de aplicação começem a fluir. Esse procedimento de apresentação, por assim dizer, alerta o cliente e o servidor, permitindo que eles se preparem para uma enxurrada de pacotes. Após a fase de apresentação, dizemos que existe uma **conexão TCP** entre os sockets dos dois processos. A conexão é full-duplex (simultânea), visto que os dois processos podem enviar mensagens um ao outro pela conexão ao mesmo tempo. Quando termina de enviar mensagens, a aplicação deve interromper a conexão. Esse serviço é chamado de serviço “orientado para conexão”, e não serviço “de conexão”, porque os dois processos estão conectados de um modo muito solto. No Capítulo 3, discutiremos detalhadamente serviço orientado para conexão e examinaremos como ele é implementado.

Serviço confiável de transporte: os processos comunicantes podem confiar no TCP para a entrega de todos os dados enviados sem erro e na ordem correta. Quando um lado da aplicação passa uma cadeia

Aplicação	Perda de dados	Largura de banda	Sensibilidade ao atraso
Transferência de arquivos	Sem perda	Elástica	Não
E-mail	Sem perda	Elástica	Não
Documentos Web	Sem perda	Elástica (alguns kbps)	Não
Telefonia via Internet/videoconferência	Tolerante à perda	Áudio: alguns kbps – 1 Mbps Vídeo: 10 kbps – 5 Mbps	Sim: décimos de segundo
Áudio/vídeo armazenado	Tolerante à perda	Igual acima	Sim: alguns segundos
Jogos interativos	Tolerante à perda	Alguns kbps – 10 Mbps	Sim: décimos de segundo
Mensagem instantânea	Sem perda	Elástica	Sim e não

Figura 2.4 Requisitos de aplicações de rede selecionadas

de bytes para dentro de um socket, pode contar com o TCP para entregar a mesma cadeia de dados ao socket receptor, sem falta de bytes nem bytes duplicados.

O TCP também inclui um mecanismo de controle de congestionamento, um serviço voltado ao bem-estar geral da Internet e não ao benefício direto dos processos comunicantes. O mecanismo de controle de congestionamento do TCP limita a capacidade de transmissão de um processo (cliente ou servidor) quando a rede está congestionada entre cliente e servidor. Em particular, como veremos no Capítulo 3, o controle de congestionamento do TCP tenta limitar cada conexão do TCP à sua justa porção de largura de banda de rede. A limitação da velocidade de transmissão pode ter um efeito muito prejudicial sobre aplicações de áudio e vídeo em tempo real que imponham uma limitação de largura de banda mínima. Além disso, aplicações em tempo real são tolerantes à perda e não necessitam de um serviço de transporte totalmente confiável. Por essas razões, desenvolvedores de aplicações em tempo real usualmente executam suas aplicações em UDP, e não em TCP.

Serviços do UDP

O UDP é um protocolo de transporte simplificado, leve, com um modelo de serviço minimalista. É um serviço não orientado para conexão; portanto, não há apresentação antes que os dois processos começem a se comunicar. O UDP provê um serviço não confiável de transferência de dados — isto é, quando um processo envia uma mensagem para dentro de um socket UDP, o protocolo não oferece *nenhuma* garantia de que a mensagem chegará ao processo receptor. Além do mais, mensagens que realmente chegam ao processo receptor podem chegar fora de ordem.

O UDP não inclui um mecanismo de controle de congestionamento; portanto, um processo originador pode bombar dados para dentro de uma camada abaixo (a camada de rede) à taxa que quiser. (Observe, entretanto, que a vazão fim a fim pode ser menor do que essa taxa devido à largura de banda limitada de enlaces intervenientes ou ao congestionamento). Como aplicações em tempo real usualmente podem tolerar uma certa perda



Segurança em foco

**Um conjunto impressionante de sistemas finais da Internet
protegendo o TCP**

Nem o TCP ou o UDP fornecem qualquer codificação — os dados que o processo remetente transfere para seu socket são os mesmos que percorrem a rede até o processo destinatário. Então, por exemplo, se o processo destinatário enviar uma senha em cleartext (ou seja, não codificado) para seu socket, ela percorrerá por todos os enlaces entre o remetente e o destinatário, sendo analisada e descoberta em qualquer um dos enlaces intervenientes. Em razão de a privacidade e outras questões de segurança terem se tornado importantes para muitas aplicações, a comunidade da Internet desenvolveu um aperfeiçoamento para o TCP, denominado Camada de Sockets Seguros (SSL). O aperfeiçoamento SSL para o TCP não somente faz tudo que o TCP tradicional faz, como também oferece serviços importantes de segurança processo a processo, incluindo codificação, integridade dos dados e autenticação do ponto de chegada. Salientamos que o SSL não é um terceiro protocolo da Internet, no mesmo nível do TCP e do UDP, mas um aperfeiçoamento do TCP implementado na camada de aplicação. Particularmente, se uma aplicação quiser utilizar o serviço do SSL, é preciso incluir o código SSL (existente, bibliotecas altamente otimizadas e as classes) da aplicação em ambas as partes cliente e servidor. O SSL possui sua própria API de socket que é semelhante à tradicional API de socket TCP. Quando uma aplicação utiliza o SSL, o processo remetente transfere dados em cleartext para o socket SSL; no hospedeiro emissor, então, o SSL codifica os dados e os passa para o socket TCP. Os dados codificados percorrem a Internet até o socket TCP no processo remetente. O socket remetente passa os dados codificados ao SSL, que os decodifica. Por fim, o SSL passa os dados em cleartext por seu socket SSL até o processo destinatário. Abordaremos o SSL em mais detalhes no Capítulo 8.

de dados, mas exigem uma taxa mínima, desenvolvedores desse tipo de aplicações frequentemente preferem executá-las por UDP, evitando, assim, o controle de congestionamento e os cabeçalhos de pacotes do TCP. Por outro lado, como muitos firewalls estão configurados para bloquear (a maioria dos tipos de) tráfego UDP, os projetistas têm escolhido, cada vez mais, executar aplicações multimídia e em tempo real por meio do TCP [Sripanidkulchai, 2004].

Serviços não providos pelos protocolos de transporte da Internet

Organizamos os serviços do protocolo de transporte em quatro dimensões: transferência confiável de dados, vazão, temporização e segurança. Quais desses serviços são providos pelo TCP e pelo UDP? Já vimos que o TCP provê a transferência confiável de dados fim a fim, e sabemos também que ele pode ser aprimorado facilmente na camada de aplicação com o SSL para oferecer serviços de segurança. Mas em nossa breve descrição sobre o TCP e o UDP, faltou mencionar as garantias de vazão e de temporização — serviços não fornecidos pelos protocolos de transporte da Internet de hoje. Isso significa que as aplicações sensíveis ao tempo, como a telefonia por Internet não podem rodar na Internet atual? A resposta é claramente negativa — a Internet tem recebido essas aplicações por muitos anos. Tais aplicações muitas vezes funcionam bem, por terem sido desenvolvidas para lidar, na medida do possível, com essa falta de garantia. Analisaremos vários desses artifícios de criação no Capítulo 7. No entanto, a criação engenhosa possui suas limitações quando o atraso é excessivo, como é o caso frequente da Internet pública. Em resumo, a Internet de hoje pode oferecer serviços satisfatórios a aplicações sensíveis ao tempo, mas não garantias de temporização ou de largura de banda.

A Figura 2.5 mostra os protocolos de transporte usados por algumas aplicações populares da Internet. Vemos que e-mail, a Web, acesso a terminais remotos e transferência de arquivos usam o TCP. Essas aplicações escolheram o TCP primordialmente porque ele oferece um serviço confiável de transferência de dados, garantindo que todos eles mais cedo ou mais tarde cheguem a seu destino. Vemos também que a aplicação de telefonia por Internet normalmente funciona em UDP. Cada lado de uma aplicação de telefone por Internet precisa enviar dados pela rede a uma taxa mínima (veja a Figura 2.4), o que é mais provavelmente possível com UDP do que com TCP. E, também, aplicações de telefone por Internet são tolerantes às perdas, de modo que não necessitam do serviço confiável de transferência de dados provido pelo TCP.

Endereçamento de processos

Nossa discussão acima se focou nos serviços de transporte entre dois processos de comunicação. Mas como um processo indica qual processo ele quer para se comunicar usando esses serviços? Como um processo rodando em um hospedeiro em Amherst, Massachusetts, EUA, especifica que ele quer se comunicar com um processo em particular rodando em um hospedeiro em Bangkok, na Tailândia? Para isso, é preciso especificar duas informações: (1) o nome ou o endereço da máquina hospedeira e (2) um identificador que especifique o processo destinatário no hospedeiro de destino.

Aplicações	Protocolo de camada de aplicação	Protocolo de transporte subjacente
Correio eletrônico	SMTP (RFC 5321)	TCP
Acesso a terminal remoto	Telnet (RFC 854)	TCP
Web	HTTP (RFC 2616)	TCP
Transferência de arquivos	FTP (RFC 959)	TCP
Multimídia em tempo real	HTTP (por exemplo, YouTube), RTP	UDP ou TCP
Telefonia por Internet	SIP, RTP ou proprietária (por exemplo, Skype)	Tipicamente UDP

Figura 2.5 Aplicações populares da Internet, seus protocolos de camada de aplicação e seus protocolos de transporte subjacentes

Na Internet, o hospedeiro é identificado por seu endereço IP. Discutiremos endereços IP em detalhes no Capítulo 4. Por enquanto, basta saber que o endereço IP é uma quantidade de 32 bits que identifica *exclusivamente* o sistema final. (Entretanto, como veremos no Capítulo 4, a implementação ampla do NAT (Tradução de Endereços de Rede) significa que, na prática, um endereço IP de 32 bits sozinho não endereça exclusivamente um hospedeiro.)

Além de saber o endereço do sistema final ao qual a mensagem se destina, o hospedeiro originador também deve identificar o processo que está rodando no outro hospedeiro. Essa informação é necessária porque, em geral, um hospedeiro poderia estar executando muitas aplicações de rede. Um **número de porta** de destino atende a essa finalidade. Aplicações populares receberam números de porta específicos. Por exemplo, um servidor Web é identificado pelo número de porta 80. Um processo servidor de correio (que usa o protocolo SMTP) é identificado pelo número de porta 25. Uma lista de números bem conhecidos de portas para todos os protocolos padronizados da Internet pode ser encontrada no site <http://www.iana.org>. Quando um desenvolvedor cria uma nova aplicação de rede, ela deve receber um novo número de porta. Examinaremos números de porta detalhadamente no Capítulo 3.

2.1.5 Protocolos de camada de aplicação

Acabamos de aprender que processos de rede comunicam-se entre si enviando mensagens para dentro de sockets. Mas, como essas mensagens são estruturadas? O que significam os vários campos nas mensagens? Quando os processos enviam as mensagens? Essas perguntas nos transportam para o mundo dos protocolos de camada de aplicação. Um **protocolo de camada de aplicação** define como processos de uma aplicação, que funcionam em sistemas finais diferentes, passam mensagens entre si. Em particular, um protocolo de camada de aplicação define:

- os tipos de mensagens trocadas, por exemplo, de requisição e de resposta;
- a sintaxe dos vários tipos de mensagens, tais como os campos da mensagem e como os campos são delineados;
- a semântica dos campos, isto é, o significado da informação nos campos;
- regras para determinar quando e como um processo envia mensagens e responde a mensagens.

Alguns protocolos de camada de aplicação estão especificados em RFCs e, portanto, são de domínio público. Por exemplo, o protocolo de camada de aplicação da Web, HTTP (HyperText Transfer Protocol [RFC 2616]), está à disposição como um RFC. Se um desenvolvedor de browser seguir as regras do RFC do HTTP, seu browser estará habilitado a extrair páginas de qualquer servidor Web que também tenha seguido as regras do RFC do HTTP. Muitos outros protocolos de camada de aplicação são proprietários e, intencionalmente, não estão disponíveis ao público. Por exemplo, muitos dos sistemas de compartilhamento de arquivos P2P existentes usam protocolos de camada de aplicação proprietários.

É importante distinguir aplicações de rede de protocolos de camada de aplicação. Um protocolo de camada de aplicação é apenas um pedaço (embora grande) de aplicação de rede. Examinemos alguns exemplos. A Web é uma aplicação cliente-servidor que permite aos usuários obter documentos de servidores Web por demanda. A aplicação Web consiste em muitos componentes, entre eles um padrão para formato de documentos (isto é, HTML), browsers Web (por exemplo, Firefox e Microsoft Internet Explorer), servidores Web (por exemplo, servidores Apache e Microsoft) e um protocolo de camada de aplicação. O protocolo de camada de aplicação da Web, HTTP, define o formato e a sequência das mensagens que são passadas entre o browser e o servidor Web. Assim, ele é apenas um pedaço (embora importante) da aplicação Web. Como outro exemplo, considere a aplicação correio eletrônico da Internet que também tem muitos componentes, entre eles servidores de correio que armazenam caixas postais de usuários, leitores de correio que permitem aos usuários ler e criar mensagens, um padrão que define como mensagens são passadas entre servidores e entre servidores e leitores de correio e como o conteúdo de certas partes da mensagem de correio (por exemplo, um cabeçalho) deve ser interpretado. O principal protocolo de camada de aplicação para o correio eletrônico é o SMTP — Simple Mail Protocol [RFC 5321]. Assim, o SMTP é apenas um pedaço (embora importante) da aplicação correio eletrônico.

2.1.6 Aplicações de rede abordadas neste livro

Novas aplicações de Internet de domínio público e proprietárias são desenvolvidas todos os dias. Em vez de tratarmos de um grande número dessas aplicações de maneira enciclopédica, preferimos focalizar um pequeno número de aplicações ao mesmo tempo importantes e populares. Neste capítulo, discutiremos cinco aplicações populares: a Web, a transferência de arquivos, o correio eletrônico, o serviço de diretório e aplicações P2P. Discutiremos primeiramente a Web não somente porque ela é uma aplicação imensamente popular, mas também porque seu protocolo de camada de aplicação, HTTP, é direto e fácil de entender. Após examinarmos a Web, examinaremos brevemente o FTP porque este protocolo oferece um ótimo contraste com o HTTP. Em seguida, discutiremos o correio eletrônico, a primeira aplicação de enorme sucesso da Internet. O correio eletrônico é mais complexo do que a Web, no sentido de que usa não somente um, mas vários protocolos de camada de aplicação. Após o e-mail, estudaremos o DNS, que provê um serviço de diretório para a Internet. A maioria dos usuários não interage diretamente com o DNS; em vez disso, eles o chamam indiretamente por meio de outras aplicações (inclusive a Web, a transferência de arquivos e o correio eletrônico). O DNS ilustra primorosamente como um componente de funcionalidade de núcleo de rede (tradução de nome de rede para endereço de rede) pode ser implementado na camada de aplicação da Internet. Finalmente, discutiremos o compartilhamento de arquivos P2P que, segundo algumas medições (por exemplo, tráfego de rede), é a classe de aplicações mais popular da Internet de hoje. Finalmente, discutiremos várias aplicações P2P, incluindo distribuição de arquivo, banco de dados distribuídos e telefonia IP.

2.2 A Web e o HTTP

Até a década de 1990, a Internet era usada primordialmente por pesquisadores, acadêmicos e estudantes universitários para se interligar com hospedeiros remotos, transferir arquivos de hospedeiros locais para hospedeiros remotos e vice-versa, enviar e receber notícias e enviar e receber correio eletrônico. Embora essas aplicações fossem (e continuem a ser) extremamente úteis, a Internet não era conhecida fora das comunidades acadêmicas e de pesquisa. Então, no início da década de 1990, entrou em cena uma nova aplicação importissíma — a World Wide Web [Berners-Lee, 1994]. A Web é a aplicação da Internet que chamou a atenção do público em geral. Ela transformou drasticamente a maneira como pessoas interagem dentro e fora de seus ambientes de trabalho. Alçou a Internet de apenas uma entre muitas redes de dados para, essencialmente, a única rede de dados.

Talvez o que mais atraia a maioria dos usuários da Web é que ela funciona *por demanda*. Usuários recebem o que querem, quando querem, o que é diferente da transmissão de rádio e de televisão, que força o usuário a sintonizar quando o provedor disponibiliza o conteúdo. Além de funcionar por demanda, a Web tem muitas outras características maravilhosas que as pessoas adoram. É muitíssimo fácil para qualquer indivíduo fazer com que informações fiquem disponíveis na Web; todo mundo pode se transformar em editor a um custo extremamente baixo. Hiperenlaces e buscadores nos ajudam a navegar pelo oceano dos sites Web. Dispositivos gráficos estimulam nossos sentidos. Formulários, applets Java e muitos outros dispositivos nos habilitam a interagir com páginas e sites. E, cada vez mais, a Web oferece um menu de interfaces para vastas quantidades de material de vídeo e áudio armazenadas na Internet — áudio e vídeo que podem ser acessados por demanda.

2.2.1 Descrição geral do HTTP

O **HTTP — Protocolo de Transferência de Hipertexto** (HyperText Transfer Protocol) — protocolo de camada de aplicação da Web, está no coração da Web e é definido no [RFC 1945] e no [RFC 2616]. O HTTP é implementado em dois programas: um programa cliente e outro servidor. Os dois programas, executados em sistemas finais diferentes, conversam um com o outro por meio da troca de mensagens HTTP. O HTTP define a

estrutura dessas mensagens e o modo como o cliente e o servidor as trocam. Antes de explicarmos detalhadamente o HTTP, devemos revisar a terminologia da Web.

Uma **página Web** (também denominada documento) é constituída de objetos. Um **objeto** é simplesmente um arquivo — tal como um arquivo HTML, uma imagem JPEG, um applet Java, ou um clipe de vídeo — que se pode acessar com um único URL. A maioria das páginas Web é constituída de um **arquivo-base HTML** e diversos objetos referenciados. Por exemplo, se uma página Web contiver um texto HTML e cinco imagens JPEG, então ela terá seis objetos: o arquivo-base HTML e mais as cinco imagens. O arquivo-base HTML referencia os outros objetos na página com os URLs dos objetos. Cada URL tem dois componentes: o nome do hospedeiro do servidor que abriga o objeto e o nome do caminho do objeto. Por exemplo, no URL

`http://www.someSchool.edu/someDepartment/picture.gif`

`www.someSchool.edu` é o nome do hospedeiro e `/someDepartment/picture.gif` é o do caminho. Como browsers Web também implementam o lado cliente do HTTP, podemos usar as palavras *browser* e *cliente* indiferentemente no contexto da Web. Os servidores Web também implementam o lado servidor do HTTP, abrigam objetos Web, cada um endereçado por um URL. São servidores Web populares o Apache e o Microsoft Internet Information Server.

O HTTP define como clientes Web requisitam páginas Web aos servidores e como eles as transferem a clientes. Discutiremos detalhadamente a interação entre cliente e servidor mais adiante, mas a ideia geral está ilustrada na Figura 2.6. Quando um usuário requisita uma página Web (por exemplo, clica sobre um hiperenlace), o browser envia ao servidor mensagens de requisição HTTP para os objetos da página. O servidor recebe as requisições e responde com mensagens de resposta HTTP que contêm os objetos.

O HTTP usa o TCP como seu protocolo de transporte subjacente (em vez de rodar em UDP). O cliente HTTP primeiramente inicia uma conexão TCP com o servidor. Uma vez estabelecida a conexão, os processos do browser e do servidor acessam o TCP por meio de suas interfaces sockets. Como descrito na Seção 2.1, no lado cliente a interface socket é a porta entre o processo cliente e a conexão TCP; no lado servidor, ela é a porta entre o processo servidor e a conexão TCP.

O cliente envia mensagens de requisição HTTP para sua interface socket e recebe mensagens de resposta HTTP de sua interface socket. De maneira semelhante, o servidor HTTP recebe mensagens de requisição de sua interface socket e envia mensagens de resposta para sua interface socket. Assim que o cliente envia uma mensagem para sua interface socket, a mensagem sai de suas mãos e “passa para as mãos do TCP”. Lembre-se de que na Seção 2.1 dissemos que o TCP provê ao HTTP um serviço confiável de transferência de dados, o que implica que toda mensagem de requisição HTTP emitida por um processo cliente chegará intacta ao servidor. De maneira semelhante, toda mensagem de resposta HTTP emitida pelo processo servidor chegará intacta ao cliente. Percebemos, nesse ponto, uma das grandes vantagens de uma arquitetura de camadas — o



Figura 2.6 Comportamento de requisição-resposta do HTTP

HTTP não precisa se preocupar com dados perdidos ou com detalhes de como o TCP recupera a perda de dados ou os reordena dentro da rede. Essa é a tarefa do TCP e dos protocolos das camadas mais inferiores da pilha de protocolos.

É importante notar que o servidor envia ao cliente os arquivos solicitados sem armazenar nenhuma informação de estado sobre este. Se um determinado cliente solicita o mesmo objeto duas vezes em um período de poucos segundos, o servidor não responde dizendo que acabou de enviar o objeto ao cliente; em vez disso, envia novamente o objeto, pois já esqueceu completamente o que fez antes. Como o servidor HTTP não mantém nenhuma informação sobre clientes, o HTTP é denominado **um protocolo sem estado**.

Salientamos também que a Web usa a arquitetura de aplicação cliente-servidor, como descrito na Seção 2.1. Um servidor Web está sempre em funcionamento, tem um endereço IP fixo e atende requisições de potencialmente milhões de browsers diferentes.

2.2.2 Conexões persistentes e não persistentes

Em muitas aplicações da Internet, o cliente e o servidor se comunicam por um período prolongado de tempo, em que o cliente faz uma série de requisições e o servidor responde a cada uma delas. Dependendo da aplicação e de como ela está sendo usada, a série de requisições pode ser feita de forma consecutiva, periodicamente em intervalos regulares ou esporadicamente. Quando a interação cliente-servidor acontece por meio de conexão TCP, o criador da aplicação precisa tomar uma importante decisão — cada par de requisição/resposta deve ser enviado por uma conexão TCP *distinta* ou todas as requisições e suas respostas devem ser enviadas por uma *mesma* conexão TCP? Na abordagem anterior, a aplicação utiliza conexões não persistentes; e na última abordagem, conexões persistentes. Para obter uma maior compreensão deste assunto, vamos analisar as vantagens e desvantagens das conexões não persistentes e das conexões persistentes no contexto de uma aplicação específica, o HTTP, que pode utilizar essas duas conexões. Embora o HTTP utilize conexões persistentes em seu modo padrão, os clientes e servidores HTTP podem ser configurados para utilizar a conexão não persistente.

O HTTP com conexões não persistentes

Vamos percorrer as etapas da transferência de uma página Web de um servidor para um cliente para o caso de conexões não persistentes. Vamos supor que uma página consista em um arquivo-base HTML e em dez imagens JPEG e que todos esses 11 objetos residam no mesmo servidor. Suponha também que o URL para o arquivo-base HTTP seja

`http://www.someSchool.edu/someDepartment/home.index`

Eis o que acontece:

1. O processo cliente HTTP inicia uma conexão TCP para o servidor `www.someSchool.edu` na porta número 80, que é o número de porta default para o HTTP. Associados à conexão TCP, haverá um socket no cliente e um socket no servidor.
2. O cliente HTTP envia uma mensagem de requisição HTTP ao servidor através de seu socket. Essa mensagem inclui o nome de caminho `/someDepartment/home.index`. (Discutiremos mensagens HTTP mais detalhadamente logo adiante.)
3. O processo servidor HTTP recebe a mensagem de requisição através de seu socket, extrai o objeto `/someDepartment/home.index` de seu armazenamento (RAM ou disco), encapsula o objeto em uma mensagem de resposta HTTP e a envia ao cliente através de seu socket.
4. O processo servidor HTTP ordena ao TCP que encerre a conexão TCP. (Mas, na realidade, o TCP só encerrará a conexão quando tiver certeza de que o cliente recebeu a mensagem de resposta intacta.)
5. O cliente HTTP recebe a mensagem de resposta e a conexão TCP é encerrada. A mensagem indica que o objeto encapsulado é um arquivo HTML. O cliente extrai o arquivo da mensagem de resposta, analisa o arquivo HTML e encontra referências aos dez objetos JPEG.
6. As primeiras quatro etapas são repetidas para cada um dos objetos JPEG referenciados.

À medida que recebe a página Web, o browser a apresenta ao usuário. Dois browsers diferentes podem interpretar (isto é, exibir ao usuário) uma página Web de modos ligeiramente diferentes. O HTTP não tem nada a ver com o modo como uma página Web é interpretada por um cliente. As especificações do HTTP [RFC 1945] e [RFC 2616] definem apenas o protocolo de comunicação entre o programa cliente HTTP e o programa servidor HTTP.

As etapas apresentadas ilustram a utilização de conexões não persistentes, nas quais cada conexão TCP é encerrada após o servidor enviar o objeto — a conexão não persiste para outros objetos. Note que cada conexão TCP transporta exatamente uma mensagem de requisição e uma mensagem de resposta. Assim, nesse exemplo, quando um usuário solicita a página Web, são geradas 11 conexões TCP.

Nas etapas descritas, fomos intencionalmente vagos sobre se os clientes obtêm as dez JPEGs por meio de dez conexões TCP em série ou se algumas delas são recebidas por conexões TCP paralelas. Na verdade, usuários podem configurar browsers modernos para controlar o grau de paralelismo. Nos modos default, a maioria dos browsers abre de cinco a dez conexões TCP paralelas e cada uma delas manipula uma transação requisição/resposta. Se o usuário preferir, o número máximo de conexões paralelas poderá ser fixado em um, caso em que as dez conexões são estabelecidas em série. Como veremos no próximo capítulo, a utilização de conexões paralelas reduz o tempo de resposta.

Antes de continuarmos, vamos fazer um cálculo simples para estimar o tempo que transcorre entre a requisição e o recebimento de um arquivo-base HTTP por um cliente. Para essa finalidade, definimos o **tempo de viagem de ida e volta** (*round-trip time* — RTT), ou seja, o tempo que leva para um pequeno pacote viajar do cliente ao servidor e de volta ao cliente. O RTT inclui atrasos de propagação de pacotes, de fila de pacotes em roteadores e comutadores intermediários e de processamento de pacotes. (Esses atrasos foram discutidos na Seção 1.4.) Considere, agora, o que acontece quando um usuário clica sobre um hiperenlace. Como ilustrado na Figura 2.7, isso faz com que o browser inicie uma conexão TCP entre ele e o servidor Web, o que envolve uma ‘apresentação de três vias’ — o cliente envia um pequeno segmento TCP ao servidor, o servidor o reconhece e responde com um pequeno segmento ao cliente que, por fim, o reconhece novamente para o servidor. As duas primeiras partes da apresentação de três vias representam um RTT. Após concluir essas partes, o cliente envia a mensagem de requisição HTTP combinada com a terceira parte da apresentação de três vias (o reconhecimento) para dentro da conexão TCP. Assim que a mensagem de requisição chega ao servidor, este envia o arquivo HTML por meio

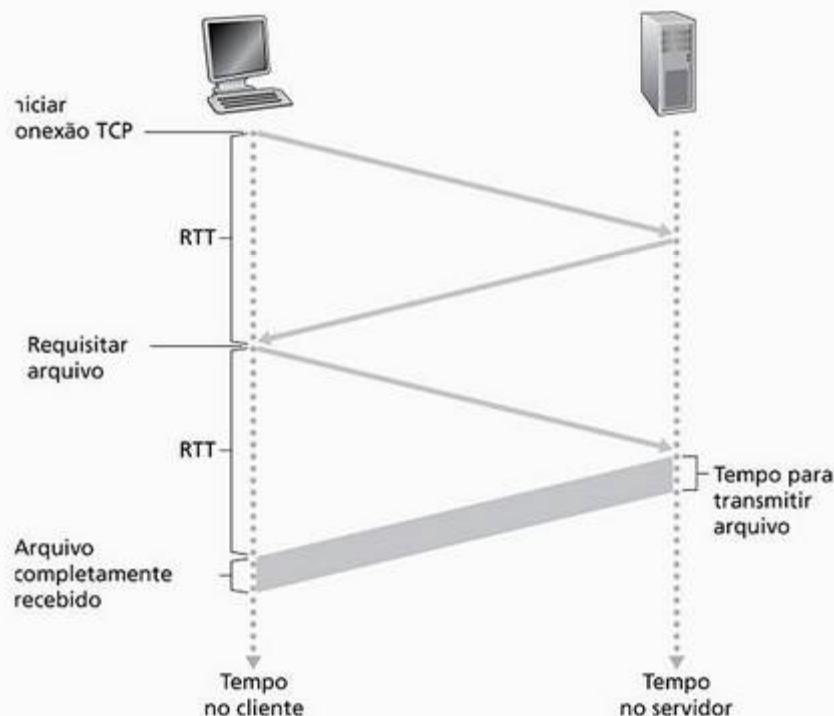


Figura 2.7 Cálculo simples para o tempo necessário para solicitar e receber um arquivo HTML

da conexão TCP. Essa requisição/resposta HTTP causa mais um RTT. Assim, aproximadamente, o tempo total de resposta são dois RTTs mais o tempo de transmissão do arquivo HTML no servidor.

O HTTP com conexões persistentes

Conexões não persistentes têm algumas desvantagens. Em primeiro lugar, uma nova conexão deve ser estabelecida e mantida para *cada objeto solicitado*. Para cada uma dessas conexões, devem ser alocados buffers TCP e conservadas variáveis TCP tanto no cliente quanto no servidor. Isso pode sobrecarregar seriamente o servidor Web, que poderá estar processando requisições de centenas de diferentes clientes ao mesmo tempo. Em segundo lugar, como acabamos de descrever, cada objeto sofre dois RTTs — um RTT para estabelecer a conexão TCP e um RTT para solicitar e receber um objeto.

Em conexões persistentes, o servidor deixa a conexão TCP aberta após enviar resposta. Requisições e respostas subsequentes entre os mesmos cliente e servidor podem ser enviadas por meio da mesma conexão. Em particular, uma página Web inteira (no exemplo anterior, o arquivo-base HTML e as dez imagens) pode ser enviada mediante uma única conexão TCP persistente. Essas requisições por objetos podem ser feitas consecutivamente sem ter de esperar por respostas a requisições pendentes (parallelismo). Normalmente, o servidor HTTP fecha uma conexão quando ela não é usada durante um certo tempo (um intervalo de pausa configurável). Quando o servidor recebe requisições consecutivas, os objetos são enviados de forma ininterrupta.

O modo default do HTTP usa conexões persistentes com paralelismo. Faremos uma comparação quantitativa entre os desempenhos de conexões persistentes e não persistentes nos exercícios de fixação dos capítulos 2 e 3. Aconselhamos o leitor interessado a consultar [Heidemann, 1997; Nielsen, 1997].

2.2.3 Formato da mensagem HTTP

As especificações do HTTP [RFC 2616] definem os formatos das mensagens HTTP. Há dois tipos de mensagens HTTP: de requisição e de resposta. Ambas serão discutidas a seguir.

Mensagem de requisição HTTP

Apresentamos a seguir uma mensagem de requisição HTTP típica:

```
GET /somedir/page.html HTTP/1.1
Host: www.someschool.edu
Connection: close
User-agent: Mozilla/4.0
Accept-language: fr
```

Podemos aprender bastante examinando essa simples mensagem de requisição. Em primeiro lugar, vemos que ela está escrita em texto ASCII comum, de modo que pode ser lida por qualquer um que conheça computadores. Em segundo lugar, vemos que ela é constituída de cinco linhas, cada uma seguida de um 'carriage return' e 'line feed' (fim de linha) para o início de uma nova linha. A última linha é seguida de um comando adicional de 'carriage return' e 'line feed'. Embora essa mensagem de requisição específica tenha cinco linhas, uma mensagem de requisição pode ter muito mais do que isso e também menos do que isso, podendo conter até mesmo uma única linha. A primeira linha de uma mensagem de requisição HTTP é denominada **linha de requisição**; as subsequentes são denominadas **linhas de cabeçalho**. A linha de requisição tem três campos: o campo do método, o do URL e o da versão do HTTP. O campo do método pode assumir vários valores diferentes, entre eles GET, POST e HEAD. A grande maioria das mensagens de requisição HTTP emprega o método GET, o qual é usado quando o browser requisita um objeto e este é identificado no campo do URL. Nesse exemplo, o browser está requisitando o objeto /somedir/page.html. A versão é autoexplicativa. Nesse exemplo, o browser implementa a versão HTTP/1.1.

Vamos agora examinar as linhas de cabeçalho do exemplo. A linha de cabeçalho `Host: www.some-school.edu` especifica o hospedeiro no qual o objeto reside. Talvez você ache que ela é desnecessária, pois já existe uma conexão TCP para o hospedeiro. Mas, como veremos na Seção 2.2.5, a informação fornecida pela linha de cabeçalho do hospedeiro é requerida por armazenadores intermediários da Web. Ao incluir a linha de cabeçalho `Connection: close`, o browser está dizendo ao servidor que não quer usar conexões persistentes; quer que o servidor feche a conexão após o envio do objeto requisitado. A linha de cabeçalho `User-agent:` especifica o agente de usuário, isto é, o tipo de browser que está fazendo a requisição ao servidor. No caso em questão, o agente de usuário é o Mozilla/4.0, um browser da Netscape. Essa linha de cabeçalho é útil porque, na verdade, o servidor pode enviar versões diferentes do mesmo objeto a tipos diferentes de agentes de usuário. (Cada uma das versões é endereçada pelo mesmo URL.) Por fim, o cabeçalho `Accept-language:` mostra que o usuário prefere receber uma versão em francês do objeto se esse objeto existir no servidor; se não existir, o servidor deve enviar a versão default. O cabeçalho `Accept-language:` é apenas um dos muitos cabeçalhos de negociação de conteúdo disponíveis no HTTP.

Acabamos de examinar um exemplo. Vamos agora analisar o formato geral de uma mensagem de requisição, ilustrado na Figura 2.8.

Vemos que o formato geral de uma mensagem de requisição é muito parecido com nosso exemplo anterior. Contudo, você provavelmente notou que, após as linhas de cabeçalho (e após a linha adicional com 'carriage return' e 'line feed'), há um 'corpo de entidade'. O corpo de entidade fica vazio com o método GET, mas é utilizado com o método POST. Um cliente HTTP normalmente usa o método POST quando o usuário preenche um formulário — por exemplo, quando fornece palavras de busca a um site buscador. Com uma mensagem POST, o usuário continua solicitando uma página Web ao servidor, mas o conteúdo específico dela depende do que o usuário escreveu nos campos do formulário. Se o valor do campo de método for POST, então o corpo de entidade conterá o que o usuário digitou nos campos do formulário.

Seríamos omissos se não mencionássemos que uma requisição gerada com um formulário não utiliza necessariamente o método POST. Ao contrário, formulários HTML frequentemente utilizam o método GET e incluem os dados digitados (nos campos do formulário) no URL requisitado. Por exemplo, se um formulário usar o método GET, tiver dois campos e as entradas desses dois campos forem `monkeys` e `bananas`, então a estrutura do URL será `www.somesite.com/animalsearch?monkeys&bananas`. Ao navegar normalmente pela Web, você provavelmente já notou URLs extensos como esse.

O método HEAD é semelhante ao GET. Quando um servidor recebe uma requisição com o método HEAD, responde com uma mensagem HTTP, mas deixa de fora o objeto requisitado. Esse método é frequentemente usado pelos desenvolvedores de servidores HTTP para depuração.

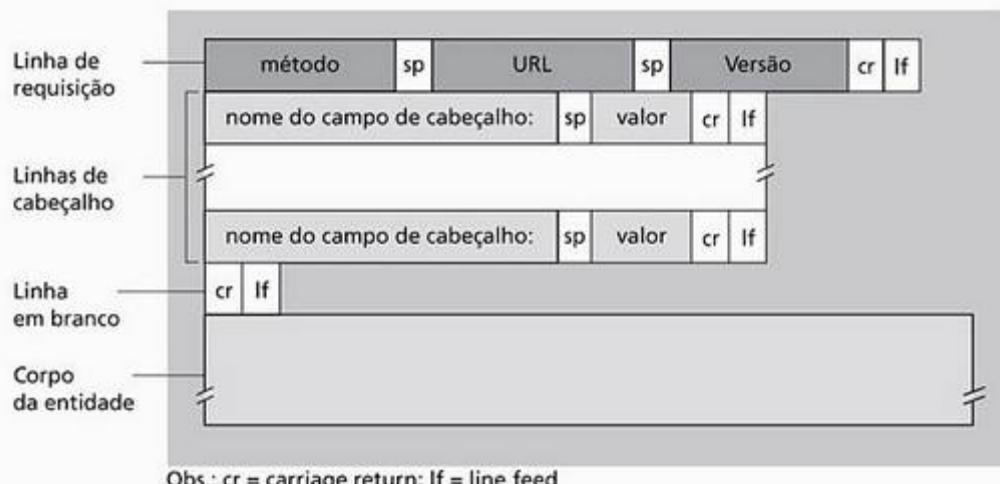


Figura 2.8 Formato geral de uma mensagem de requisição

O método PUT é frequentemente usado juntamente com ferramentas de edição da Web. Permite que um usuário carregue um objeto para um caminho específico (diretório) em um servidor Web específico. O método PUT também é usado por aplicações que precisam carregar objetos para servidores Web. O método DELETE permite que um usuário, ou uma aplicação, elimine um objeto em um servidor Web.

Mensagem de resposta HTTP

Apresentamos a seguir uma mensagem de resposta HTTP típica. Essa mensagem poderia ser a resposta ao exemplo de mensagem de requisição que acabamos de discutir.

```
HTTP/1.1 200 OK
Connection: close
Date: Sat, 07 Jul 2007 12:00:15 GMT
Server: Apache/1.3.0 (Unix)
Last-Modified: Sun, 6 May 2007 09:23:24 GMT
Content-Length: 6821
Content-Type: text/html
(data data data data data ...)
```

Vamos examinar cuidadosamente essa mensagem de resposta. Ela tem três seções: uma linha inicial ou **linha de estado**, seis linhas de cabeçalho e, em seguida, o **corpo da entidade**, que é o 'filé mignon' da mensagem — contém o objeto solicitado (representado por data data data data data ...). A linha de estado tem três campos: o campo de versão do protocolo, um código de estado e uma mensagem de estado correspondente. Nesse exemplo, ela mostra que o servidor está usando o HTTP/1.1 e que está tudo OK (isto é, o servidor encontrou e está enviando o objeto solicitado).

Agora, vamos examinar as linhas de cabeçalho. O servidor usa a linha de cabeçalho Connection: close para informar ao cliente que fechará a conexão TCP após enviar a mensagem. A linha de cabeçalho Date: indica a hora e a data em que a resposta HTTP foi criada e enviada pelo servidor. Note que esse não é o horário em que o objeto foi criado nem o de sua modificação mais recente; é a hora em que o servidor extraiu o objeto de seu sistema de arquivos, inseriu-o na mensagem de resposta e enviou-a. A linha de cabeçalho Server: mostra que a mensagem foi gerada por um servidor Web Apache, análoga à linha de cabeçalho User-agent: na mensagem de requisição HTTP. A linha de cabeçalho Last-Modified: indica a hora e a data em que o objeto foi criado ou sofreu a última modificação. Esse cabeçalho, que logo estudaremos mais detalhadamente, é fundamental para fazer cache do objeto, tanto no cliente local quanto em servidores de cache da rede (também conhecidos como servidores proxy). A linha de cabeçalho Content-Length: indica o número de bytes do objeto que está sendo enviado e a linha Content-Type: mostra que o objeto presente no corpo da mensagem é um texto HTML. (O tipo do objeto é oficialmente indicado pelo cabeçalho Content-Type: e não pela extensão do arquivo.)

Acabamos de ver um exemplo. Vamos agora examinar o formato geral de uma mensagem de resposta, ilustrado na Figura 2.9.

Esse formato geral de mensagem de resposta condiz com o exemplo anterior. Mas falemos um pouco mais sobre códigos de estado e suas frases, que indicam o resultado da requisição. Eis alguns códigos de estado e frases associadas comuns:

- 200 OK: requisição bem-sucedida e a informação é entregue com a resposta.
- 301 Moved Permanently: objeto requisitado foi removido permanentemente; novo URL é especificado no cabeçalho Location: da mensagem de resposta. O software do cliente recuperará automaticamente o novo URL.
- 400 Bad Request: código genérico de erro que indica que a requisição não pôde ser entendida pelo servidor.

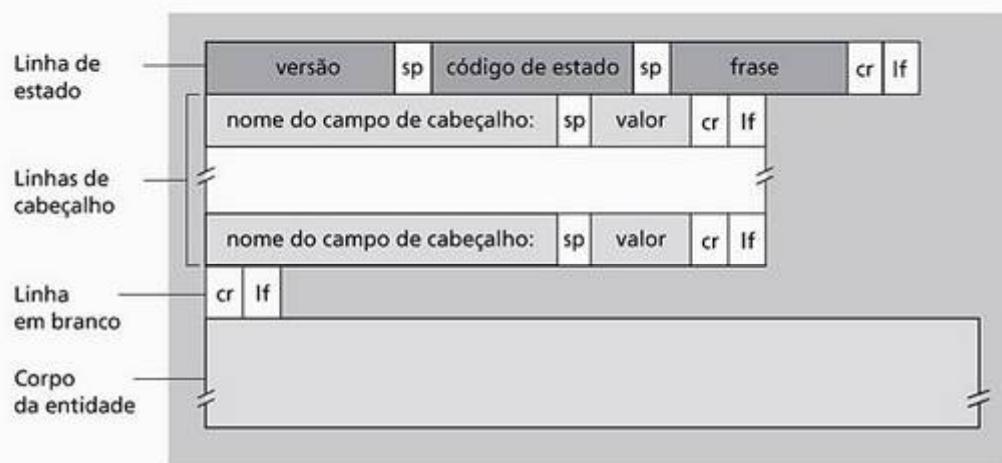


Figura 2.9 Formato geral de uma mensagem de resposta

404 Not Found: o documento requisitado não existe no servidor.

505 HTTP Version Not Supported: a versão do protocolo HTTP requisitada não é suportada pelo servidor.

Você gostaria de ver uma mensagem de resposta HTTP real? É muito recomendável e muito fácil! Primeiramente, dê um comando Telnet em seu servidor Web favorito. Em seguida, digite uma mensagem de requisição de uma linha solicitando algum objeto abrigado no servidor. Por exemplo, se você tem acesso a um prompt de comando, digite:

telnet cis.poly.edu 80

```

GET /~ross/ HTTP/1.1
Host: cis.poly.edu
  
```

(Clique duas vezes a tecla 'enter' após digitar a última linha.) Essa sequência de comandos abre uma conexão TCP para a porta número 80 do hospedeiro `cis.poly.edu` e, em seguida, envia a mensagem de requisição HTTP. Deverá aparecer uma mensagem de resposta que inclui o arquivo-base HTML da homepage do professor Ross. Se você preferir apenas ver as linhas da mensagem HTTP e não receber o objeto em si, substitua `GET` por `HEAD`. Finalmente, substitua `/~ross/` por `/~banana/` e veja tipo de mensagem você obtém.

Nesta seção, discutimos várias linhas de cabeçalho que podem ser usadas em mensagens de requisição e de resposta HTTP. A especificação do HTTP define muitas outras linhas de cabeçalho que podem ser inseridas por browsers, servidores Web e servidores cache de redes. Vimos apenas um pouco do total de linhas de cabeçalho. Examinaremos mais algumas a seguir e mais um pouco quando discutirmos armazenagem Web na Seção 2.2.5. Uma discussão muito abrangente e fácil de ler sobre o protocolo HTTP, seus cabeçalhos e códigos de estado, pode ser encontrada em [Krishnamurty, 2001]; veja também [Luotonen, 1998] se estiver interessado no ponto de vista do desenvolvedor.

Como um browser decide quais linhas de cabeçalho serão incluídas em uma mensagem de requisição? Como um servidor Web decide quais linhas de cabeçalho serão incluídas em uma mensagem de resposta? Um browser vai gerar linhas de cabeçalho em função de seu tipo e versão (por exemplo, um browser HTTP/1.0 não vai gerar nenhuma linha de cabeçalho 1.1), da configuração do usuário para o browser (por exemplo, idioma preferido) e se o browser tem uma versão do objeto possivelmente ultrapassada em sua memória. Servidores Web se comportam de maneira semelhante: há diferentes produtos, versões e configurações, e todos influenciam as linhas de cabeçalho que são incluídas nas mensagens de resposta.

2.2.4 Intereração usuário-servidor: cookies

Mencionamos anteriormente que um servidor HTTP não tem estado, o que simplifica o projeto do servidor e vem permitindo que engenheiros desenvolvam servidores Web de alto desempenho que podem manipular milhares de conexões TCP simultâneas. No entanto, é sempre bom que um site Web identifique usuários, seja porque o servidor deseja restringir acesso de usuário, seja porque quer apresentar conteúdo em função da identidade do usuário. Para essas finalidades, o HTTP usa cookies. Cookies, definidos no RFC 2965, permitem que sites monitorem seus usuários. Hoje, a maioria dos sites Web comerciais utiliza cookies.

Como ilustrado na Figura 2.10, a tecnologia dos cookies tem quatro componentes: (1) uma linha de cabeçalho de cookie na mensagem de resposta HTTP; (2) uma linha de cabeçalho de cookie na mensagem de requisição HTTP; (3) um arquivo de cookie mantido no sistema final do usuário e gerenciado pelo browser do usuário; (4) um banco de dados de apoio no site Web. Utilizando a Figura 2.10, vamos esmiuçar um exemplo de como os cookies são usados. Suponha que Susan, que sempre acessa a Web usando o Internet Explorer de seu PC, acesse o Amazon.com pela primeira vez, e que, no passado, ela já tenha visitado o site da eBay. Quando a requisição chega ao servidor Web da Amazon, ele cria um número de identificação exclusivo e uma entrada no seu banco de dados de apoio, que é indexado pelo número de identificação. Então, o servidor Web da Amazon responde ao browser de Susan, incluindo na resposta HTTP um cabeçalho Set-cookie: que contém o número de identificação. Por exemplo, a linha de cabeçalho poderia ser:

Set-cookie: 1678

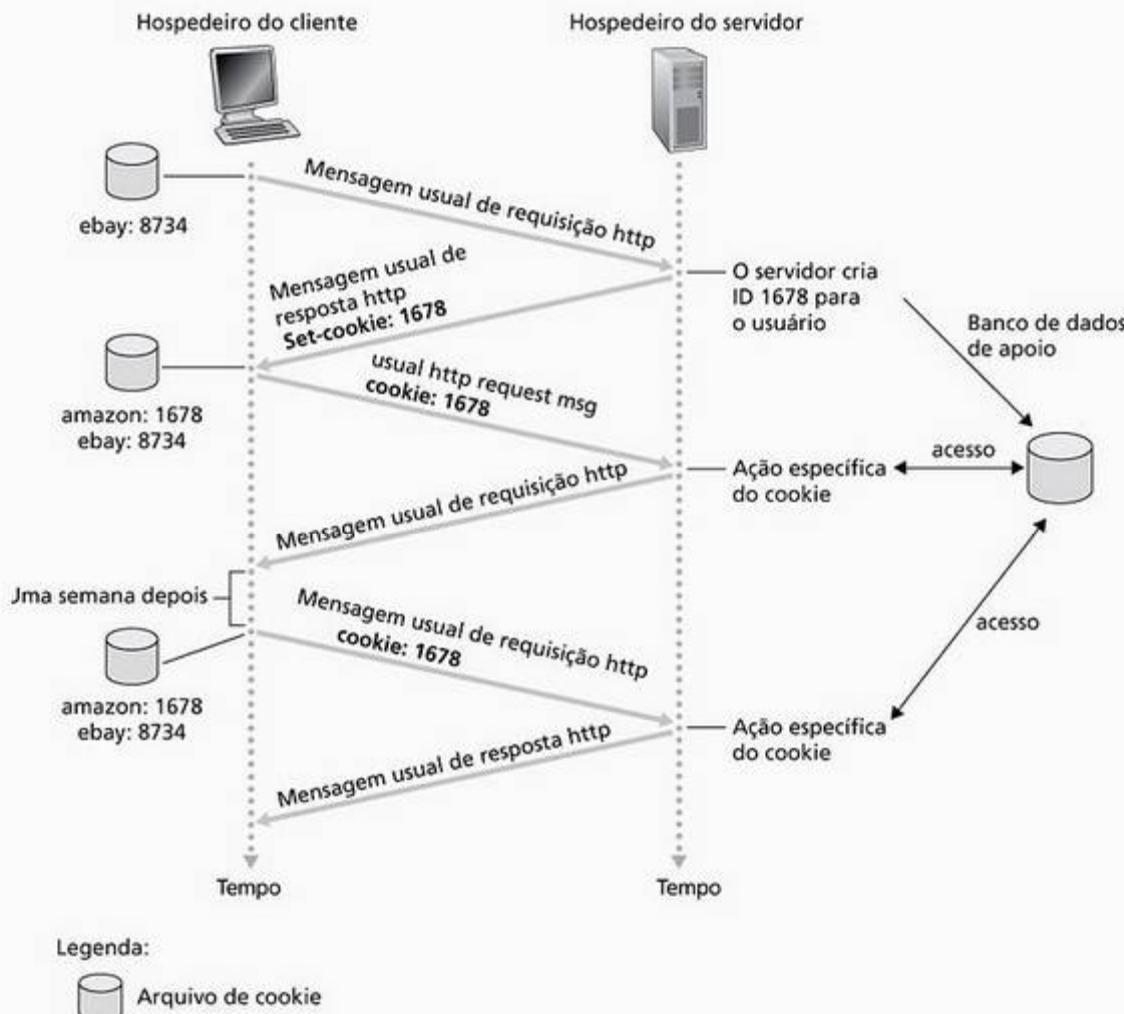


Figura 2.10 Mantendo o estado do usuário com cookies

Quando recebe a mensagem de resposta HTTP, o browser de Susan vê o cabeçalho Set-cookie: e, então, anexa uma linha ao arquivo especial de cookies que ele gerencia. Essa linha inclui o nome de hospedeiro do servidor e seu número de identificação nesse cabeçalho. Observe que o arquivo de cookie já possui um entrada para o eBay, uma vez que Susan já visitou esse site no passado. Toda vez que Susan requisita uma página Web enquanto navega pelo site da Amazon em questão, seu browser consulta seu arquivo de cookies, extrai seu número de identificação para esse site e insere na requisição HTTP uma linha de cabeçalho de cookie que inclui o número de identificação. Especificamente, cada uma de suas requisições HTTP ao servidor do site de comércio eletrônico inclui a linha de cabeçalho:

Cookie: 1678

Dessa maneira, o servidor da Amazon pode monitorar a atividade de Susan e, embora não saiba necessariamente que o nome dela é Susan, sabe exatamente quais páginas o usuário 1678 visitou, em que ordem e em que horários! Então, a Amazon pode utilizar cookies para oferecer um serviço de carrinho de compra — a Amazon pode manter uma lista de todas as compras de Susan, de modo que ela possa pagar por todas elas ao mesmo tempo, no final da sessão.

Se Susan voltar ao site, digamos, uma semana depois, seu browser continuará a inserir a linha de cabeçalho Cookie: 1678 nas mensagens de requisição. A Amazon pode recomendar produtos a Susan com base nas páginas Web que ela visitou na Amazon anteriormente. Se ela também se registrar no site — fornecendo seu nome completo, endereço de e-mail, endereço postal e informações de cartão de crédito — a Amazon pode incluir essas informações em seu banco de dados e, assim, associar o nome de Susan com seu número de identificação (e com todas as páginas que ela consultou em suas visitas anteriores). É assim que a Amazon e outros sites de comércio eletrônico oferecem “compras com um só clique” — quando quiser comprar um item durante uma visita subsequente, Susan não precisará digitar novamente seu nome, número de seu cartão de crédito, nem endereço.

Essa discussão nos mostrou que cookies podem ser usados para identificar um usuário. Quando visitar um site pela primeira vez, um usuário pode fornecer dados de identificação (possivelmente seu nome). No decorrer das sessões subsequentes, o browser passa um cabeçalho de cookie ao servidor durante todas as visitas subsequentes ao site, identificando, desse modo, o usuário ao servidor. A discussão também deixou claro que cookies podem ser usados para criar uma camada de sessão de usuário sobre HTTP sem estado. Por exemplo, quando um usuário acessa uma aplicação de e-mail baseada na Web (como o Hotmail), o browser envia informações de cookie ao servidor que, por sua vez, identifica o usuário por meio da sessão do usuário com a aplicação.

Embora cookies frequentemente simplifiquem a experiência de compra pela Internet, continuam provocando muita controvérsia porque também podem ser considerados como invasão da privacidade de um usuário. Como acabamos de ver, usando uma combinação de cookies e informações de conta fornecidas pelo usuário, um site Web pode ficar sabendo muita coisa sobre esse usuário e, potencialmente, vender o que sabe para algum terceiro. O Cookie Central [Cookie Central, 2008] inclui informações abrangentes sobre a controvérsia dos cookies.

2.2.5 Caches Web

Um **cache Web** — também denominado **servidor proxy** — é uma entidade da rede que atende requisições HTTP em nome de um servidor Web de origem. O cache Web tem seu próprio disco de armazenagem e mantém, dentro dele, cópias de objetos recentemente requisitados. Como ilustrado na Figura 2.11, o browser de um usuário pode ser configurado de modo que todas as suas requisições HTTP sejam dirigidas primeiramente ao cache Web. Uma vez que esteja configurado um browser, cada uma das requisições de um objeto que o browser faz é primeiramente dirigida ao cache Web. Como exemplo, suponha que um browser esteja requisitando o objeto <http://www.someschool.edu/campus.gif>. Eis o que acontece:

1. O browser estabelece uma conexão TCP com o cache Web e envia a ele uma requisição HTTP para um objeto.
2. O cache Web verifica se tem uma cópia do objeto armazenada localmente. Se tiver, envia o objeto ao browser do cliente, dentro de uma mensagem de resposta HTTP.



Figura 2.11 Clientes requisitando objetos por meio de um cache Web

3. Se não tiver o objeto, o cache Web abre uma conexão TCP com o servidor de origem, isto é, com `www.someschool.edu`. Então, envia uma requisição HTTP do objeto para a conexão TCP. Após receber essa requisição, o servidor de origem envia o objeto ao cache Web, dentro de uma resposta HTTP.
4. Quando recebe o objeto, o cache Web guarda uma cópia em seu armazenamento local e envia outra, dentro de uma mensagem de resposta, ao browser do cliente (pela conexão TCP existente entre o browser do cliente e o cache Web).

Note que um cache é, ao mesmo tempo, um servidor e um cliente. Quando recebe requisições de um browser e lhe envia respostas, é um servidor. Quando envia requisições para um servidor de origem e recebe respostas dele, é um cliente.

Em geral, é um ISP que compra e instala um cache Web. Por exemplo, uma universidade poderia instalar um cache na rede de seu *campus* e configurar todos os browsers apontando para esse cache. Ou um importante ISP residencial (como a AOL) poderia instalar um ou mais caches em sua rede e configurar antecipadamente os browsers que fornece apontando para os caches instalados.

O cache na Web tem sido utilizado amplamente na Internet por duas razões. Em primeiro lugar, um cache Web pode reduzir substancialmente o tempo de resposta para a requisição de um cliente, em particular se o gargalo da largura de banda entre o cliente e o servidor de origem for muito menor do que aquele entre o cliente e o cache. Se houver uma conexão de alta velocidade entre o cliente e o cache, como em geral é o caso, e se este tiver o objeto requisitado, então ele poderá entregar rapidamente o objeto ao cliente. Em segundo lugar, como logo ilustraremos com um exemplo, caches Web podem reduzir substancialmente o tráfego no enlace de acesso de uma instituição qualquer à Internet. Com a redução do tráfego, a instituição (por exemplo, uma empresa ou uma universidade) não precisa ampliar sua largura de banda tão rapidamente, o que diminui os custos. Além disso, caches Web podem reduzir substancialmente o tráfego na Internet como um todo, melhorando, assim, o desempenho para todas as aplicações.

Para entender melhor os benefícios dos caches, vamos considerar um exemplo no contexto da Figura 2.12. Essa figura mostra duas redes: uma rede institucional e a Internet pública. A rede institucional é uma LAN de alta velocidade. Um roteador da rede institucional e um roteador da Internet estão ligados por um enlace de 15 Mbps. Os servidores de origem estão todos ligados à Internet, mas localizados pelo mundo todo. Suponha que o tamanho médio do objeto seja 1 Mbits e que a taxa média de requisição dos browsers da instituição até os servidores de origem seja de 15 requisições por segundo. Imagine também que o tamanho das mensagens de requisição HTTP seja insignificante e, portanto, elas não criem tráfego nas redes ou no enlace de acesso (do roteador da instituição até o da Internet). Suponha ainda que o tempo entre o envio de uma requisição HTTP (dentro de um datagrama IP) pelo roteador do lado da Internet do enlace de acesso mostrado na Figura 2.12 e o recebimento da resposta (normalmente, dentro de muitos datagramas IPs) seja de 2 segundos em média. Esse último atraso é denominado informalmente ‘atraso da Internet’.

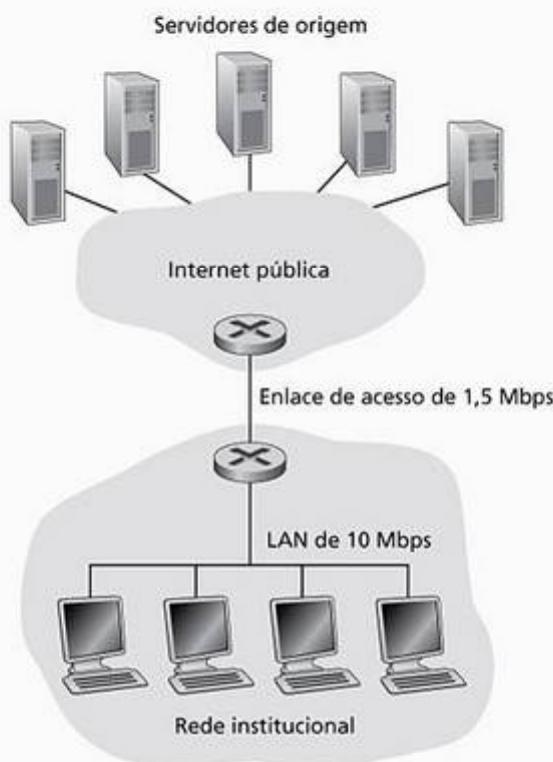


Figura 2.12 Gargalo entre uma rede institucional e a Internet

O tempo de resposta total — isto é, aquele transcorrido entre a requisição de um objeto feita pelo browser e o recebimento dele — é a soma do atraso da LAN, do atraso de acesso (isto é, o atraso entre os dois roteadores) e do atraso da Internet. Vamos fazer agora um cálculo bastante rudimentar para estimar esse atraso. A intensidade de tráfego na LAN (veja a Seção 1.4.2) é

$$(15 \text{ requisições/segundo}) \cdot (1 \text{ Mbits/requisição}) / (100 \text{ Mbps}) = 0,15$$

ao passo que a intensidade de tráfego no enlace de acesso (do roteador da Internet ao da instituição) é

$$(15 \text{ requisições/segundo}) = (1 \text{ Mbits/requisição}) / (15 \text{ Mbps}) = 1$$

Uma intensidade de tráfego de 0,15 em uma LAN resulta em, no máximo, dezenas de milissegundos de atraso; consequentemente, podemos desprezar o atraso da LAN. Contudo, como discutimos na Seção 1.4.2, à medida que a intensidade de tráfego se aproxima de 1 (como é o caso do enlace de acesso da Figura 2.12), o atraso em um enlace se torna muito grande e cresce sem limites. Assim, o tempo médio de resposta para atender requisições será da ordem de minutos, se não for maior, o que é inaceitável para os usuários da instituição. Evidentemente, algo precisa ser feito.

Uma possível solução seria aumentar a velocidade de acesso de 15 Mbps para, digamos, 100 Mbps. Isso reduziria a intensidade de tráfego no enlace de acesso a 0,15, o que se traduziria em atrasos desprezíveis entre os dois roteadores. Nesse caso, o tempo total de resposta seria aproximadamente 2 segundos, isto é, o atraso da Internet. Mas essa solução também significa que a instituição tem de atualizar seu enlace de acesso de 15 Mbps para 100 Mbps, o que pode ser muito dispendioso.

Considere agora a solução alternativa de não atualizar o enlace de acesso, mas, em vez disso, instalar um cache Web na rede institucional. Essa solução é ilustrada na Figura 2.13. A taxa de resposta local — a fração de requisições atendidas por um cache — em geral fica na faixa de 0,2 a 0,7 na prática. Para demonstrarmos isso, vamos supor que a taxa de resposta local do cache dessa instituição seja 0,4. Como os clientes e o cache estão conectados à mesma LAN de alta velocidade, 40 por cento das requisições serão atendidas quase

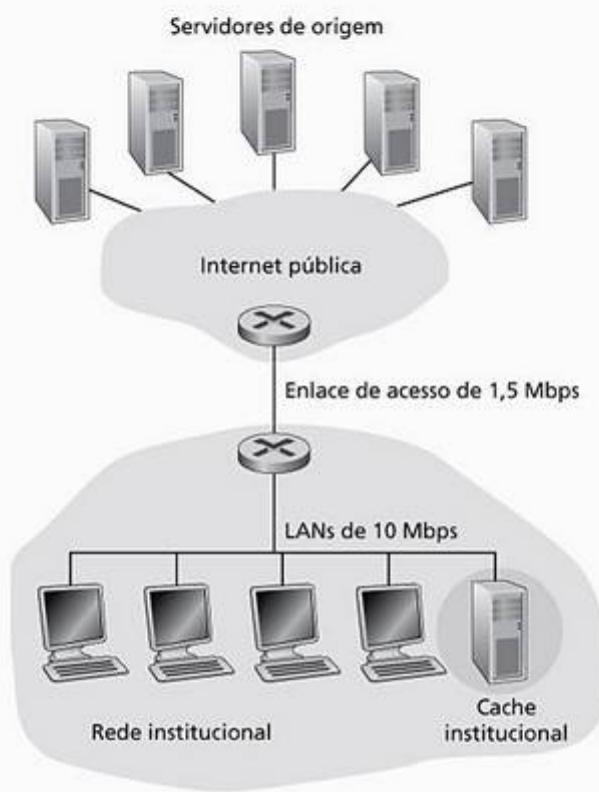


Figura 2.13 Adição de um cache à rede institucional

imediatamente, digamos, em 10 milissegundos. Mesmo assim, os demais 60 por cento das requisições ainda precisam ser atendidos pelos servidores de origem. Mas, com apenas 60 por cento dos objetos requisitados passando pelo enlace de acesso, a intensidade de tráfego neste diminui de 1,0 para 0,6. Em geral, uma intensidade de tráfego menor do que 0,8 corresponde a um atraso pequeno, digamos, dezenas de milissegundos, no caso de um enlace de 15 Mbps. Esse atraso é desprezível se comparado aos 2 segundos do atraso da Internet. Dadas essas considerações, o atraso médio é, por conseguinte, apenas ligeiramente maior do que 1,2 segundo.

$$0,4 \cdot (0,01 \text{ segundo}) + 0,6 \cdot (2,01 \text{ segundos}),$$

Assim, essa segunda solução resulta em tempo de resposta até menor do que o que se conseguiria com a troca do enlace de acesso e não requer que a instituição atualize seu enlace com a Internet. Evidentemente, a instituição terá de comprar e instalar um cache Web. Mas esse custo é baixo — muitos caches usam softwares de domínio público que rodam em PCs baratos.

2.2.6 GET condicional

Embora possa reduzir os tempos de resposta do ponto de vista do usuário, fazer cache introduz um novo problema — a cópia de um objeto existente no cache pode estar desatualizada. Em outras palavras, o objeto abrigado no servidor Web pode ter sido modificado desde a data em que a cópia entrou no cache no cliente. Felizmente, o HTTP tem um mecanismo que permite que um cache verifique se seus objetos estão atualizados. Esse mecanismo é denominado **GET condicional** (*conditional GET*). Uma mensagem de requisição HTTP é denominada uma mensagem GET condicional se (1) usar o método GET e (2) possuir uma linha de cabeçalho *If-Modified-Since*:

Para ilustrar como o GET condicional funciona, vamos examinar um exemplo. Em primeiro lugar, um cache proxy envia uma mensagem de requisição a um servidor Web em nome de um browser requisitante:

```
GET /fruit/kiwi.gif HTTP/1.1  
Host: www.exotiquecuisine.com
```

Em segundo lugar, o servidor Web envia ao cache uma mensagem de resposta com o objeto requisitado:

```
HTTP/1.1 200 OK  
Date: Sat, 7 Jul 2007 15:39:29  
Server: Apache/1.3.0 (Unix)  
Last-Modified: Wed, 4 Jul 2007 09:23:24  
Content-Type: image/gif  
  
(data data data data data ...)
```

O cache passa o objeto ao browser requisitante, mas também o guarda em sua memória cache local. O importante é que ele também guarda, juntamente com o objeto, a data da última modificação. Em terceiro lugar, uma semana depois, um outro browser requisita ao cache o mesmo objeto, que ainda está no cache. Como esse objeto pode ter sido modificado no servidor Web na semana anterior, o browser realiza uma verificação de atualização emitindo um GET condicional. Especificamente, o cache envia:

```
GET /fruit/kiwi.gif HTTP/1.1  
Host: www.exotiquecuisine.com  
If-modified-since: Wed, 4 Jul 2007 09:23:24
```

Note que o valor da linha de cabeçalho `If-modified-since:` é exatamente igual ao valor da linha de cabeçalho `Last-Modified:` que foi enviada pelo servidor há uma semana. Esse GET condicional está dizendo ao servidor para enviar o objeto somente se ele tiver sido modificado desde a data especificada. Suponha que o objeto não tenha sofrido modificação desde 4 Jul 2007 09:23:24. Então, em quarto lugar, o servidor Web envia uma mensagem de resposta ao cache:

```
HTTP/1.1 304 Not Modified  
Date: Sat, 14 Jul 2007 15:39:29  
Server: Apache/1.3.0 (Unix)  
  
(corpo de mensagem vazio)
```

Vemos que, em resposta ao GET condicional, o servidor Web ainda envia uma mensagem de resposta, mas não inclui nela o objeto requisitado, o que apenas desperdiçaria largura de banda e aumentaria o tempo de resposta do ponto de vista do usuário, particularmente se o objeto fosse grande. Note que, na linha de estado dessa última mensagem de resposta está inserido 304 Not Modified, que informa ao cache que ele pode seguir adiante e transmitir ao browser requisitante a cópia do objeto que está contida nele.

Finalizamos nossa discussão sobre HTTP, o primeiro protocolo da Internet (um protocolo da camada de aplicação) que estudamos em detalhes. Vimos o formato das mensagens HTTP e as ações tomadas pelo cliente e servidor Web quando essas mensagens são enviadas e recebidas. Também estudamos um pouco da infraestrutura da aplicação da Web, incluindo caches, cookies e banco de dados de apoio, todos associados, de algum modo, ao protocolo HTTP.

2.3 Transferência de arquivo: FTP

Em uma sessão FTP típica, o usuário, sentado à frente de um hospedeiro (o local), quer transferir arquivos de ou para um hospedeiro remoto. Para acessar a conta remota, o usuário deve fornecer uma identificação e uma senha. Após fornecer essas informações de autorização, pode transferir arquivos do sistema local de arquivos para o sistema remoto e vice-versa. Como mostra a Figura 2.14, o usuário interage com o FTP por meio de um agente de usuário FTP. Em primeiro lugar, ele fornece o nome do hospedeiro remoto, o que faz com que o processo cliente FTP do hospedeiro local estabeleça uma conexão TCP com o processo servidor FTP do hospedeiro remoto. O usuário então fornece sua identificação e senha, que são enviadas pela conexão TCP como parte dos



Figura 2.14 FTP transporta arquivos entre sistemas de arquivo local e remoto

comandos FTP. Assim que autorizado pelo servidor, o usuário copia um ou mais arquivos armazenados no sistema de arquivo local para o sistema de arquivo remoto (ou vice-versa).

O HTTP e o FTP são protocolos de transferência de arquivos e têm muitas características em comum; por exemplo, ambos utilizam o TCP. Contudo, esses dois protocolos de camada de aplicação têm algumas diferenças importantes. A mais notável é que o FTP usa duas conexões TCP paralelas para transferir um arquivo: uma **conexão de controle** e uma **conexão de dados**. A primeira é usada para enviar informações de controle entre os dois hospedeiros — como identificação de usuário, senha, comandos para trocar diretório remoto e comandos de ‘inserir’ e ‘pegar’ arquivos. A conexão de dados é a usada para efetivamente enviar um arquivo. Como o FTP usa uma conexão de controle separada, dizemos que ele envia suas informações de controle **fora da banda**. No Capítulo 7, veremos que o protocolo RTSP, usado para controlar a transferência de meios contínuos como áudio e vídeo, também envia suas informações de controle fora da banda. O HTTP, como você recorda, envia linhas de cabeçalho de requisição e de resposta pela mesma conexão TCP que carrega o próprio arquivo transferido. Por essa razão, dizemos que o HTTP envia suas informações de controle **na banda**. Na próxima seção, veremos que o SMTP, o principal protocolo para correio eletrônico, também envia suas informações de controle na banda. As conexões de controle e de dados do FTP estão ilustradas na Figura 2.15.

Quando um usuário inicia uma sessão FTP com um hospedeiro remoto, o lado cliente do FTP (usuário) inicia primeiramente uma conexão TCP de controle com o lado servidor (hospedeiro remoto) na porta número 21 do servidor e envia por essa conexão de controle a identificação e a senha do usuário, além de comandos para mudar o diretório remoto. Quando o lado servidor recebe, pela conexão de controle, um comando para uma transferência de arquivo (de ou para o hospedeiro remoto), abre uma conexão TCP de dados para o lado cliente. O FTP envia exatamente um arquivo pela conexão de dados e em seguida fecha-a. Se, durante a mesma sessão, o usuário quiser transferir outro arquivo, o FTP abrirá outra conexão de dados. Assim, com FTP, a conexão de controle permanece aberta durante toda a sessão do usuário, mas uma nova conexão de dados é criada para cada arquivo transferido dentro de uma sessão (ou seja, a conexão de dados é não persistente).

Durante uma sessão, o servidor FTP deve manter informações de **estado** sobre o usuário. Em particular, o servidor deve associar a conexão de controle com uma conta de usuário específica e também deve monitorar o

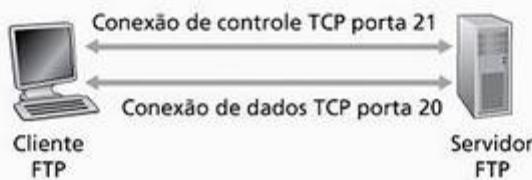


Figura 2.15 Conexões de controle e de dados

diretório corrente do usuário enquanto este passeia pela árvore do diretório remoto. Monitorar essas informações de estado para cada sessão de usuário em curso limita significativamente o número total de sessões que o FTP pode manter simultaneamente. Lembre-se de que o HTTP, por outro lado, é sem estado — não tem de monitorar o estado de nenhum usuário.

2.3.1 Comandos e respostas FTP

Encerraremos esta seção com uma breve discussão sobre alguns dos comandos mais comuns do FTP. Os comandos, do cliente para o servidor, e as respostas, do servidor para o cliente, são enviados por meio da conexão de controle no formato ASCII de 7 bits. Assim, tal como comandos HTTP, comandos FTP também podem ser lidos pelas pessoas. Para separar comandos sucessivos, um ‘carriage return’ e um ‘line feed’ encerram cada um deles. Cada comando é constituído de quatro caracteres ASCII maiúsculos, alguns com argumentos opcionais. Alguns dos comandos mais comuns são descritos a seguir:

- **USER username:** usado para enviar identificação do usuário ao servidor.
- **PASS password:** usado para enviar a senha do usuário ao servidor.
- **LIST:** usado para pedir ao servidor que envie uma lista com todos os arquivos existentes no atual diretório remoto. A lista de arquivos é enviada por meio de uma conexão de dados (nova e não persistente), e não pela conexão TCP de controle.
- **RETR filename:** usado para extrair (isto é, obter) um arquivo do diretório atual do hospedeiro remoto. Ativa o hospedeiro remoto para que abra uma conexão de dados e envie o arquivo requisitado por essa conexão.
- **STOR filename:** usado para armazenar (isto é, inserir) um arquivo no diretório atual do hospedeiro remoto.

Há, particularmente, uma correspondência unívoca entre o comando que o usuário gera e o comando FTP enviado pela conexão de controle. Cada comando é seguido de uma resposta, que é enviada do servidor ao cliente. As respostas são números de três dígitos com uma mensagem opcional após o número. Elas se assemelham, em estrutura, à codificação de estado e à frase da linha de estado da mensagem de resposta HTTP. Os inventores do HTTP incluíram intencionalmente essa similaridade nas mensagens de resposta HTTP. Algumas respostas típicas, junto com suas possíveis mensagens, são as seguintes:

- 331 Nome de usuário OK, senha requisitada
- 125 Conexão de dados já aberta; iniciando transferência
- 425 Não é possível abrir a conexão de dados
- 452 Erro ao escrever o arquivo

Para saber mais sobre outros comandos e respostas FTP, o leitor interessado pode consultar o RFC 959.

2.4 Correio eletrônico na Internet

O correio eletrônico existe desde o início da Internet. Era uma das aplicações mais populares quando ela ainda estava na infância [Segaller, 1998], e ficou mais e mais elaborado e poderoso ao longo dos anos. É uma das aplicações mais importantes e de maior uso da Internet.

Tal como o correio normal, o e-mail é um meio de comunicação assíncrono — as pessoas enviam e recebem mensagens quando for conveniente para elas, sem ter de estar coordenadas com o horário das outras pessoas. Ao contrário do correio normal, que anda a passos lentos, o correio eletrônico é rápido, fácil de distribuir e barato. O correio eletrônico moderno tem muitas características poderosas. Utilizando listas de mala direta, é possível enviar mensagens de e-mail, desejadas e indesejadas, a milhares de destinatários ao mesmo tempo. As mensagens do correio eletrônico moderno muitas vezes incluem anexos, hiperlinks, textos formatados em HTML e fotos.

Nesta seção, examinaremos os protocolos de camada de aplicação que estão no coração do correio eletrônico da Internet. Mas, antes de entrarmos nessa discussão, vamos tomar uma visão geral do sistema de correio da Internet e de seus componentes principais.

A Figura 2.16 apresenta uma visão do sistema de correio da Internet utilizando uma analogia com a correspondência por correio. Vemos, por esse diagrama, que há três componentes principais: **agentes de usuário**, **servidores de correio** e o **SMTP**. Descreveremos agora cada um desses componentes partindo do seguinte contexto: um remetente, Alice, está enviando uma mensagem de e-mail para um destinatário, Bob. Agentes de usuários permitem que usuários leiam, respondam, retransmitam, salvem e componham mensagens. (Às vezes, agentes de usuário de correio eletrônico são denominados *leitores de correio*, mas, neste livro, evitaremos essa expressão.) Quando Alice termina de compor sua mensagem, seu agente de usuário a envia a seu servidor de correio, onde ela é colocada na fila de saída de mensagens desse servidor. Quando Bob quer ler uma mensagem, seu agente de usuário a extrai da caixa de correio em seu servidor de correio. No final da década de 1990, agentes de usuário com interfaces gráficas de usuário (GUI) se tornaram populares, pois permitiam que usuários vissem e compusessem mensagens multimídia. Atualmente, o Outlook da Microsoft, o Apple Mail e o Mozilla Thunderbird são alguns dos agentes de usuário com interface gráfica populares para e-mail. Também há muitos agentes de usuário de texto de domínio público, (entre eles 'mail', 'pine' e 'elm'), assim como interfaces baseadas na Web, como veremos rapidamente.

Servidores de correio formam o núcleo da infraestrutura do e-mail. Cada destinatário, como Bob, tem uma **caixa postal** localizada em um dos servidores de correio. A de Bob administra e guarda as mensagens que foram enviadas a ele. Uma mensagem típica inicia sua jornada no agente de usuário do remetente, vai até o servidor de correio dele e viaja até o servidor de correio do destinatário, onde é depositada na caixa postal. Quando Bob quer acessar as mensagens de sua caixa postal, o servidor de correio que contém sua caixa postal o autentica (com nome de usuário e senha). O servidor de correio de Alice também deve cuidar das falhas no servidor de

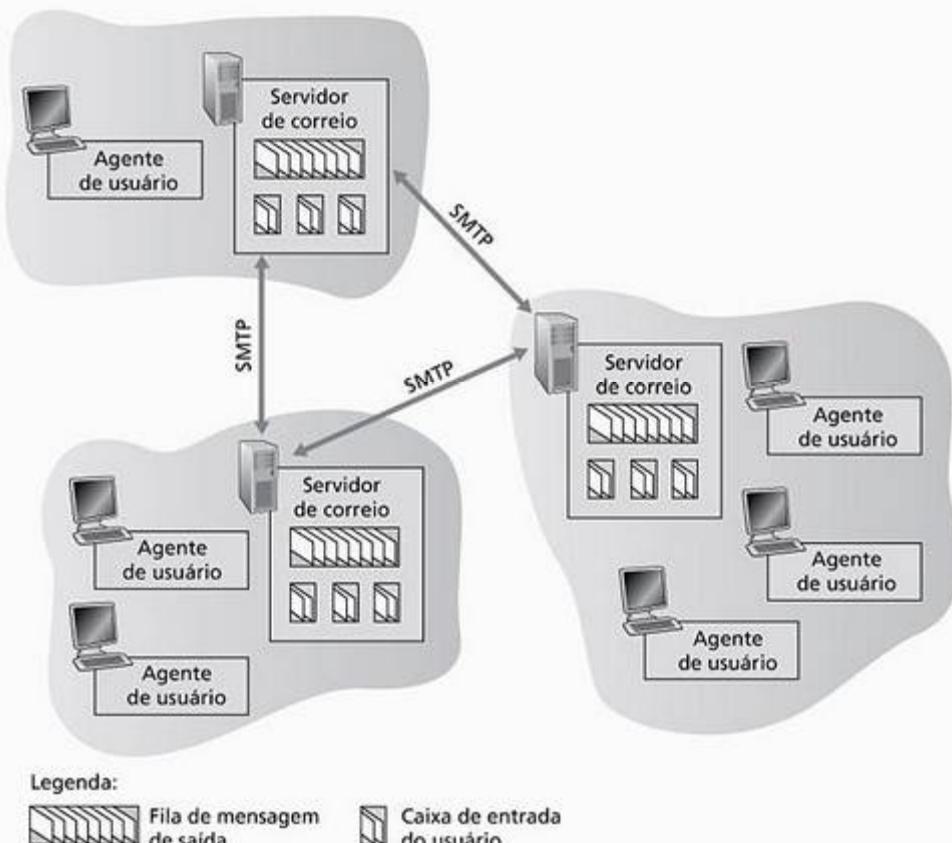


Figura 2.16 Uma visão do sistema de e-mail da Internet em analogia com a correspondência por correio



História

E-mail da Web

Em dezembro de 1995, após alguns anos depois da “invenção” da Web, Sabeer Bhatia e Jack Smith fizeram uma visita a Draper Fisher Jurvetson, um investidor em empreendimentos de Internet, e propuseram desenvolver um sistema de e-mail de livre acesso baseado na Web. A ideia era oferecer uma conta de e-mail grátis a quem quisesse e tornar essas contas acessíveis pela Web. Em troca de 15 por cento da empresa, Draper Fisher Jurvetson financiou Bhatia e Smith, que formaram uma empresa denominada Hotmail. Com três funcionários em tempo integral e mais 12 a 14 em tempo parcial, que trabalhavam em troca de opções de compra de ações da empresa, eles conseguiram desenvolver e lançar o serviço em julho de 1996. Um mês após o lançamento, a Hotmail tinha cem mil assinantes. Em dezembro de 1997, menos de 18 meses após seu lançamento, a Hotmail, já com mais de 12 milhões de assinantes, foi adquirida pela Microsoft, ao que se saiba, por 400 milhões de dólares.

O sucesso da Hotmail é muitas vezes atribuído à vantagem de ela ter sido a primeira a entrar no mercado e ao inerente ‘marketing viral’ do e-mail. (Talvez alguns dos estudantes que estão lendo este livro estarão entre os novos empreendedores que conceberão e desenvolverão serviços de Internet pioneiros no mercado e com marketing viral).

O e-mail da Web continua a prosperar, tornando-se, a cada ano, mais sofisticado e potente. Um dos serviços mais populares de hoje é o gmail do Google, que oferece livre armazenagem de gigabytes, filtro de spam e detector de vírus avançados, codificação de email opcional (utilizando SSL), coletor de e-mails e uma interface orientada a busca.

correio de Bob. Se o servidor de correio dela não puder entregar a correspondência ao servidor dele, manterá a mensagem em uma **fila de mensagens** e tentará transferi-la mais tarde. Em geral, novas tentativas serão feitas a cada 30 minutos aproximadamente; se não obtiver sucesso após alguns dias, o servidor removerá a mensagem e notificará o remetente por meio de uma mensagem de correio. O SMTP é o principal protocolo de camada de aplicação do correio eletrônico da Internet. Usa o serviço confiável de transferência de dados do TCP para transferir mensagens do servidor de correio do remetente para o do destinatário. Como acontece com a maioria dos protocolos de camada de aplicação, o SMTP tem dois lados: um lado cliente, que funciona no servidor de correio do remetente, e um lado servidor, que funciona no servidor de correio do destinatário. Ambos, o lado cliente e o lado servidor do SMTP, funcionam em todos os servidores de correio. Quando um servidor de correio envia correspondência para outros, age como um cliente SMTP. Quando o servidor de correio recebe correspondência de outros, age como um servidor SMTP.

2.4.1 SMTP

O SMTP, definido no RFC 5321, está no coração do correio eletrônico da Internet. Como mencionamos antes, esse protocolo transfere mensagens de servidores de correio remetentes para servidores de correio destinatários. O SMTP é muito mais antigo que o HTTP. (O RFC original do SMTP data de 1982, e ele já existia muito antes disso.) Embora tenha inúmeras qualidades maravilhosas, como evidencia sua ubiquidade na Internet, o SMTP é uma tecnologia antiga que possui certas características arcaicas. Por exemplo, restringe o corpo (e não apenas o cabeçalho) de todas as mensagens de correio ao simples formato ASCII de 7 bits. Essa restrição tinha sentido no começo da década de 1980, quando a capacidade de transmissão era escassa e ninguém enviava correio eletrônico com anexos volumosos nem arquivos grandes com imagens, áudio ou vídeo. Mas, hoje em dia, na era da multimídia, a restrição do ASCII de 7 bits é um tanto incômoda — exige que os dados binários de multimídia sejam codificados em ASCII antes de ser enviados pelo SMTP e que a mensagem correspondente em ASCII seja decodificada novamente para o sistema binário depois do transporte pelo SMTP. Lembre-se da Seção 2.2, na qual dissemos que o HTTP não exige que os dados de multimídia sejam codificados em ASCII antes da transferência.

Para ilustrar essa operação básica do SMTP, vamos percorrer um cenário comum. Suponha que Alice queira enviar a Bob uma simples mensagem ASCII:

1. Alice chama seu agente de usuário para e-mail, fornece o endereço de Bob (por exemplo, bob@someschool.edu), compõe uma mensagem e instrui o agente de usuário a enviar a mensagem.
2. O agente de usuário de Alice envia a mensagem para seu servidor de correio, onde ela é colocada em uma fila de mensagens.
3. O lado cliente do SMTP, que funciona no servidor de correio de Alice, vê a mensagem na fila e abre uma conexão TCP para um servidor SMTP, que funciona no servidor de correio de Bob.
4. Após alguns procedimentos iniciais de apresentação, o cliente SMTP envia a mensagem de Alice para dentro da conexão TCP.
5. No servidor de correio de Bob, o lado servidor do SMTP recebe a mensagem e a coloca na caixa postal dele.
6. Bob chama seu agente de usuário para ler a mensagem quando for mais conveniente para ele.

Esse cenário está resumido na Figura 2.17.

É importante observar que o SMTP normalmente não usa servidores de correio intermediários para enviar correspondência, mesmo quando os dois servidores estão localizados em lados opostos do mundo. Se o servidor de Alice está em Hong Kong, e o de Bob, em St. Louis, a conexão TCP é uma conexão direta entre os servidores em Hong Kong e St. Louis. Em particular, se o servidor de correio de Bob não estiver em funcionamento, a mensagem permanece no servidor de correio de Alice esperando por uma nova tentativa — a mensagem não é colocada em nenhum servidor de correio intermediário.

Vamos agora examinar mais de perto como o SMTP transfere uma mensagem de um servidor de correio remetente para um servidor de correio destinatário. Veremos que o protocolo SMTP tem muitas semelhanças com protocolos usados na interação humana cara a cara. Primeiramente, o cliente SMTP (que funciona no hospedeiro do servidor de correio remetente) faz com que o TCP estabeleça uma conexão na porta 25 com o servidor SMTP (que funciona no hospedeiro do servidor de correio destinatário). Se o servidor não estiver em funcionamento, o cliente tenta novamente mais tarde. Uma vez estabelecida a conexão, o servidor e o cliente trocam alguns procedimentos de apresentação de camada de aplicação — exatamente como os seres humanos, que frequentemente se apresentam antes de transferir informações, clientes e servidores SMTP também se apresentam antes de transferir informações. Durante essa fase, o cliente SMTP indica os endereços de e-mail do remetente (a pessoa que gerou a mensagem) e do destinatário. Assim que o cliente e o servidor SMTP terminam de se apresentar, o cliente envia a mensagem. O SMTP pode contar com o serviço confiável de transferência de dados do TCP para entregar a mensagem ao servidor sem erros. Então, o cliente repetirá esse processo, na mesma conexão TCP, se houver outras mensagens a enviar ao servidor; caso contrário, dará uma instrução ao TCP para encerrar a conexão.

Vamos examinar um exemplo de troca de mensagens entre um cliente (C) e um servidor SMTP (S). O nome do hospedeiro do cliente é crepes.fr e o nome do hospedeiro do servidor é hamburger.edu. As linhas de

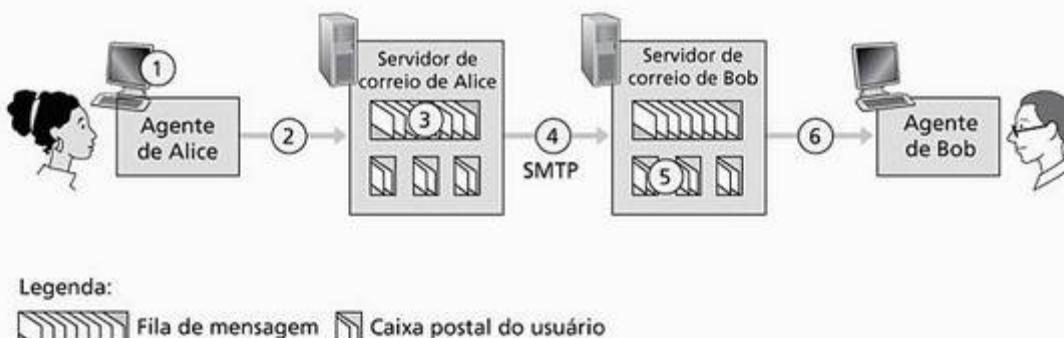


Figura 2.17 Alice envia uma mensagem a Bob

texto ASCII iniciadas com C: são exatamente as linhas que o cliente envia para dentro de seu socket TCP e as iniciadas com S: são exatamente as linhas que o servidor envia para dentro de seu socket TCP. A transcrição a seguir começa assim que a conexão TCP é estabelecida:

```
S: 220 hamburger.edu
C: HELO crepes.fr
S: 250 Hello crepes.fr, pleased to meet you
C: MAIL FROM: <alice@crepes.fr>
S: 250 alice@crepes.fr ... Sender ok
C: RCPT TO: <bob@hamburger.edu<
S: 250 bob@hamburger.edu ... Recipient ok
C: DATA
S: 354 Enter mail, end with "." on a line by itself
C: Do you like ketchup?
C: How about pickles?
C: .
S: 250 Message accepted for delivery
C: QUIT
S: 221 hamburger.edu closing connection
```

Nesse exemplo, o cliente enviou uma mensagem ("Do you like ketchup? How about pickles?") do servidor de correio crepes.fr ao servidor de correio hamburger.edu. Como parte do diálogo, o cliente emitiu cinco comandos: HELO (uma abreviação de HELLO), MAIL FROM, RCPT TO, DATA e QUIT. Esses comandos são autoexplicativos. O cliente também enviou uma linha consistindo em um único ponto final, que indica o final da mensagem para o servidor. (No jargão ASCII, cada mensagem termina com CRLF.CRLF, onde CR significa 'carriage return' e LF significa 'line feed'). O servidor emite respostas a cada comando, e cada resposta tem uma codificação de resposta e algumas explicações (opcionais) em inglês. Mencionamos aqui que o SMTP usa conexões persistentes: se o servidor de correio remetente tiver diversas mensagens para enviar ao mesmo servidor de correio destinatário, poderá enviar todas pela mesma conexão TCP. Para cada mensagem, o cliente inicia o processo com um novo MAIL FROM: crepes.fr, indica o final da mensagem com um ponto final isolado e emite QUIT somente após todas as mensagens terem sido enviadas.

Recomendamos veementemente que você utilize o Telnet para executar um diálogo direto com um servidor SMTP. Para fazer isso digite

```
telnet serverName 25
```

em que serverName é o nome de um servidor de correio local. Ao fazer isso, você está simplesmente estabelecendo uma conexão TCP entre seu hospedeiro local e o servidor de correio. Após digitar essa linha, você deverá receber imediatamente do servidor a resposta 220. Digite, então, os comandos HELO, MAIL FROM, RCPT TO, DATA, CRLF.CRLF e QUIT nos momentos apropriados. Também recomendamos veementemente que você faça a Tarefa de Programação 2 no final deste capítulo. Nessa tarefa, você construirá um agente de usuário simples que implementa o lado cliente do SMTP. Esse agente permitirá que você envie uma mensagem de e-mail a um destinatário qualquer, por meio de um servidor de correio local.

2.4.2 Comparação com o HTTP

Agora, vamos fazer uma breve comparação entre o SMTP e o HTTP. Ambos os protocolos são usados para transferir arquivos de um hospedeiro para outro. O HTTP transfere arquivos (também denominados objetos) de um servidor Web para um cliente Web (normalmente um browser). O SMTP transfere arquivos (isto é, mensagens de e-mail) de um servidor de correio para outro. Ao transferir os arquivos, o HTTP persistente e o SMTP usam conexões persistentes. Assim, os dois protocolos têm características em comum. Existem, todavia, diferenças importantes. A primeira é que o HTTP é, principalmente, um **protocolo de recuperação de informações (pull)**

protocol) — alguém carrega informações em um servidor Web e os usuários utilizam o HTTP para recuperá-las do servidor quando quiserem. Em particular, a conexão TCP é ativada pela máquina que quer receber o arquivo. O SMTP, por sua vez, é, primordialmente, um **protocolo de envio de informações** (*push protocol*) — o servidor de correio remetente envia o arquivo para o servidor de correio destinatário. Em particular, a conexão TCP é ativada pela máquina que quer enviar o arquivo.

A segunda diferença, à qual aludimos anteriormente, é que o SMTP exige que cada mensagem, inclusive o corpo, esteja no formato ASCII de 7 bits. Se a mensagem contiver caracteres que não estejam nesse formato (por exemplo, caracteres em francês, com acento) ou dados binários (como um arquivo de imagem), terá de ser codificada em ASCII de 7 bits. Dados HTTP não impõem esta restrição.

A terceira diferença importante refere-se ao modo como um documento que contém texto e imagem (juntamente com outros tipos possíveis de mídia) é manipulado. Como vimos na Seção 2.2, o HTTP encapsula cada objeto em sua própria mensagem HTTP. O correio pela Internet, como discutiremos com maiores detalhes mais adiante, coloca todos os objetos de mensagem em um única mensagem.

2.4.3 Formatos de mensagem de correio e MIME

Quando Alice escreve uma carta a Bob e a envia pelo correio normal, ela pode incluir todos os tipos de informações periféricas no cabeçalho da carta, como seu próprio endereço, o endereço de Bob e a data. De modo semelhante, quando uma mensagem de e-mail é enviada, um cabeçalho contendo informações periféricas antecede o corpo da mensagem em si. Essas informações periféricas estão contidas em uma série de linhas de cabeçalho definidas no RFC 5322. As linhas de cabeçalho e o corpo da mensagem são separados por uma linha em branco (isto é, por CRLF). O RFC 5322 especifica o formato exato das linhas de cabeçalho das mensagens, bem como suas interpretações semânticas. Como acontece com o HTTP, cada linha de cabeçalho contém um texto legível, consistindo em uma palavra-chave seguida de dois-pontos e de um valor. Algumas palavras-chave são obrigatórias e outras, opcionais. Cada cabeçalho deve ter uma linha de cabeçalho *From:* e uma *To:* e pode incluir também uma *Subject:* bem como outras opcionais. É importante notar que essas linhas de cabeçalho são diferentes dos comandos SMTP que estudamos na Seção 2.4.1 (ainda que contenham algumas palavras em comum, como 'from' e 'to'). Os comandos daquela seção faziam parte do protocolo de apresentação SMTP; as linhas de cabeçalho examinadas nesta seção fazem parte da própria mensagem de correio.

Um cabeçalho de mensagem típico é semelhante a:

```
From: alice@crepes.fr
To: bob@hamburger.edu
Subject: Searching for the meaning of life.
```

Após o cabeçalho da mensagem, vem uma linha em branco e, em seguida, o corpo da mensagem (em ASCII). Você pode usar o Telnet para enviar a um servidor de correio uma mensagem que contenha algumas linhas de cabeçalho, inclusive *Subject:*. Para tal, utilize o comando `telnet serverName 25`, como discutido na Seção 2.4.1.

2.4.4 Protocolos de acesso ao correio

Quando o SMTP entrega a mensagem do servidor de correio de Alice ao servidor de correio de Bob, ela é colocada na caixa postal de Bob. Durante toda essa discussão, ficou tacitamente subentendido que Bob lê sua correspondência ao entrar no hospedeiro servidor e, em seguida, executa o leitor de correio que roda naquela máquina. Até o início da década de 1990, este era o modo padronizado de acessar o correio. Mas hoje o acesso ao correio usa uma arquitetura cliente-servidor — o usuário típico lê e-mails com um cliente que funciona em seu sistema final, por exemplo, um PC no escritório, um laptop ou um PDA. Quando executam um cliente de correio em PC local, usuários desfrutam de uma série de propriedades, inclusive a capacidade de ver mensagens e anexos multimídia.

Dado que Bob (o destinatário) executa seu agente de usuário em seu PC local, é natural que ele considere a instalação de um servidor de correio nessa máquina. Adotando essa abordagem, o servidor de correio de Alice

dialogaria diretamente com o PC de Bob. Porém, há um problema com essa abordagem. Lembre-se de que um servidor de correio gerencia caixas postais e executa os lados cliente e servidor do SMTP. Se o servidor de correio de Bob residisse em seu PC local, este teria de ficar sempre em funcionamento e ligado na Internet para poder receber novas correspondências que poderiam chegar a qualquer hora, o que não é prático para muitos usuários da Internet. Em vez disso, um usuário típico executa um agente de usuário no PC local, mas acessa sua caixa postal armazenada em um servidor de correio compartilhado que está sempre em funcionamento. Esse servidor de correio é compartilhado com outros usuários e, em geral, é mantido pelo ISP do usuário (por exemplo, universidade ou empresa).

Agora, vamos considerar o caminho que uma mensagem percorre quando é enviada de Alice para Bob. Acabamos de aprender que, em algum ponto ao longo do caminho, a mensagem de e-mail precisa ser depositada no servidor de correio de Bob. Essa tarefa poderia ser realizada simplesmente fazendo com que o agente de usuário de Alice enviasse a mensagem diretamente ao servidor de correio de Bob, o que pode ser feito com o SMTP — realmente, o SMTP foi projetado para enviar e-mail de um hospedeiro para outro. Contudo, normalmente o agente de usuário do remetente não dialoga diretamente com o servidor de correio do destinatário. Em vez disso, como mostra a Figura 2.18, o agente de usuário de Alice usa SMTP para enviar a mensagem de e-mail a seu servidor de correio. Em seguida, esse servidor usa SMTP (como um cliente SMTP) para retransmitir a mensagem de e-mail para o servidor de correio de Bob. Por que esse procedimento em duas etapas? Primordialmente porque, sem a retransmissão através do servidor de correio de Alice, o agente de usuário dela não dispõe de nenhum recurso para um servidor de correio de destinatário que não pode ser alcançado. Fazendo com que Alice primeiramente deposite o e-mail em seu próprio servidor de correio, este pode tentar, várias vezes, enviar a mensagem ao servidor de correio de Bob, digamos, a cada 30 minutos, até que esse servidor entre em operação. (E, se o servidor de correio de Alice não estiver funcionando, ela terá o recurso de se queixar ao administrador do seu sistema!) O RFC do SMTP define como os comandos do SMTP podem ser usados para retransmitir uma mensagem por vários servidores SMTP.

Mas ainda falta uma peça do quebra-cabeça! De que forma um destinatário como Bob, que executa um agente de usuário em seu PC local, obtém suas mensagens que estão em um servidor de correio dentro do seu ISP? Note que o agente de usuário de Bob não pode usar SMTP para obter as mensagens porque essa operação é de recuperação (*pull*), e o SMTP é um protocolo de envio (*push*). O quebra-cabeça é concluído com a introdução de um protocolo especial de acesso ao correio que transfere mensagens do servidor de correio de Bob para seu PC local. Atualmente, há vários protocolos populares de acesso a correio, entre eles **POP3** (Post Office Protocol versão 3), **IMAP** (Internet Mail Access Protocol) e **HTTP**.

A Figura 2.18 apresenta um resumo dos protocolos usados no correio da Internet: o SMTP é utilizado para transferir correspondência do servidor de correio remetente para o servidor de correio destinatário; também é usado para transferir correspondência do agente de usuário remetente para o servidor de correio remetente. Um protocolo de acesso de correio, como o POP3, é usado para transferir correspondência do servidor de correio destinatário para o agente de usuário destinatário.

POP3

O POP3 é um protocolo de acesso de correio extremamente simples. É definido no RFC 1939, que é curto e bem fácil de ler. Por esse protocolo ser tão simples, sua funcionalidade é bastante limitada. O POP3 começa

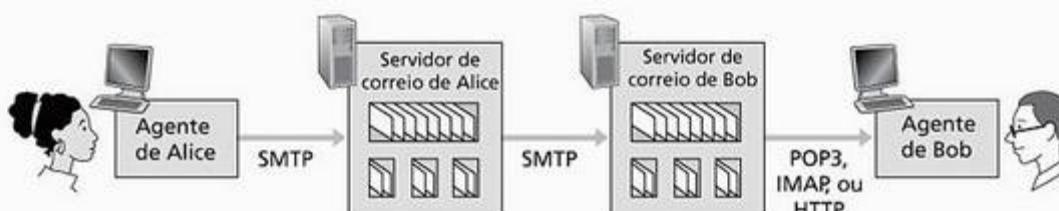


Figura 2.18 Protocolos de e-mail e suas entidades comunicantes

quando o agente de usuário (o cliente) abre uma conexão TCP com o servidor de correio (o servidor) na porta 110. Com a conexão TCP ativada, o protocolo passa por três fases: autorização, transação e atualização. Durante a primeira fase, autorização, o agente de usuário envia um nome de usuário e uma senha (às claras) para autenticar o usuário. Na segunda fase, transação, recupera mensagens; é também nessa fase que o agente de usuário pode marcar mensagens que devem ser apagadas, remover essas marcas e obter estatísticas de correio. A terceira fase, atualização, ocorre após o cliente ter dado o comando `quit` que encerra a sessão POP3. Nesse momento, o servidor de correio apaga as mensagens que foram marcadas.

Em uma transação POP3, o agente de usuário emite comandos e o servidor, uma resposta para cada um deles. Há duas respostas possíveis: `+OK` (às vezes seguida de dados do servidor para o cliente), usada pelo servidor para indicar que correu tudo bem com o comando anterior e `-ERR`, que o servidor usa para informar que houve algo errado com o comando anterior.

A fase de autorização tem dois comandos principais: `user <user name>` e `pass <password>`. Para ilustrar esses dois comandos, sugerimos que você realize uma sessão Telnet diretamente com um servidor POP3, usando a porta 110, e emita os dois comandos. Suponha que `mailServer` seja o nome de seu servidor de correio. O que você verá será algo parecido com:

```
telnet mailServer 110
+OK POP3 server ready
user bob
+OK
pass hungry
+OK user successfully logged on
```

Se você escrever um comando errado, o POP3 responderá com uma mensagem `-ERR`.

Agora, vamos examinar a fase de transação. Um agente de usuário que utiliza POP3 frequentemente pode ser configurado (pelo usuário) para 'ler-e-apagar' ou para 'ler-e-guardar'. A sequência de comandos emitida por um agente de usuário POP3 depende do modo em que o agente de usuário estiver operando. No modo ler-e-apagar, o agente de usuário emite os comandos `list`, `retr` e `dele`. Como exemplo, suponha que o usuário tenha duas mensagens em sua caixa postal. No diálogo abaixo, `C:` (que representa o cliente) é o agente de usuário e `S:` (que representa o servidor), o servidor de correio. A transação será mais ou menos assim:

```
C: list
S: 1 498
S: 2 912
S: .
C: retr 1
S: (blah blah ...
S: .....
S: .....blah)
S: .
C: dele 1
C: retr 2
S: (blah blah ...
S: .....
S: .....blah)
S: .
C: dele 2
C: quit
S: +OK POP3 server signing off
```

O agente de usuário primeiramente pede ao servidor de correio que apresente o tamanho de cada uma das mensagens armazenadas. Então, ele recupera e apaga cada mensagem do servidor. Note que, após a fase de

autorização, o agente de usuário empregou apenas quatro comandos: `list`, `retr`, `dele` e `quit`. A sintaxe para esses comandos é definida no RFC 1939. Depois de processar o comando de saída (`quit`), o servidor POP3 entra na fase de atualização e remove as mensagens 1 e 2 da caixa postal.

Há um problema com o modo ler-e-apagar: o destinatário, Bob, pode ser nômade e querer acessar seu correio de muitas máquinas, por exemplo, do PC de seu escritório, do PC de sua casa e de seu computador portátil. O modo ler-e-apagar reparte as mensagens de correio de Bob entre essas três máquinas; em particular, se ele ler primeiramente uma mensagem no PC de seu escritório, não poderá lê-la novamente mais tarde em seu computador portátil. No modo ler-e-guardar, o agente de usuário deixa as mensagens no servidor de correio após descarregá-las. Nesse caso, Bob pode reler mensagens em máquinas diferentes; pode acessar uma mensagem em seu local de trabalho e, uma semana depois, acessá-la novamente em casa.

Durante uma sessão POP3 entre um agente de usuário e o servidor de correio, o servidor POP3 mantém alguma informação de estado; em particular, monitora as mensagens do usuário marcadas para apagar. Contudo, não mantém informação de estado entre sessões POP3. Essa falta de informação simplifica a implementação de um servidor POP3.

IMAP

Usando POP3, assim que baixar suas mensagens na máquina local, Bob pode criar pastas de correspondência e transferir as mensagens baixadas para elas. Em seguida, pode apagar as mensagens, mudá-las de pastas e procurar mensagens (por nome de remetente ou assunto). Mas esse paradigma — pastas e mensagens na máquina local — apresenta um problema para o usuário nômade que gostaria de manter uma hierarquia de pastas em um servidor remoto que possa ser acessado de qualquer computador: com o POP3, isso não é possível. Esse protocolo não provê nenhum meio para um usuário criar pastas remotas e designar mensagens a pastas.

Para resolver esse e outros problemas, foi inventado o protocolo IMAP, definido no RFC 3501. Como o POP3, o IMAP é um protocolo de acesso a correio, porém com mais recursos, mas é também significativamente mais complexo. (E, portanto, também as implementações dos lados cliente e servidor são significativamente mais complexas.)

Um servidor IMAP associa cada mensagem a uma pasta. Quando uma mensagem chega a um servidor pela primeira vez, é associada com a pasta INBOX do destinatário, que, então, pode transferir a mensagem para uma nova pasta criada por ele, lê-la, apagá-la e assim por diante. O protocolo IMAP provê comandos que permitem que os usuários criem pastas e transfiram mensagens de uma para outra. O protocolo também provê comandos que os usuários podem usar para pesquisar pastas remotas em busca de mensagens que obejam critérios específicos. Note que, diferentemente do POP3, um servidor IMAP mantém informação de estado de usuário entre sessões IMAP — por exemplo, os nomes das pastas e quais mensagens estão associadas a elas.

Outra característica importante do IMAP é que ele tem comandos que permitem que um agente de usuário obtenha componentes de mensagens. Por exemplo, um agente de usuário pode obter apenas o cabeçalho ou somente uma das partes de uma mensagem MIME multiparte. Essa característica é útil quando há uma conexão de largura de banda estreita entre o agente de usuário e seu servidor de correio, como, por exemplo, uma conexão de modem sem fio ou de baixa velocidade. Com uma conexão de pequena largura de banda, o usuário pode decidir não baixar todas as mensagens de sua caixa postal, evitando, em particular, mensagens longas que possam conter, por exemplo, um clipe de áudio ou de vídeo. Se quiser ler tudo sobre o IMAP, consulte o site oficial [IMAP, 2009].

E-mail pela Web

Hoje, um número cada vez maior de usuários está enviando e acessando e-mails por meio de seus browsers Web. O Hotmail lançou o e-mail com acesso pela Web em meados da década de 1990; agora, esse tipo de acesso é provido por praticamente todos os sites ISP, bem como universidades e empresas importantes. Com esse serviço, o agente de usuário é um browser Web comum e o usuário se comunica com sua caixa postal remota via HTTP. Quando um destinatário, por exemplo, Bob, quer acessar uma mensagem em sua caixa postal, ela é enviada do servidor de correio para o browser de Bob usando o protocolo HTTP, e não os protocolos POP3 ou IMAP. Quando

um remetente, por exemplo, Alice, quer enviar uma mensagem de e-mail, esta é enviada do browser de Alice para seu servidor de correio por HTTP, e não por SMTP. O servidor de correio de Alice, contudo, ainda envia mensagens para outros servidores de correio e recebe mensagens de outros servidores de correio usando o SMTP.

2.5 DNS: o serviço de diretório da Internet

Nós, seres humanos, podemos ser identificados por diversas maneiras. Por exemplo, podemos ser identificados pelo nome que aparece em nossa certidão de nascimento, pelo número do RG ou da carteira de motorista. Embora cada um desses números possa ser usado para identificar pessoas, em um dado contexto um pode ser mais adequado que outro. Por exemplo, os computadores da Receita Federal preferem usar o número do CPF (de tamanho fixo) ao nome que consta em nossa certidão de nascimento. Por outro lado, pessoas comuns preferem nosso nome de batismo, mais fácil de lembrar, ao número do CPF. (Realmente, você pode se imaginar dizendo: "Oi, meu nome é 132-67-9875. Este é meu marido, 178-87-1146"?)

Assim como seres humanos podem ser identificados de muitas maneiras, exatamente o mesmo acontece com hospedeiros da Internet. Um identificador é seu nome de **hospedeiro** (*hostname*). Nomes de hospedeiro — como `cnn.com`, `www.yahoo.com`, `gaia.cs.umass.edu` e `cis.poly.edu` — são fáceis de lembrar e, portanto, apreciados pelos seres humanos. Todavia, eles fornecem pouca, se é que alguma, informação sobre a localização de um hospedeiro na Internet. (Um nome como `www.eurecom.fr`, que termina com o código do país `.fr`, nos informa que o hospedeiro provavelmente está na França, mas não diz muito mais do que isso.) Além disso, como nomes de hospedeiros podem consistir em caracteres alfanuméricos de comprimento variável, seriam difíceis de ser processados por roteadores. Por essas razões, hospedeiros também são identificados pelo que denominamos **endereços IP**. Discutiremos endereços IP mais detalhadamente no Capítulo 4, mas é importante falar um pouco sobre eles agora. Um endereço IP é constituído de 4 bytes e sua estrutura hierárquica é rígida. Ele é semelhante a `121.7.106.83`, no qual cada ponto separa um dos bytes expressos em notação decimal de 0 a 255. Um endereço IP é hierárquico porque, ao examiná-lo da esquerda para a direita, obtemos gradativamente mais informações específicas sobre onde o hospedeiro está localizado na Internet (isto é, em qual rede, dentre as muitas que compõem a Internet). De maneira semelhante, quando examinamos um endereço postal de cima para baixo, obtemos informações cada vez mais específicas sobre a localização do destinatário.

2.5.1 Serviços fornecidos pelo DNS

Acabamos de ver que há duas maneiras de identificar um hospedeiro — por um nome de hospedeiro e por um endereço IP. As pessoas preferem o identificador nome de hospedeiro por ser mais fácil de lembrar, ao passo que roteadores preferem endereços IP de comprimento fixo e estruturados hierarquicamente. Para conciliar essas preferências, é necessário um serviço de diretório que traduza nomes de hospedeiro para endereços IP. Esta é a tarefa principal do DNS (*domain name system — sistema de nomes de domínio*) da Internet. O DNS é (1) um banco de dados distribuído implementado em uma hierarquia de servidores de nome (**servidores DNS**), e (2) um protocolo de camada de aplicação que permite que hospedeiros consultem o banco de dados distribuído. Os servidores de nome são frequentemente máquinas UNIX que executam o software BIND (Berkeley Internet Name Domain) [BIND, 2009]. O protocolo DNS utiliza UDP e usa a porta 53.

O DNS é comumente empregado por outras entidades da camada de aplicação — inclusive HTTP, SMTP e FTP — para traduzir nomes de hospedeiros fornecidos por usuários para endereços IP. Como exemplo, considere o que acontece quando um browser (isto é, um cliente HTTP), que executa na máquina de algum usuário, requisita o URL `www.someschool.edu/index.html`. Para que a máquina do usuário possa enviar uma mensagem de requisição HTTP ao servidor Web `www.someschool.edu`, ela precisa primeiramente obter o endereço IP de `www.someschool.edu`. Isso é feito da seguinte maneira:

1. A própria máquina do usuário executa o lado cliente da aplicação DNS.
2. O browser extrai o nome de hospedeiro, `www.someschool.edu`, do URL e passa o nome para o lado cliente da aplicação DNS.

3. O cliente DNS envia uma consulta contendo o nome do hospedeiro para um servidor DNS.
4. O cliente DNS finalmente recebe uma resposta, que inclui o endereço IP correspondente ao nome de hospedeiro.
5. Tão logo o browser receba o endereço do DNS, pode abrir uma conexão TCP com o processo servidor HTTP localizado naquele endereço IP.

Vemos, por esse exemplo, que o DNS adiciona mais um atraso — às vezes substancial — às aplicações de Internet que o usam. Felizmente, como discutiremos mais adiante, o endereço IP procurado frequentemente está no cache de um servidor DNS ‘próximo’, o que ajuda a reduzir o tráfego DNS na rede, bem como o atraso médio do DNS.

O DNS provê alguns outros serviços importantes além da tradução de nomes de hospedeiro para endereços IP:

Apelidos de hospedeiro. Um hospedeiro com nome complicado pode ter um ou mais apelidos. Um nome como `relay1.west-coast.enterprise.com` pode ter, por exemplo, dois apelidos, como `enterprise.com` e `www.enterprise.com`. Nesse caso, o nome de hospedeiro `relay1.west-coast.enterprise.com` é denominado **nome canônico**. Apelidos, quando existem, são em geral mais fáceis de lembrar do que o nome canônico. O DNS pode ser chamado por uma aplicação para obter o nome canônico correspondente a um apelido fornecido, bem como para obter o endereço IP do hospedeiro.

Apelidos de servidor de correio. Por razões óbvias, é adequado que endereços de e-mail sejam fáceis de lembrar. Por exemplo, se Bob tem uma conta no Hotmail, seu endereço de e-mail pode ser simplesmente `bob@hotmail.com`. Contudo, o nome de hospedeiro do servidor do Hotmail é mais complicado e muito mais difícil de lembrar do que simplesmente `hotmail.com` (por exemplo, o nome canônico pode ser algo parecido com `relay1.west-coast.hotmail.com`). O DNS pode ser chamado por uma aplicação de correio para obter o nome canônico a partir de um apelido fornecido, bem como o endereço IP do hospedeiro. Na verdade, o registro MX (veja adiante) permite que o servidor de correio e o servidor Web de uma empresa tenham nomes (apelidos) idênticos; por exemplo, o servidor Web e o servidor de correio de uma empresa podem ambos ser denominados `enterprise.com`.

Distribuição de carga. O DNS também é usado para realizar distribuição de carga entre servidores replicados, tais como os servidores Web replicados. Sites movimentados como `cnn.com` são replicados em vários servidores, sendo que cada servidor roda em um sistema final diferente e tem um endereço IP diferente. Assim, no caso de servidores Web replicados, um conjunto de endereços IP fica associado a um único nome canônico e contido no banco de dados do DNS. Quando clientes consultam um nome mapeado para um conjunto de endereços, o DNS responde com o conjunto inteiro de endereços IP, mas faz um rodízio da ordem dos endereços dentro de cada resposta. Como um cliente normalmente envia sua mensagem de requisição HTTP ao endereço IP que ocupa o primeiro lugar no conjunto, o rodízio

Princípios na prática

DNS: funções decisivas de rede via paradigma cliente-servidor

Como os protocolos HTTP, FTP e SMTP, o DNS é um protocolo da camada de aplicação, já que (1) roda entre sistemas finais comunicantes usando o paradigma cliente-servidor e (2) depende de um protocolo de transporte fim a fim subjacente para transferir mensagens DNS entre sistemas finais comunicantes. Em outro sentido, contudo, o papel do DNS é bastante diferente das aplicações Web, da transferência de arquivo e do e-mail. Diferentemente dessas aplicações, o DNS não é uma aplicação com a qual o usuário interage diretamente. Em vez disso, fornece uma função interna da Internet — a saber, a tradução de nomes de hospedeiro para seus endereços IP subjacentes, para aplicações de usuário e outros softwares da Internet. Notamos, na Seção 1.2, que muito da complexidade da arquitetura da Internet está localizada na ‘periferia’ da rede. O DNS, que implementa o processo crucial de tradução de nome para endereço usando clientes e servidores localizados nas bordas da rede, é mais um exemplo dessa filosofia de projeto.

de DNS distribui o tráfego entre os servidores replicados. O rodízio de DNS também é usado para e-mail, de modo que vários servidores de correio podem ter o mesmo apelido. Recentemente, empresas distribuidoras de conteúdo como a Akamai [Akamai, 2009] passaram a usar o DNS de maneira mais sofisticada para prover distribuição de conteúdo na Web (veja Capítulo 7).

O DNS está especificado no RFC 1034 e no RFC 1035 e atualizado em diversos RFCs adicionais. É um sistema complexo e, neste livro, apenas mencionamos os aspectos fundamentais de sua operação. O leitor interessado pode consultar os RFCs citados, o livro escrito por Abitz e Liu [Abitz, 1993] e também o artigo de [Mockapetris, 1988], que apresenta uma retrospectiva e uma ótima descrição do que e do porquê do DNS, e [Mockapetris, 2005].

2.5.2 Visão geral do modo de funcionamento do DNS

Apresentaremos, agora, uma visão panorâmica do modo de funcionamento do DNS. Nossa discussão focalizará o serviço de tradução de nome de hospedeiro para endereço IP.

Suponha que uma certa aplicação (como um browser Web ou um leitor de correio), que executa na máquina de um usuário, precise traduzir um nome de hospedeiro para um endereço IP. A aplicação chamará o lado cliente do DNS, especificando o nome de hospedeiro que precisa ser traduzido. (Em muitas máquinas UNIX, `gethostbyname()` é a chamada de função que uma aplicação invoca para realizar a tradução. Na Seção 2.7, mostraremos como uma aplicação Java pode chamar o DNS). A partir daí, o DNS do hospedeiro do usuário assume o controle, enviando uma mensagem de consulta para dentro da rede. Todas as mensagens de consulta e de resposta do DNS são enviadas dentro de datagramas UDP à porta 53. Após um atraso na faixa de milissegundos a segundos, o DNS no hospedeiro do usuário recebe uma mensagem de resposta DNS fornecendo o mapeamento desejado, que é, então, passado para a aplicação que está interessada. Portanto, do ponto de vista dessa aplicação, que está na máquina do cliente, o DNS é uma caixa-preta que provê um serviço de tradução simples e direto. Mas, na realidade, a caixa-preta que implementa o serviço é complexa, consistindo em um grande número de servidores de nomes distribuídos ao redor do mundo, bem como em um protocolo de camada de aplicação que especifica como se comunicam os servidores de nomes e os hospedeiros que fazem a consulta.

Um arranjo simples para DNS seria ter um servidor de nomes contendo todos os mapeamentos. Nesse projeto centralizado, os clientes simplesmente dirigiriam todas as consultas a esse único servidor de nomes, que responderia diretamente aos clientes que estão fazendo a consulta. Embora a simplicidade desse arranjo seja atraente, ele não é adequado para a Internet de hoje com seu vasto (e crescente) número de hospedeiros. Dentre os problemas de um arranjo centralizado, estão:

- **Um único ponto de falha.** Se o servidor de nomes quebrar, a Internet inteira quebrará!
- **Volume de tráfego.** Um único servidor de nomes teria de manipular todas as consultas DNS (para todas as requisições HTTP e mensagens de e-mail geradas por centenas de milhões de hospedeiros).
- **Banco de dados centralizado distante.** Um único servidor de nomes nunca poderia estar ‘próximo’ de todos os clientes que fazem consultas. Se colocarmos o único servidor de nomes na cidade de Nova York, todas as buscas provenientes da Austrália terão de viajar até o outro lado do globo, talvez por linhas lentas e congestionadas, o que pode resultar em atrasos significativos.
- **Manutenção.** O único servidor de nomes teria de manter registros de todos os hospedeiros da Internet. Esse banco de dados não somente seria enorme, mas também teria de ser atualizado frequentemente para atender a todos os novos hospedeiros.

Em resumo, um banco de dados centralizado em um único servidor DNS simplesmente não é escalável. Consequentemente, o DNS é distribuído por conceito de projeto. Na verdade, ele é um ótimo exemplo de como um banco de dados distribuído pode ser implementado na Internet.

Um banco de dados distribuído, hierárquico

Para tratar da questão da escala, o DNS usa um grande número de servidores, organizados de maneira hierárquica e distribuídos por todo o mundo. Nenhum servidor de nomes isolado tem todos os mapeamentos para

todos os hospedeiros da Internet. Em vez disso, os mapeamentos são distribuídos pelos servidores de nomes. Como uma primeira aproximação, há três classes de servidores de nomes: servidores de nomes raiz, servidores DNS de domínio de alto nível (*top-level domain* — TLD) e servidores DNS com autoridade — organizados em uma hierarquia, como mostra a Figura 2.19.

Para entender como essas três classes de servidores interagem, suponha que um cliente DNS queira determinar o endereço IP para o nome de hospedeiro `www.amazon.com`. Como uma primeira aproximação, ocorrerão os seguintes eventos. Em primeiro lugar, o cliente contatará um dos servidores raiz, que retornará endereços IP dos servidores TLD para o domínio de alto nível `.com`. Então, o cliente contatará um desses servidores TLD, que retornará o endereço IP de um servidor com autoridade para `amazon.com`. Finalmente, o cliente contatará um dos servidores com autoridade para `amazon.com`, que retornará o endereço IP para o nome de hospedeiro `www.amazon.com`. Mais adiante, analisaremos mais detalhadamente esse processo de consulta DNS. Mas, em primeiro lugar, vamos examinar mais de perto as três classes de servidores DNS.

Servidores de nomes raiz. Na Internet há 13 servidores de nomes raiz (denominados de A a M) e a maior parte deles está localizada na América do Norte. Um mapa de servidores de nomes raiz de outubro de 2006 é mostrado na Figura 2.20; uma lista dos servidores de nomes raiz existentes hoje está disponível em [Root-servers, 2009]. Embora tenhamos nos referido a cada um dos 13 servidores de nomes raiz como se fossem um servidor único, na realidade, cada um é um conglomerado de servidores replicados, para fins de segurança e confiabilidade.

Servidores de nomes de Domínio de Alto Nível (TLD). Esses servidores são responsáveis por domínios de alto nível como `.com`, `.org`, `.net`, `.edu` e `.gov`, e por todos os domínios de alto nível de países, tais como `.uk`, `.fr`, `.ca` e `.jp`. A empresa Network Solutions mantinha os servidores TLD para o domínio de alto nível `.com` e a empresa Educause mantinha os servidores para o domínio de alto nível `.edu`.

Servidores de nomes com autoridade. Toda organização que tiver hospedeiros que possam ser acessados publicamente na Internet (como servidores Web e servidores de correio) deve fornecer registros DNS também acessíveis publicamente que mapeiem os nomes desses hospedeiros para endereços IP. Um servidor DNS com autoridade de uma organização abriga esses registros. Uma organização pode preferir implementar seu próprio servidor DNS com autoridade para abrigar esses registros ou, como alternativa, pode pagar para armazená-los em um servidor DNS com autoridade de algum provedor de serviço. A maioria das universidades e empresas de grande porte implementam e mantêm seus próprios servidores DNS primário e secundário (backup) com autoridade.

Os servidores de nomes raiz, TLD e com autoridade pertencem à hierarquia de servidores DNS, como mostra a Figura 2.19. Há um outro tipo importante de DNS, denominado **servidor DNS local**, que não pertence, estreitamente, à hierarquia de servidores, mas, mesmo assim, é central para a arquitetura DNS.

Cada ISP — como o de uma universidade, de um departamento acadêmico, de uma empresa ou de uma residência — tem um servidor de nomes local (também denominado servidor de nomes default). Quando um hospedeiro se conecta com um ISP, o ISP fornece ao hospedeiro os endereços IP de um ou mais de seus servidores

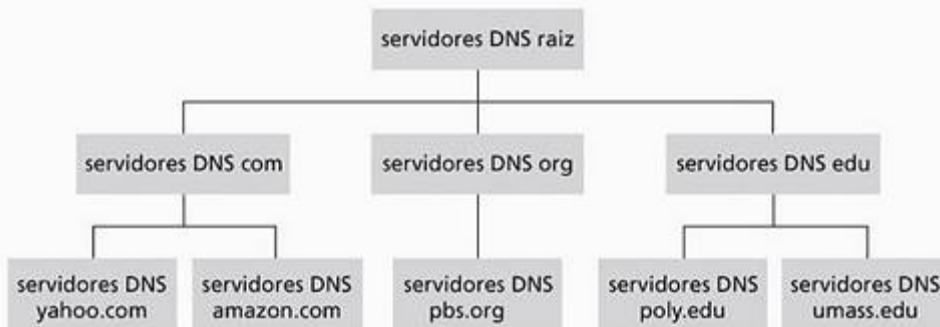


Figura 2.19 Parte da hierarquia de servidores DNSs

de nomes locais (normalmente por DHCP, que será discutido no Capítulo 4). Determinar o endereço IP do seu servidor de nomes local é fácil: basta acessar as janelas de estado da rede no Windows ou UNIX. O servidor de nomes local de um hospedeiro normalmente está 'próximo' dele. No caso de um ISP institucional, pode estar na mesma LAN do hospedeiro; já no caso de um ISP residencial, em geral o servidor de nomes está separado do hospedeiro por não mais do que alguns roteadores. Quando um hospedeiro faz uma consulta ao DNS, ela é enviada ao servidor de nomes local, que age como proxy e a retransmite para a hierarquia do servidor DNS, como discutiremos mais detalhadamente a seguir.

Vamos examinar um exemplo simples. Suponha que o hospedeiro `cis.poly.edu` deseje o endereço IP de `gaia.cs.umass.edu`. Suponha também que o servidor de nomes local da Polytechnic seja denominado `dns.poly.edu` e que um servidor de nomes com autoridade para `gaia.cs.umass.edu` seja denominado `dns.umass.edu`. Como mostra a Figura 2.21, o hospedeiro `cis.poly.edu` primeiramente envia uma mensagem de consulta DNS a seu servidor de nomes local `dns.poly.edu`. Essa mensagem contém o nome de hospedeiro a ser traduzido, isto é, `gaia.cs.umass.edu`. O servidor de nomes local transmite a mensagem de consulta a um servidor de nomes raiz, que percebe o sufixo `edu` e retorna ao servidor de nomes local uma lista de endereços IP contendo servidores TLD responsáveis por `edu`. Então, o servidor de nomes local retransmite a mensagem de consulta a um desses servidores TLD que, por sua vez, percebe o sufixo `umass.edu` e responde com o endereço IP do servidor de nomes com autoridade para a University of Massachusetts, a saber, `dns.umass.edu`. Finalmente, o servidor de nomes local reenvia a mensagem de consulta diretamente a `dns.umass.edu`, que responde com o endereço IP de `gaia.cs.umass.edu`. Note que, nesse exemplo, para poder obter o mapeamento para um único nome de hospedeiro, foram enviadas oito mensagens DNS: quatro mensagens de consulta e quatro mensagens de resposta! Em breve veremos como o cache de DNS reduz esse tráfego de consultas.

Nosso exemplo anterior afirmou que o servidor TLD conhece o servidor de nomes com autoridade para o nome de hospedeiro, o que nem sempre é verdade. Ele pode conhecer apenas um servidor de nomes intermediário que, por sua vez, conhece o servidor de nomes com autoridade para o nome de hospedeiro. Por exemplo, suponha novamente que a Universidade de Massachusetts tenha um servidor de nomes para a universidade denominado `dns.umass.edu`. Imagine também que cada um dos departamentos da universidade tenha seu próprio servidor de nomes e que cada servidor de nomes departamental seja um servidor de nomes com autoridade para todos os hospedeiros do departamento. Nesse caso, quando o servidor de nomes intermediário `dns.umass.edu` receber uma consulta para um hospedeiro cujo nome termina com `cs.umass.edu`, ele retornará a `dns.poly.edu` o endereço IP de `dns.cs.umass.edu`, que tem autoridade para todos os nomes de hospedeiro que terminam com `cs.umass.edu`. Então, o servidor de nomes local `dns.poly.edu` enviará a consulta ao servidor de nomes com



Figura 2.20 Servidores DNS raiz em 2009 (nome, organização, localização)

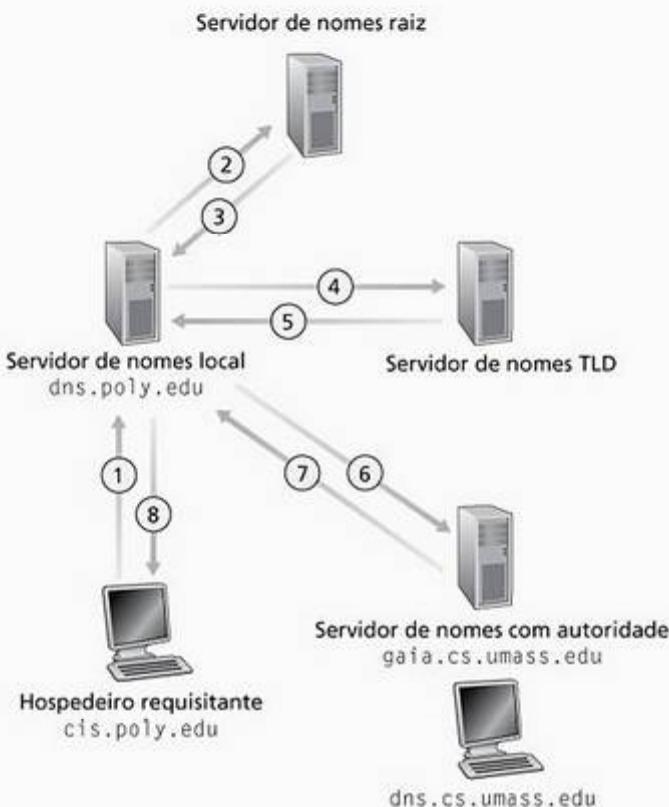


Figura 2.21 Intereração de vários servidores DNS

autoridade, que retornará o mapeamento desejado para o servidor de nomes local e que, por sua vez, o repassará ao hospedeiro requisitante. Nesse caso, serão enviadas dez mensagens DNS no total!

O exemplo mostrado na Figura 2.21 usa **consultas recursivas** e **consultas iterativas**. A consulta enviada de `cis.poly.edu` para `dns.poly.edu` é recursiva, visto que pede a `dns.poly.edu` que obtenha o mapeamento em seu nome. Mas as três consultas subsequentes são iterativas, visto que todas as respostas são retornadas diretamente a `dns.poly.edu`. Teoricamente, qualquer consulta DNS pode ser iterativa ou recursiva. Por exemplo, a Figura 2.22 mostra uma cadeia de consultas DNS na qual todas elas são recursivas. Na prática, as consultas normalmente seguem o padrão mostrado na Figura 2.21: a consulta do hospedeiro requisitante ao servidor de nomes local é recursiva e todas as outras são iterativas.

Cache DNS

Até aqui, nossa discussão ignorou o **cache DNS**, uma característica muito importante do sistema DNS. Na realidade, o DNS explora extensivamente o cache para melhorar o desempenho quanto ao atraso e reduzir o número de mensagens DNS que ricocheteia pela Internet. A ideia que está por trás do cache DNS é muito simples. Em uma cadeia de consultas, quando um servidor de nomes recebe uma resposta DNS (contendo, por exemplo, o mapeamento de um nome de hospedeiro para um endereço IP), ele pode fazer cache das informações da resposta em sua memória local. Por exemplo, na Figura 2.21, toda vez que o servidor de nomes local `dns.poly.edu` recebe uma resposta de algum servidor DNS, pode fazer cache de qualquer informação contida na resposta. Se um par nome de hospedeiro/endereço IP estiver no cache de um servidor DNS e outra consulta chegar ao mesmo servidor para o mesmo nome de máquina, o servidor de nomes poderá fornecer o endereço IP desejado, mesmo que não tenha autoridade para esse nome. Como hospedeiros e mapeamentos entre hospedeiros e endereços IP não são, de modo algum, permanentes, após um período de tempo (frequentemente dois dias), os servidores DNS descartam as informações armazenadas em seus caches.

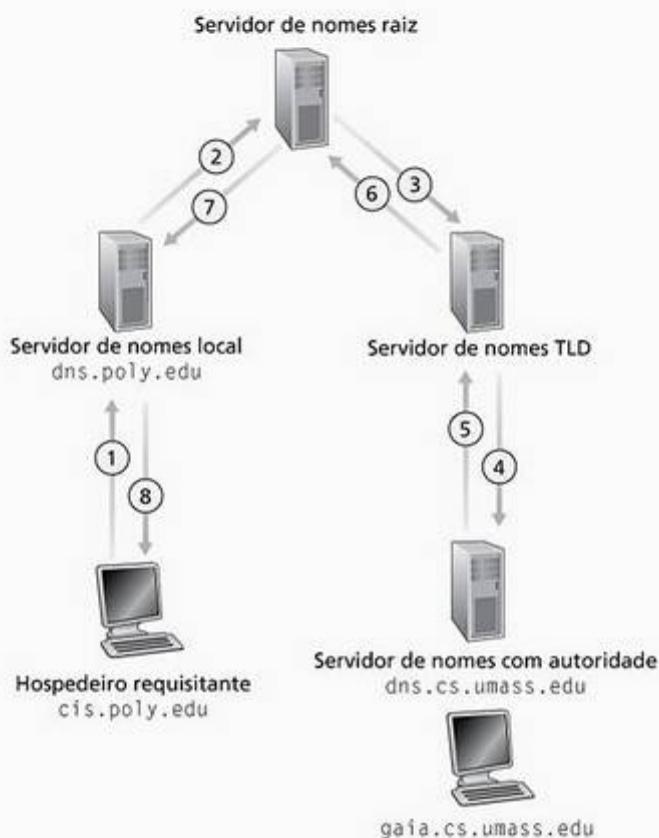


Figura 2.22 Consultas recursivas em DNS

Como exemplo, imagine que um hospedeiro `apricot.poly.edu` consulte `dns.poly.edu` para o endereço IP da máquina `cnn.com`. Além disso, suponha que algumas horas mais tarde outra máquina da Polytechnic University, digamos, `kiwi.poly.fr` também consulte `dns.poly.edu` para o mesmo nome de hospedeiro. Por causa do cache, o servidor local poderá imediatamente retornar o endereço IP de `cnn.com` a esse segundo hospedeiro requisitante, sem ter de consultar quaisquer outros servidores DNS. Um servidor de nomes local também pode fazer cache de endereços IP de servidores TLD, permitindo, assim, que servidores de nomes locais evitem os servidores de nomes raiz em uma cadeia de consultas (isso acontece frequentemente).

2.5.3 Registros e mensagens DNS

Os servidores de nomes que juntos implementam o banco de dados distribuído do DNS armazenam registros de recursos (RR) que fornecem mapeamentos de nomes de hospedeiros para endereços IP. Cada mensagem de resposta DNS carrega um ou mais registros de recursos. Nesta seção e na subsequente, apresentaremos uma breve visão geral dos registros de recursos e mensagens DNS. Para mais detalhes, consulte [Abitz, 1993] ou [RFC 1034; RFC 1035].

Um registro de recurso é uma tupla de quatro elementos que contém os seguintes campos:
 (Name, Value, Type, TTL)

TTL é o tempo de vida útil do registro de recurso; determina quando um recurso deve ser removido de um cache. Nos exemplos de registros dados a seguir, ignoramos o campo TTL. Os significados de Name e Value dependem de Type:

Se Type=A, então Name é um nome de hospedeiro e Value é o endereço IP para o nome de hospedeiro. Assim, um registro Type A fornece o mapeamento padrão de nomes hospedeiros para endereços IP. Como exemplo, (`relay1.bar.foo.com`, `145.37.93.126`, A) é um registro com Type igual a A.

Se Type=NS, então Name é um domínio (como foo.com) e Value é o nome de um servidor de nomes com autoridade que sabe como obter os endereços IP para hospedeiros do domínio. Esse registro é usado para encaminhar consultas DNS ao longo da cadeia de consultas. Como exemplo, (foo.com, dns.foo.com, NS) é um registro com Type igual a NS.

Se Type=CNAME, então Value é um nome canônico de hospedeiro para o apelido de hospedeiro contido em Name. Esse registro pode fornecer aos hospedeiros consultantes o nome canônico correspondente a um apelido de hospedeiro. Como exemplo, (foo.com, relay1.bar.foo.com, CNAME) é um registro CNAME.

Se Type=MX, então Value é o nome canônico de um servidor de correio cujo apelido de hospedeiro está contido em Name. Como exemplo, (foo.com, mail.bar.foo.com, MX) é um registro MX. Registros MX permitem que os nomes de hospedeiros de servidores de correio tenham apelidos simples. Note que usando o registro MX, uma empresa pode ter o mesmo apelido para seu servidor de arquivo e para um de seus outros servidores (tal como seu servidor Web). Para obter o nome canônico do servidor de correio, um cliente DNS consultaria um registro MX; para obter o nome canônico do outro servidor, o cliente DNS consultaria o registro CNAME.

Se um servidor de nomes tiver autoridade para um determinado nome de hospedeiro, então conterá um registro Type A para o nome de hospedeiro. (Mesmo que não tenha autoridade, o servidor de nomes pode conter um registro Type A em seu cache.) Se um servidor não tiver autoridade para um nome de hospedeiro, conterá um registro Type NS para o domínio que inclui o nome e um registro Type A que fornece o endereço IP do servidor de nomes no campo Value do registro NS. Como exemplo, suponha que um servidor TLD.edu não tenha autoridade para o hospedeiro gaia.cs.umass.edu. Nesse caso, esse servidor conterá um registro para um domínio que inclui o hospedeiro gaia.cs.umass.edu, por exemplo (umass.edu, dns.umass.edu, NS). O servidor TLD.edu conterá também um registro Type A, que mapeia o servidor de nome dns.umass.edu para um endereço IP, por exemplo (dns.umass.edu, 128.119.40.111, A).

Mensagens DNS

Abordamos anteriormente nesta seção mensagens de consulta e de resposta DNS, que são as duas únicas espécies de mensagem DNS. Além disso, tanto as mensagens de consulta como as de resposta têm o mesmo formato, como mostra a Figura 2.23. A semântica dos vários campos de uma mensagem DNS é a seguinte:

Os primeiros 12 bytes formam a seção de cabeçalho, que tem vários campos. O primeiro campo é um número de 16 bits que identifica a consulta. Esse identificador é copiado para a mensagem de resposta a uma consulta, permitindo que o cliente combine respostas recebidas com consultas enviadas. Há várias flags no campo de flag. Uma flag de consulta/resposta de 1 bit indica se a mensagem é uma consulta (0) ou uma resposta (1). Uma flag de autoridade de 1 bit é marcada em uma mensagem de resposta quando o servidor de nomes é um servidor com autoridade para um nome consultado. Uma flag de recursão desejada de 1 bit é estabelecida quando um cliente (hospedeiro ou servidor de nomes) quer que um servidor de nomes proceda recursivamente sempre que não tem o registro. Um campo de recursão disponível de 1 bit é marcado em uma resposta se o servidor de nomes suporta buscas recursivas. No cabeçalho, há também quatro campos de 'número de'. Esses campos indicam o número de ocorrências dos quatro tipos de seção de dados que se seguem ao cabeçalho.

A seção de pergunta contém informações sobre a consulta que está sendo feita. Essa seção inclui (1) um campo de nome que contém o nome que está sendo consultado e (2) um campo de tipo que indica o tipo de pergunta que está sendo feito sobre o nome — por exemplo, um endereço de hospedeiro associado a um nome (Type A) ou o servidor de correio para um nome (Type MX).

Em uma resposta de um servidor de nomes, a seção de resposta contém os registros de recursos para o nome que foi consultado originalmente. Lembre-se de que em cada registro de recurso há o Type (por exemplo, A, NS, CSNAME e MX), o Value e o TTL. Uma resposta pode retornar vários RRs, já que

Identificação	Flags	
Número de perguntas	Número de RRs de resposta	12 bytes
Número de RRs com autoridade	Número de RRs adicionais	
Perguntas (número variável de perguntas)		Nome, campos de tipo para uma consulta
Respostas (número variável de registros de recursos)		RRs de resposta à consulta
Autoridade (número variável de registros de recursos)		Registros para servidores com autoridade
Informação adicional (número variável de registros de recursos)		Informação adicional 'útil', que pode ser usada

Figura 2.23 Formato da mensagem DNS

um nome de hospedeiro pode ter diversos endereços IP (por exemplo, para servidores Web replicados, como já discutimos anteriormente nesta seção).

A seção de autoridade contém registros de outros servidores com autoridade.

A seção adicional contém outros registros úteis. Por exemplo, o campo resposta em uma resposta a uma consulta MX conterá um registro de recurso que informa o nome canônico de um servidor de correio. A seção adicional conterá um registro Type A que fornece o endereço IP para o nome canônico do servidor de correio.

Você gostaria de enviar uma mensagem de consulta DNS diretamente de sua máquina a algum servidor DNS? Isso pode ser feito facilmente com o **programa nslookup**, que está disponível na maioria das plataformas Windows e UNIX. Por exemplo, se um hospedeiro executar Windows, abra o Command Prompt e chame o programa nslookup simplesmente digitando 'nslookup'. Depois de chamar o programa, você pode enviar uma consulta DNS a qualquer servidor de nomes (raiz, TLD ou com autoridade). Após receber a mensagem de resposta do servidor de nomes, o nslookup apresentará os registros incluídos na resposta (em formato que pode ser lido normalmente). Como alternativa a executar nslookup na sua própria máquina, você pode visitar um dos muitos sites Web que permitem o emprego remoto do programa. (Basta digitar 'nslookup' em um buscador e você será levado a um desses sites.)

Para inserir registros no banco de dados do DNS

A discussão anterior focalizou como são extraídos registros do banco de dados DNS. É possível que você esteja se perguntando como os registros entraram no banco de dados em primeiro lugar. Vamos examinar como isso é feito no contexto de um exemplo específico. Imagine que você acabou de criar uma nova empresa muito interessante denominada Network Utopia. A primeira coisa que você certamente deverá fazer é registrar o nome de domínio `networkutopia.com` em uma entidade registradora. Uma **entidade registradora** é uma entidade comercial que verifica se o nome de domínio é exclusivo, registra-o no banco de dados do DNS (como discutiremos mais adiante) e cobra uma pequena taxa por seus serviços. Antes de 1999, uma única entidade registradora, a Network Solutions, detinha o monopólio do registro de nomes para os domínios `.com`, `.net` e `.org`. Mas agora existem muitas entidades registradoras credenciadas pela Internet Corporation for Assigned Names and Numbers (ICANN) competindo por clientes. Uma lista completa dessas entidades está disponível em <http://www.internic.net>.



Segurança em foco

Vulnerabilidades do DNS

Vimos que o DNS é um componente fundamental da infraestrutura da Internet, com muitos serviços importantes — incluindo a Web e o e-mail — simplesmente incapazes de funcionar sem ele. Desta maneira, perguntamos: como o DNS pode ser atacado? O DNS é um alvo esperando para ser atingido, pois causa dano à maioria das aplicações da Internet junto com ele?

O primeiro tipo de ataque que vem à mente é o ataque inundação na largura de banda DDoS (veja a Seção 1.6) contra servidores DNS. Por exemplo, um atacante pode tentar enviar para cada servidor DNS raiz uma inundação de pacotes, fazendo com que a maioria das consultas DNS legítimas nunca seja respondida. Tal ataque DDoS em larga escala contra servidores DNS raiz aconteceu em 21 de outubro de 2002. Nesse ataque, os atacantes se aproveitavam de um botnet para enviar centenas de mensagens ping para cada um dos servidores DNS raiz. (As mensagens ICMP são discutidas no Capítulo 4. Por enquanto, basta saber que os pacotes ICMP são tipos especiais de datagramas IP.) Felizmente, esse ataque em larga escala causou um dano mínimo, tendo um pequeno ou nenhum impacto sobre a experiência dos usuários com a Internet. Os atacantes obtiveram êxito ao direcionar centenas de pacotes aos servidores raiz. Mas muitos dos servidores DNS raiz foram protegidos por filtros de pacotes, configurados para sempre bloquear todas as mensagens ping ICMP encaminhadas aos servidores raiz. Desse modo, esses servidores protegidos foram poupadados e funcionaram normalmente. Além disso, a maioria dos servidores DNS locais oculta os endereços IP dos servidores de domínio de nível superior, permitindo que o processo de consulta ultrapasse frequentemente os servidores DNS raiz.

Um ataque DDoS potencialmente mais eficaz contra o DNS seria enviar uma inundação de consultas DNS aos servidores de domínio de alto nível, por exemplo, para todos os servidores de domínio que lidam com o domínio .com. Seria mais difícil filtrar as consultas DNS direcionadas aos servidores DNS, e os servidores de domínio de alto nível não são ultrapassados tão facilmente quanto os servidores raiz. Mas a gravidade de tal ataque poderia ser parcialmente amenizada pelo cache nos servidores DNS locais.

O DNS poderia ser atacado potencialmente de outras maneiras. Em um ataque de homem no meio, o atacante intercepta consultas do hospedeiro e retorna respostas falsas. No ataque de envenenamento, o atacante envia respostas falsas a um servidor DNS, fazendo com que o servidor armazene os registros falsos em sua cache. Ambos os ataques podem ser utilizados, por exemplo, para redirecionar um usuário da Web inocente ao site Web do atacante. Esses ataques, entretanto, são difíceis de implementar, uma vez que requerem a intercepção de pacotes ou o estrangulamento de servidores [Skoudis, 2006].

Outro ataque DNS importante não é um ataque ao serviço DNS por si mesmo, mas, em vez disso, se aproveitar da infraestrutura do DNS para lançar um ataque DDoS contra um hospedeiro-alvo (por exemplo, o servidor de mensagens de sua universidade). Nesse ataque, o atacante envia consultas DNS para muitos servidores DNS autoritativos, com cada consulta tendo o endereço-fonte falsificado do hospedeiro-alvo. Os servidores DNS, então, enviam suas respostas diretamente para o hospedeiro-alvo. Se as consultas puderem ser realizadas de tal maneira que uma resposta seja muito maior (em bytes) do que uma consulta (denominada amplificação), então o atacante pode entupir o alvo sem ter que criar muito de seu próprio tráfego. Tais ataques de reflexão que exploram o DNS possuem um sucesso limitado até hoje [Mirkovic, 2005].

Em resumo, não houve um ataque que tenha interrompido o serviço DNS com sucesso. Houve ataques refletores bem-sucedidos; entretanto, eles podem ser (e estão sendo) abordados por uma configuração apropriada de servidores DNS.

Ao registrar o nome de domínio networkutopia.com, você também precisará informar os nomes e endereços IP dos seus servidores DNS com autoridade, primários e secundários. Suponha que os nomes e endereços IP sejam dns1.networkutopia.com, dns2.networkutopia.com, 212.212.212.1 e 212.212.212.2. A entidade registradora ficará encarregada de providenciar a inserção dos registros Type NS e de um registro Type A nos servidores TLD do domínio com para cada um desses dois servidores de nomes com autoridade. Especificamente para o servidor primário com autoridade networkutopia.com, a autoridade registradora inseriria no sistema DNS os dois registros de recursos seguintes:

(networkutopia.com, dns1.networkutopia.com, NS)
(dns1.networkutopia.com, 212.212.212.1, A)

Não esqueça de providenciar também a inserção em seus servidores de nomes com autoridade do registro de recurso Type A para seu servidor Web www.networkutopia.com e o registro de recurso Type MX para seu servidor de correio mail.networkutopia.com. (Até há pouco tempo, o conteúdo de cada servidor DNS era configurado estaticamente, por exemplo, a partir de um arquivo de configuração criado por um gerenciador de sistema. Mais recentemente, foi acrescentada ao protocolo DNS uma opção UPDATE que permite que dados sejam dinamicamente acrescentados no banco de dados ou apagados deles por meio de mensagens DNS. O [RFC 2136] e o [RFC 3007] especificam atualizações dinâmicas do DNS.)

Quando todas essas etapas estiverem concluídas, o público em geral poderá visitar seu site Web e enviar e-mails aos empregados de sua empresa. Vamos concluir nossa discussão do DNS verificando que essa afirmação é verdadeira, o que também ajudará a solidificar aquilo que aprendemos sobre o DNS. Suponha que Alice, que está na Austrália, queira consultar a página Web www.networkutopia.com. Como discutimos anteriormente, seu hospedeiro primeiramente enviará uma consulta DNS a seu servidor de nomes local, que então contatará um servidor TLD do domínio com. (O servidor de nomes local também terá de contatar um servidor de nomes raiz caso não tenha em seu cache o endereço de um servidor TLD com.) Esse servidor TLD contém os registros de recursos Type NS e Type A citados anteriormente, porque a entidade registradora já os tinha inserido em todos os servidores TLD com. O servidor TLD com envia uma resposta ao servidor de nomes local de Alice, contendo os dois registros de recursos. Então, o servidor de nomes local envia uma consulta DNS a 212.212.212.1, solicitando o registro Type A correspondente a www.networkutopia.com. Este registro provê o endereço IP do servidor Web desejado, digamos, 212.212.71.4, que o servidor local de nomes transmite para o hospedeiro de Alice. Agora, o browser de Alice pode iniciar uma conexão TCP com o hospedeiro 212.212.71.4 e enviar uma requisição HTTP pela conexão. Ufa! Acontecem muito mais coisas do que percebemos quando navegamos na Web!

2.6 Aplicações P2P

As aplicações descritas neste capítulo até agora — inclusive a Web, e-mail e DNS — todas empregam arquiteturas cliente-servidor com dependência significativa em servidores com infraestrutura que sempre permanecem ligados. Como consta na Seção 2.1.1, com uma arquitetura P2P, há dependência mínima (se houver) de servidores com infraestrutura que permanecem sempre ligados. Em vez disso, duplas de hospedeiros intermitentemente conectados, chamados pares, comunicam-se diretamente entre si. Os pares não são de propriedade de um provedor de serviços, mas sim de desktops e laptops controlados por usuários.

Nesta seção, examinaremos três diferentes aplicações que são particularmente bem apropriadas a projetos P2P. A primeira é a distribuição de arquivos, em que a aplicação distribui um arquivo a partir de uma única fonte para um grande número de pares. A distribuição de arquivos é um bom local para iniciar a investigação de P2P, visto que expõe claramente a autoescalabilidade de arquiteturas P2P. Como exemplo específico para distribuição de arquivos, descreveremos o popular sistema BitTorrent. A segunda aplicação P2P que examinaremos é um banco de dados distribuído em uma grande comunidade de pares. Para essa aplicação, exploraremos o conceito de uma Distributed Hash Table (DHT). Por fim, para nossa terceira aplicação, examinaremos o Skype, uma aplicação de telefonia P2P da Internet de sucesso fenomenal.

2.6.1 Distribuição de arquivos P2P

Começaremos nossa investida em P2P considerando uma aplicação bastante natural, ou seja, a distribuição de um grande arquivo a partir de um único servidor a um grande número de hospedeiros (chamados pares). O arquivo pode ser uma nova versão do sistema operacional Linux, um patch de software para um sistema operacional ou aplicação existente, um arquivo de música MP3 ou um arquivo de vídeo MPEG. Em uma distribuição de arquivo cliente-servidor, o servidor deve enviar uma cópia do arquivo para cada um dos pares — colocando um enorme fardo sobre o servidor e consumindo uma grande quantidade de banda do servidor. Na distribuição de arquivos P2P, cada par pode redistribuir qualquer parte do arquivo que recebeu para outros pares, auxiliando, assim, o servidor no processo de distribuição. Atualmente (outono [Nos EUA] de 2009), o protocolo de distribuição de arquivos P2P mais popular é o BitTorrent [BitTorrent, 2009]. Originalmente desenvolvido por Bram Cohen (consulte a entrevista com Bram Cohen no final deste capítulo), há atualmente muitos diferentes clientes independentes de BitTorrent conforme o protocolo do BitTorrent, assim como há diversos clientes de navegadores Web conformes ao protocolo HTTP. Nesta subseção, examinaremos primeiro a autoescalabilidade de arquiteturas P2P no contexto de distribuição de arquivos. Então, descreveremos o BitTorrent em um certo nível de detalhes, destacando suas características mais importantes.

Escalabilidade de arquiteturas P2P

Para comparar arquiteturas cliente-servidor com arquiteturas P2P, e ilustrar a inerente autoescalabilidade de P2P, consideraremos um modelo quantitativo simples para a distribuição de um arquivo para um conjunto fixo de pares para ambos os tipos de arquitetura. Conforme demonstrado na Figura 2.24, o servidor e os pares são conectados por enlaces de acesso da Internet. A taxa de upload do enlace de acesso do servidor é denotada por u_s , e a taxa de upload do enlace de acesso do par i é denotada por u_i , e a taxa de download do enlace de acesso do par i é denotada por d_i . O tamanho do arquivo a ser distribuído (em bits) é denotado por F e o número de pares que querem obter uma cópia do arquivo, por N . O tempo de distribuição é o tempo necessário para que todos os N pares obtenham uma cópia do arquivo. Em nossa análise do tempo de distribuição abaixo, tanto para a arquitetura cliente-servidor como para a arquitetura P2P, fazemos a hipótese simplificada (e geralmente precisa [Akella, 2003]) de que o núcleo da Internet tem largura de banda abundante, o que implica que todos os

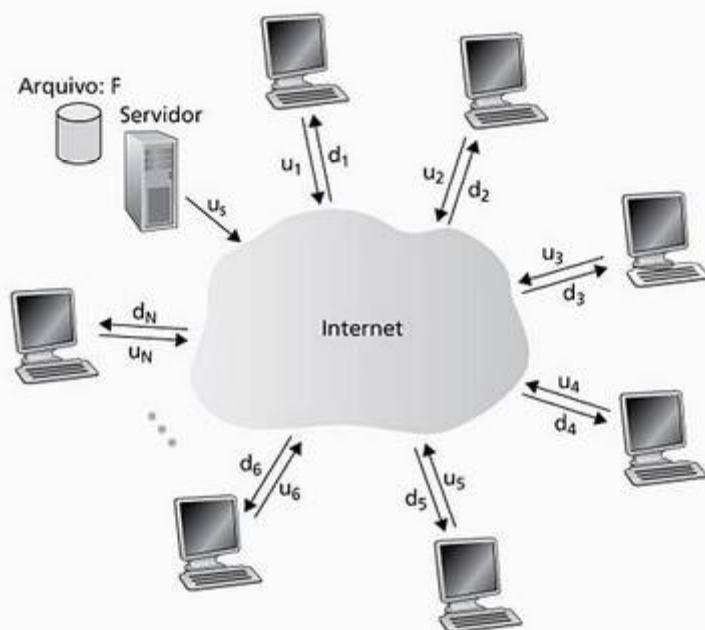


Figura 2.24 Um problema ilustrativo de distribuição de arquivo

gargalos encontram-se no acesso à rede. Suponha também que o servidor e os clientes não participam de nenhuma outra aplicação de rede, para que toda sua largura de banda de acesso de upload e download possa ser totalmente devotada à distribuição do arquivo.

Determinemos primeiro o tempo de distribuição para a arquitetura cliente-servidor, que denotaremos por D_{cs} . Na arquitetura cliente-servidor, nenhum dos pares auxilia na distribuição do arquivo. Fazemos as observações abaixo:

O servidor deve transmitir uma cópia do arquivo a cada um dos N pares. Assim, o servidor deve transmitir NF bits. Como a taxa de upload do servidor é de u_s , o tempo para distribuição do arquivo deve ser de pelo menos NF/u_s .

Deixemos que d_{\min} denote a taxa de download do par com menor taxa de download, ou seja, $d_{\min} = \min\{d_1, d_p, \dots, d_n\}$. O par com a menor taxa de download não pode obter todos os bits F do arquivo em menos de F/d_{\min} segundos. Assim, o tempo de distribuição mínimo é de pelo menos F/d_{\min} .

Reunindo essas observações, temos:

$$D_{cs} \geq \max \left\{ \frac{NF}{u_s}, \frac{F}{d_{\min}} \right\}$$

Isso proporciona um limite inferior para o tempo mínimo de distribuição para a arquitetura cliente-servidor. Nos problemas do final do capítulo, você deverá demonstrar que o servidor pode programar suas transmissões de forma que o limite inferior seja sempre alcançado. Portanto, consideraremos esse limite inferior fornecido anteriormente como o tempo real de distribuição, ou seja,

$$D_{cs} = \max \left\{ \frac{NF}{u_s}, \frac{F}{d_{\min}} \right\} \quad (2.1)$$

Vemos, a partir da Equação 2.1, que para N grande o suficiente, o tempo de distribuição cliente-servidor é dado por NF/u_s . Assim, o tempo de distribuição aumenta linearmente com o número de pares N . Portanto, por exemplo, se o número de pares de uma semana para a outra for multiplicado por mil, de mil para um milhão, o tempo necessário para distribuir o arquivo para todos os pares aumentará mil vezes.

Passemos agora para uma análise semelhante para a arquitetura P2P, em que cada par pode auxiliar o servidor na distribuição do arquivo. Em particular, quando um par recebe alguns dados do arquivo, ele pode usar sua própria capacidade de upload para redistribuir os dados a outros pares. Calcular o tempo de distribuição para a arquitetura P2P é, de certa forma, mais complicado do que para a arquitetura cliente-servidor, visto que o tempo de distribuição depende de como cada par distribui parcelas do arquivo aos outros pares. Não obstante, uma simples expressão para o tempo mínimo de distribuição pode ser obtida [Kumar, 2006]. Para essa finalidade, faremos as observações a seguir:

No início da distribuição, apenas o servidor tem o arquivo. Para levar esse arquivo à comunidade de pares, o servidor deve enviar cada bit do arquivo pelo menos uma vez para seu enlace de acesso. Assim, o tempo de distribuição mínimo é de pelo menos F/u_s . (Diferente do esquema cliente-servidor, um bit enviado uma vez pelo servidor pode não precisar ser enviado novamente, visto que os pares podem redistribuir entre si esse bit).

Assim como na arquitetura cliente-servidor, o par com a menor taxa de download não pode obter todos os bits F do arquivo em menos de F/d_{\min} segundos. Assim, o tempo mínimo de distribuição é de pelo menos F/d_{\min} .

Finalmente, observemos que a capacidade de upload total do sistema como um todo é igual à taxa de upload do servidor mais as taxas de upload de cada um dos pares individuais, ou seja, $u_{\text{total}} = u_s + u_1 + \dots + u_n$. O sistema deve entregar (fazer o upload de) F bits para cada um dos N pares, entregando assim um total de NF bits. Isso não pode ser feito em uma taxa mais rápida do que u_{total} . Assim, o tempo mínimo de distribuição é também de pelo menos $NF/(u_s + u_1 + \dots + u_n)$.

Juntando essas três observações, obtemos o tempo mínimo de distribuição para P2P, denotado por D_{P2P} .

$$D_{P2P} \geq \max \left\{ \frac{F}{u_s}, \frac{F}{d_{\min}}, \frac{NF}{u_s + \sum_{i=1}^N u_i} \right\} \quad (2.2)$$

A Equação 2.2 fornece um limite inferior para o tempo mínimo de distribuição para a arquitetura P2P. Ocorre que, se imaginarmos que cada par pode redistribuir um bit assim que o recebe, há um esquema de redistribuição que, de fato, alcança esse limite inferior [Kumar, 2006] (Provaremos um caso especial desse resultado na lição de casa). Na realidade, quando blocos do arquivo são redistribuídos, em vez de bits individuais, a Equação 2.2 serve como uma boa aproximação do tempo mínimo real de distribuição. Assim, peguemos o limite inferior fornecido pela Equação 2.2 como o tempo mínimo real de distribuição, que é:

$$D_{P2P} = \max \left\{ \frac{F}{u_s}, \frac{F}{d_{\min}}, \frac{NF}{u_s + \sum_{i=1}^N u_i} \right\} \quad (2.3)$$

A Figura 2.25 compara o tempo mínimo de distribuição para as arquiteturas cliente-servidor e P2P, pressupondo que todos os pares têm a mesma taxa de upload u . Na Figura 2.25, temos definido que $F/u = 1$ hora, $u_s = 10u$ e $d_{\min} \geq u_s$. Assim, um par pode transmitir todo o arquivo em uma hora, sendo a taxa de transmissão do servidor 10 vezes a taxa de upload do par, e (para simplicidade) as taxas de download de par são definidas grandes o suficiente de forma a não ter efeito. Vemos na Figura 2.25 que, para a arquitetura cliente-servidor, o tempo de distribuição aumenta linearmente e sem limite, conforme aumenta o número de pares. No entanto, para a arquitetura P2P, o tempo mínimo de distribuição não é apenas sempre menor do que o tempo de distribuição da arquitetura cliente-servidor; é também de menos do que uma hora para qualquer número de pares N . Assim, aplicações com a arquitetura P2P podem ter autoescalabilidade. Essa escalabilidade é uma consequência direta de pares sendo redistribuidores, bem como consumidores de bits.

BitTorrent

O BitTorrent é um protocolo P2P popular para distribuição de arquivos [BitTorrent, 2009]. No jargão do BitTorrent, a coleção de todos os pares que participam da distribuição de um determinado arquivo é chamada de *torrent*. Os pares em um torrent fazem o download de *blocos* de tamanho igual do arquivo entre si, com um

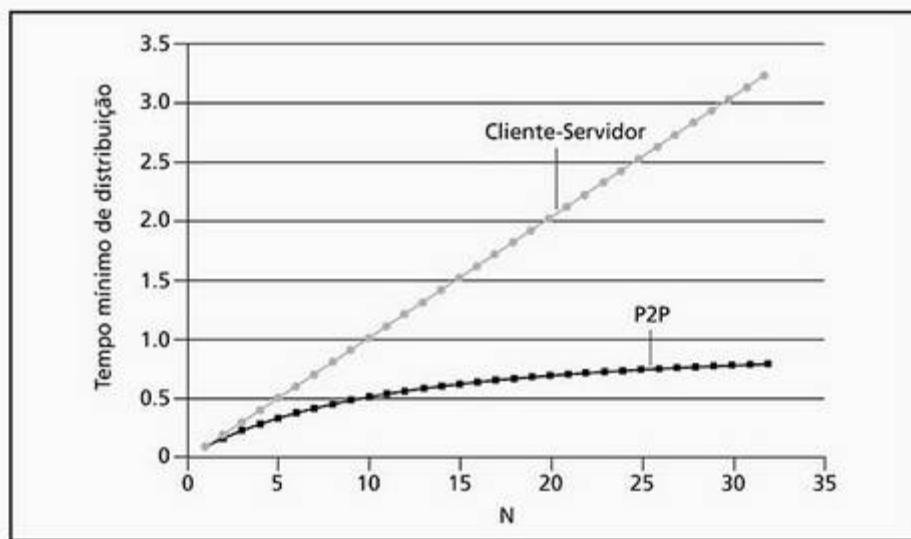


Figura 2.25 Tempo de distribuição para arquiteturas P2P e cliente-servidor

tamanho típico de bloco de 256 KBytes. Quando um par entra em um torrent, ele não tem nenhum bloco. Com o tempo, ele acumula mais blocos. Enquanto ele faz o download de blocos, faz também uploads de blocos para outros pares. Uma vez que um par adquire todo o arquivo, ele pode (de forma egoísta) sair do torrent ou (de forma altruista) permanecer no torrent e continuar fazendo o upload de blocos a outros pares. Além disso, qualquer par pode sair do torrent a qualquer momento com apenas um subconjunto de blocos, e depois voltar.

Observemos agora, mais atentamente, como opera o BitTorrent. Como o BitTorrent é um protocolo e sistema complicado, descreveremos apenas seus mecanismos mais importantes, ignorando alguns detalhes; isso nos permitirá ver a floresta através das árvores. Cada torrent tem um nó de infraestrutura chamado rastreador. Quando um par chega em um torrent, ele se registra com o rastreador e periodicamente informa ao rastreador que ainda está no torrent. Dessa forma, o rastreador mantém um registro dos pares que participam do torrent. Um determinado torrent pode ter menos de dez ou mais de mil pares participando a qualquer momento.

Como demonstrado na Figura 2.26, quando um novo par, Alice, chega no torrent, o rastreador seleciona aleatoriamente um subconjunto de pares (para dados concretos, digamos que sejam 50) do conjunto de pares participantes, e envia os endereços IP desses 50 pares para Alice. Com a lista de pares, Alice tenta estabelecer conexões TCP simultâneas com todos os pares da lista. Chamaremos todos os pares com quem Alice consiga estabelecer uma conexão TCP de “pares vizinhos” (na Figura 2.26, Alice é representada com apenas três pares vizinhos. Normalmente, ela teria muito mais). Com o tempo, alguns desses pares podem sair e outros pares (fora dos 50 iniciais) podem tentar estabelecer conexões TCP com Alice. Portanto, os pares vizinhos de um par podem flutuar com o tempo.

A qualquer momento, cada par terá um subconjunto de blocos do arquivo, com pares diferentes com subconjuntos diferentes. Periodicamente, Alice pedirá a cada um de seus pares vizinhos (nas conexões TCP) a lista de quais blocos eles têm. Caso Alice tenha L vizinhos diferentes, ela obterá L listas de blocos. Com essa informação, Alice emitirá solicitações (novamente, nas conexões TCP) de blocos que ela não tem.

Portanto, a qualquer momento, Alice terá um subconjunto de blocos e saberá quais blocos seus vizinhos têm. Com essa informação, Alice terá duas decisões importantes a fazer. Primeiro, quais blocos ela deve solicitar primeiro de seus vizinhos, e segundo, a quais vizinhos ela deve enviar os blocos solicitados. Ao decidir quais

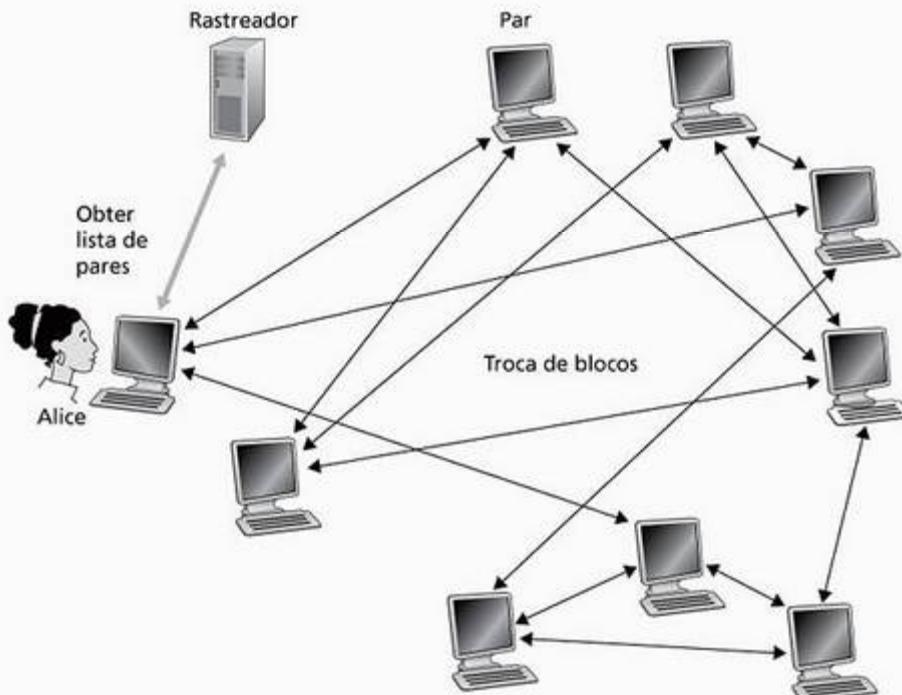


Figura 2.26 Distribuição de arquivos com o BitTorrent

blocos solicitar, Alice usa uma técnica chamada *rarest first* (o mais raro primeiro). A ideia é determinar, dentre os blocos que ela não tem, quais são os mais raros dentre seus vizinhos (ou seja, os blocos que têm o menor número de cópias repetidas em seus vizinhos) e então solicitar esses blocos mais raros primeiro. Dessa forma, os blocos mais raros são redistribuídos mais rapidamente, procurando (*grosso modo*) equalizar os números de cópias de cada bloco no torrent.

Para determinar a quais pedidos atender, o BitTorrent usa um algoritmo de troca inteligente. A ideia básica é Alice dar prioridade aos vizinhos que atualmente fornecem seus dados com a maior taxa. Especificamente, para cada um de seus vizinhos, Alice mede continuamente a taxa em que recebe bits e determina os quatro pares que lhe fornecem na melhor taxa. Então, ela reciprocamente envia blocos a esses mesmos quatro pares. A cada 10 segundos, ela recalcula as taxas e possivelmente modifica o conjunto de quatro pares. No jargão do BitTorrent, esses quatro pares são chamados de *unchoked* (não sufocado). É importante informar que, a cada 30 segundos, ela também escolhe um vizinho adicional aleatoriamente e envia blocos para ele. Chamaremos o vizinho escolhido aleatoriamente de Bob. No jargão de BitTorrent, Bob é chamado de *optimistically unchoked*. Como Alice envia dados a Bob, ela pode se tornar um dos quatro melhores transmissores para Bob, caso em que Bob começaria a enviar dados para Alice. Caso a taxa em que Bob envie dados a Alice seja alta o suficiente, Bob pode, em troca, tornar-se um dos quatro melhores transmissores para Alice. Em outras palavras, Alice aleatoriamente escolherá um novo parceiro de troca e a começará com ele. Caso os dois pares estejam satisfeitos com a troca, eles colocarão um ao outro nas suas listas de quatro melhores pares e continuarão a troca até que um dos pares encontre um parceiro melhor. O efeito é que pares capazes de fazer uploads em taxas compatíveis tendem a se encontrar. A seleção aleatória de vizinho também permite que novos pares obtenham blocos, de forma que possam ter algo para trocar. Todos os pares vizinhos, além desses cinco pares (quatro pares "top" e um em experiência) estão "sufocados", ou seja, não recebem nenhum bloco de Alice. O BitTorrent tem diversos mecanismos interessantes não discutidos aqui, incluindo pedaços (miniblocos), pipelining (tubulação), primeira seleção aleatória, modo *endgame* (fim de jogo) e *anti-snubbing* (antirrejeição) [Cohen 2003].

O mecanismo de incentivo para troca descrito acima é normalmente chamado de *tit-for-tat* (olho por olho) [Chen 2003]. Demonstrou-se que esse esquema de incentivo pode ser burlado [Liogkas, 2006; Locher, 2006; Piatek, 2007]. Não obstante, o ecossistema do BitTorrent é muito bem-sucedido, com milhões de pares simultâneos compartilhando arquivos ativamente em centenas de milhares de torrents. Caso o BitTorrent tivesse sido projetado sem o tit-for-tat (ou uma variante), mas com o restante da mesma maneira, ele provavelmente nem existisse mais, visto que a maioria dos usuários são pessoas que apenas querem obter as coisas gratuitamente [Saroui, 2002].

Variantes interessantes do protocolo BitTorrent são propostas [Guo, 2005; Piatek, 2007]. Além disso, muitas das aplicações de transmissão em tempo real P2P, como PPLive e ppstream, foram inspirados pelo BitTorrent [Hei 2007].

2.6.2 Distributed Hash Tables (DHTs)

Um componente crítico de muitas aplicações P2P e outras aplicações distribuídas é um índice (ou seja, um banco de dados simples), que suporta operações de busca e atualização. Quando esse banco de dados é distribuído, os pares podem realizar caching de conteúdo e roteamento sofisticado de consultas entre si. Como a indexação e busca de informações são um componente crítico nesses sistemas, cobriremos uma técnica popular de indexação e busca, Distributed Hash Tables (DHTs).

Consideremos assim a construção de um banco de dados simples distribuído em um grande número (possivelmente milhões) de pares que suportam indexação e solicitação simples. As informações armazenadas em nosso banco de dados consistirão de duplas (chave, valor). Por exemplo, as chaves podem ser números de segurança social e os valores podem ser nomes humanos correspondentes; nesse caso, um exemplo de dupla chave-valor é (156-45-7081, Johnny Wu). Ou as duplas podem ser nomes de conteúdo (ex.: nomes de filmes, álbuns e software), e os valores podem ser endereços IP onde o conteúdo está armazenado; nesse caso, um exemplo de par chave-valor é (Led Zeppelin IV, 203.17.123.38). Pares consultam nossos bancos de dados fornecendo a chave: caso haja duplas (chave, valor) em seus bancos de dados que correspondam à chave, o banco de dados retorna

as duplas correspondentes ao par solicitante. Portanto, por exemplo, caso o banco de dados armazene números de segurança social e seus nomes humanos correspondentes, um par pode consultar um número de segurança social específico e o banco de dados retornará o nome do humano que possui aquele número de segurança social. Pares também devem ser capazes de inserir duplas (chave, valor) em nosso banco de dados.

A construção desse banco de dados é direta com uma arquitetura cliente-servidor na qual todos os pares (chave, valor) são armazenados em um servidor central. Essa abordagem centralizada também foi considerada em antigos sistemas P2P como o Napster. Mas o problema é significativamente mais desafiador e interessante em um sistema distribuído que consiste de milhões de pares conectados sem autoridade central. Em um sistema P2P, queremos distribuir as duplas (chave, valor) entre todos os pares, de forma que cada par tenha apenas um pequeno subconjunto da totalidade dos pares (chave, valor). Uma abordagem inocente para a construção desse banco de dados P2P é (1) espalhar aleatoriamente as duplas (chave, valor) entre os pares e (2) fazer com que cada par tenha uma lista dos endereços de IP de todos os pares participantes. Dessa forma, o par solicitante pode enviar uma solicitação a todos os outros pares, e os pares que tenham duplas (chave, valor) que correspondam à chave podem responder com as duplas correspondentes. Essa abordagem é completamente não escalável, logicamente, visto que cada par precisaria rastrear todos os outros pares (possivelmente milhões) e, pior, enviar cada solicitação a *todos* os pares.

Agora descreveremos uma abordagem elegante para um projeto de banco de dados P2P. Para isso, primeiro designaremos um identificador a cada par, em que cada identificador é um número inteiro na faixa $[0, 2^n - 1]$ de algum n fixo. Observe que cada identificador pode ser expresso por uma representação com n -bits. Vamos também exigir que cada chave seja um número inteiro na mesma faixa. O leitor astuto pode ter observado que as chaves de exemplo descritas anteriormente não são (números de segurança social e nomes de conteúdo) números inteiros. Para criar números inteiros a partir dessas chaves, precisaremos usar uma função hash que mapeie cada chave (por exemplo, número de segurança social) em um número inteiro na faixa $[0, 2^n - 1]$. Uma função de hash é uma função de muitos-para-um para a qual duas entradas diferentes podem ter a mesma saída (mesmo número inteiro), mas a probabilidade de terem a mesma saída é extremamente pequena (leitores não familiarizados com funções de hash podem querer consultar o Capítulo 7, que discute detalhadamente funções de hash). A função de hash é considerada publicamente disponível a todos os pares no sistema. Portanto, quando nos referirmos à "chave", nos referimos ao hash da chave original. Portanto, por exemplo, caso a chave original seja "Led Zeppelin IV", a chave será o número inteiro que corresponda ao hash de "Led Zeppelin IV". Além disso, como estamos usando hashes de chaves, em vez das próprias chaves, nos referiremos ao banco de dados distribuído como **Distributed Hash Table (DHT)**.

Consideraremos agora o problema de armazenar as duplas (chave, valor) no DHT. A questão central aqui é definir uma regra para designar chaves a pares. Considerando que cada par tenha um identificador de número inteiro e cada chave também seja um número inteiro na mesma faixa, uma abordagem natural é designar cada dupla (chave, valor) ao par cujo identificador está *mais próximo* da chave. Para implementar esse esquema, precisaremos definir o que significa 'mais próximo', o que admite muitas convenções. Por conveniência, definiremos que o par mais próximo é o *sucessor imediato da chave*. Para visualizarmos, observaremos um exemplo específico. Suponha que $n = 4$, portanto, todos os identificadores de par e chave estarão na faixa de $[0, 15]$. Suponha ainda que haja oito pares no sistema com identificadores 1, 3, 4, 5, 8, 10, 12 e 15. Finalmente, suponha que queiramos armazenar o par chave-valor (11, Johnny Wu) em um dos oito pares. Mas em qual? Usando nossa convenção de mais próximo, como o par 12 é o sucessor imediato da chave 11, armazenaremos, portanto, a dupla (11, Johnny Wu) no par 12 [para concluir nossa definição de mais próximo, caso a chave seja exatamente igual a um dos identificadores do par, armazenaremos o par (chave-valor) em um par correspondente; e caso a chave seja maior do que todos os identificadores de par, usaremos uma convenção módulo- 2^n , que armazena o par (chave-valor) no par com o menor identificador].

Suponha agora que um par, Alice, queira inserir uma dupla (chave-valor) no DHT. Conceitualmente, é um processo objetivo: ela primeiro determina o par cujo identificador é o mais próximo da chave; então ela envia uma mensagem a esse par, instruindo-o a armazenar a dupla (chave, valor). Mas como Alice determina o par mais próximo da chave? Se Alice rastreasse todos os pares no sistema (IDs de par e endereços IP correspondentes), ela poderia

determinar localmente o par mais próximo. Mas essa abordagem requer que *cada* par rastreie *todos* os outros pares no DHT — o que é completamente impraticável para um sistema de grande escala com milhões de pares.

DHT circular

Para abordar esse problema, consideraremos agora organizar os pares em um círculo. Nessa disposição circular, cada par rastreia apenas seu sucessor imediato ($módulo\ 2^n$). Um exemplo desse círculo é exibido na Figura 2.27(a). Nesse exemplo, n é novamente 4 e há os mesmos oito pares do exemplo anterior. Cada par está ciente apenas de seu sucessor imediato; por exemplo, o par 5 sabe o endereço IP e o identificador do par 8, mas não sabe necessariamente nada sobre quaisquer outros pares no DHT. Essa disposição circular dos pares é um caso especial de uma rede sobreposta. Em uma rede sobreposta, os pares formam uma rede lógica abstrata que reside acima da rede de computadores “inferior” que consiste de enlaces físicos, roteadores e hospedeiros. Os enlaces em uma rede sobreposta não são físicos, mas enlaces virtuais entre duplas de pares. Na rede de sobreposição da Figura 2.27(a), há oito pares e oito enlaces sobrepostos; na sobreposição da Figura 2.27(b) há oito pares e 16 enlaces sobrepostos. Um único enlace de sobreposição normalmente usa muitas ligações físicas e roteadores físicos na rede inferior.

Usando a rede de sobreposição circular da Figura 2.27(a), suponha agora que o par 3 deseja determinar qual par no DHT é responsável pela chave 11 [para inserir ou para requisitar uma dupla (chave-valor)]. Usando a rede sobreposta circular, o par de origem (par 3) cria uma mensagem que pergunta “Quem é responsável pela chave 11?” e a envia a seu sucessor, o par 4. Sempre que um par recebe essa mensagem, como sabe o identificador de seu sucessor, pode determinar se é responsável (ou seja, mais próximo) pela chave em questão. Caso um par não seja responsável pela chave, ele simplesmente envia a mensagem a seu sucessor. Portanto, por exemplo, quando o par 4 recebe a mensagem perguntando sobre a chave 11, ele determina que não é responsável pela chave (porque seu sucessor está mais perto da dela), portanto, ele passa a mensagem a seu sucessor, ou seja, o par 5. Esse processo continua até que a mensagem chegue ao par 12, que determina que é o mais próximo da chave 11. A essa altura, o par 12 pode enviar uma mensagem de volta à origem, o par 3, indicando que é responsável pela chave 11.

O DHT circular oferece uma solução bastante elegante para reduzir a quantidade de informação sobreposta que cada par deve gerenciar. Em particular, cada par está ciente apenas de dois pares, seu sucessor imediato e seu predecessor imediato (por padrão, o par está ciente de seu predecessor, visto que este lhe envia mensagens). Porém, essa solução ainda introduz um novo problema. Embora cada par esteja ciente de dois pares vizinhos, para encontrar o nó responsável por uma chave (no pior das hipóteses), todos os N nós no DHT deverão encaminhar uma mensagem pelo círculo; $N/2$ mensagens são enviadas em média.

Assim, no projeto de um DHT, há uma troca entre o número de vizinhos que cada par tem de rastrear e o número de mensagens que o DHT precisa enviar para resolver uma única solicitação. Por um lado, se cada par

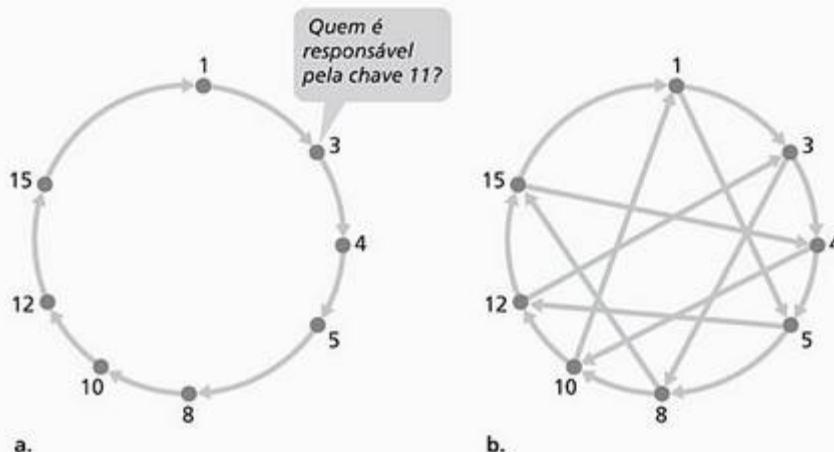


Figura 2.27 (a) Um DHT circular. O par 3 quer determinar quem é responsável pela chave 11. (b) Um DHT circular com atalhos

rastrear todos os outros pares (sobreposições de malha), apenas uma mensagem será enviada por solicitação, mas cada par deverá rastrear N pares. Por outro lado, com um DHT circular, cada par está ciente apenas de dois pares, mas $N/2$ mensagens são enviadas em média para cada solicitação. Felizmente, podemos refinar nossos projetos de DHTs de forma que o número de vizinhos por par, bem como o número de mensagens por solicitação seja mantido em um tamanho aceitável. Um desses refinamentos é usar a rede sobreposta circular como fundação, mas adicionar “atalhos” de forma que cada par não apenas rastreie seu sucessor imediato, mas também um número relativamente pequeno de pares espalhados pelo círculo. Um exemplo desse DHT circular com alguns atalhos é demonstrado na Figura 2.27(b). Atalhos são usados para expedir o roteamento das mensagens de solicitação. Especificamente, quando um par recebe uma mensagem que solicita uma chave, ele encaminha a mensagem ao vizinho (vizinho sucessor ou um dos vizinhos via atalho) que está mais perto da chave. Assim, na Figura 2.27(b), quando o par 4 recebe a mensagem solicitando a chave 11, ele determina que o par mais próximo (entre seus vizinhos) é seu vizinho no atalho 10 e então envia a mensagem diretamente ao par 10. Claramente, atalhos podem reduzir significativamente o número de mensagens usado para processar uma solicitação.

A próxima questão natural é “Quantos vizinhos no atalho cada par deve ter, e quais pares devem ser esses vizinhos de atalho?” Essa pergunta recebeu atenção significativa da comunidade de pesquisa [Stoica, 2001; Rowstron, 2001; Ratnasamy, 2001; Zhao, 2004; Maymounkov, 2002; Garces-Erce, 2003]. De forma importante, demonstrou-se que o DHT pode ser projetado de forma que tanto o número de vizinhos como o número de mensagens por solicitação seja da ordem de $\log N$, em que N é o número de pares. Esses projetos obtêm um compromisso satisfatório entre as soluções extremas de se usar topologias de sobreposição circular e de malha.

Peer churn

Em sistemas P2P, um par pode vir ou ir sem aviso. Assim, no projeto de um DHT, devemos nos preocupar em manter a sobreposição de DHT na presença desse peer churn. Para termos uma compreensão abrangente de como isso pode ser realizado, consideraremos mais uma vez o DHT circular da Figura 2.27(a). Para manejar o peer churn, exigiremos que cada par rastreie (ou seja, saiba o endereço IP de) seu primeiro e segundo sucessores; por exemplo, o par 4 agora rastreia tanto o par 5 como o par 8. Exigiremos também que cada par verifique periodicamente se seus dois sucessores estão vivos (por exemplo, enviando periodicamente mensagens de ping e pedindo respostas). Consideraremos agora como o DHT é mantido quando um par sai abruptamente. Por exemplo, suponha que o par 5 da Figura 2.27(a) saia abruptamente. Nesse caso, os dois pares precedentes ao par que saiu (4 e 3) saberão que o par saiu, pois não responde mais às mensagens de ping. Os pares 4 e 3 precisam, portanto, atualizar as informações do estado de seu sucessor. Consideraremos agora como o par 4 atualiza seu estado:

1. O par 4 substitui seu primeiro sucessor (par 5) por seu segundo sucessor (par 8).
2. O par 4, então, pergunta a seu novo primeiro sucessor (par 8) o identificador e o endereço IP de seu sucessor imediato (par 10). O par 4, então, torna o par 10 seu segundo sucessor.

Nos problemas, você deverá determinar como o par 3 atualiza suas informações de determinação de roteamento de sobreposição.

Tendo abordado brevemente o que deve ser feito quando um par sai, consideraremos agora o que acontece quando um par quer entrar no DHT. Digamos que um par com identificador 13 quer entrar no DHT, e quando entra, sabe apenas da existência do par 1 no DHT. O par 13 primeiro envia ao par 1 uma mensagem, perguntando “quem serão o predecessor e o sucessor do par 13?”. Essa mensagem é encaminhada através do DHT até alcançar o par 12, que percebe que será o predecessor do par 13, e que seu sucessor, o par 15, será o sucessor do par 13. Em seguida, o par 12 envia as informações de sucessor e predecessor ao par 13. O par 13, então, pode entrar no DHT, tornando o par 15 seu sucessor e notificando ao par 12 que deve mudar seu sucessor imediato para 13.

DHTs têm amplo uso na prática. Por exemplo, o BitTorrent usa o DHT Kademia para criar um rastreador distribuído. No BitTorrent, a chave é o identificador do torrent e o valor é o endereço IP dos pares que atualmente participam dos torrents [Falkner, 2007; Neglia, 2007]; Dessa forma, solicitando ao DHT um identificador de torrent, um par de BitTorrent recém-chegado pode determinar o par responsável pelo identificador (ou seja, por determinar os pares no torrent). Após ter encontrado esse par, o par recém-chegado pode solicitar dele uma lista

de outros pares no torrent. DHTs são usados extensamente no sistema de compartilhamento de arquivos eMule para localizar conteúdos em pares [Liang, 2006].

2.6.3 Estudo de caso: telefonia por Internet P2P com Skype

O Skype é uma aplicação P2P imensamente popular, muitas vezes com sete ou oito milhões de usuários conectados simultaneamente. Além de fornecer serviço de telefonia PC para PC na Internet, o Skype oferece serviço de telefonia PC-para-telefone, telefone-para-PC e serviço de videoconferência PC-a-PC. Fundado pelos mesmos indivíduos que criaram o FastTrack e o Kazaa, o Skype foi adquirido pelo eBay em 2005 por US\$ 2,6 bilhões.

O Skype usa técnicas P2P de diversas maneiras inovadoras, ilustrando de uma boa maneira como o P2P pode ser usado em aplicações que vão além da distribuição de conteúdo e compartilhamento de arquivos. Assim como com programas de mensagens instantâneas, a telefonia PC-para-PC na Internet é inherentemente P2P, visto que, no núcleo do aplicativo, duplas de usuários (ou seja, pares) comunicam-se entre si em tempo real. Porém, o Skype também emprega técnicas P2P para duas outras funções importantes, que são localização de usuário e NAT traversal.

Os protocolos do Skype não são apenas proprietários, como todas as transmissões de pacotes do Skype (pacotes de controle e voz) são criptografadas. Não obstante, a partir do website do Skype e de diversos estudos de medição, os pesquisadores descobriram como o Skype geralmente funciona [Baset, 2006; Guha, 2006; Chen, 2006; Suh, 2006 Ren, 2006]. Assim como com o FastTrack, os nós no Skype são organizados em uma rede sobreposta hierárquica, com cada par classificado como superpar ou par comum. O Skype inclui um índice que mapeia os nomes de usuários do Skype a endereços IP atuais (e números de porta). Esse índice é distribuído entre os superpares. Quando Alice deseja telefonar para Bob, seu cliente Skype procura o índice distribuído para determinar o endereço IP atual de Bob. Como o protocolo do Skype é proprietário, atualmente não está claro como os mapeamentos de índice são organizados nos superpares, embora alguma forma de organização DHT seja muito possível.

Técnicas de P2P também são usadas em retransmissores do Skype, que são úteis para estabelecer chamadas entre hospedeiros em redes domésticas. Muitas configurações de redes domésticas fornecem acesso à Internet através de um roteador (tipicamente um roteador sem fio). Esses roteadores são, na verdade, mais do que roteadores, e normalmente incluem um elemento chamado Network Address Translator [Tradutor de Endereço da Internet] (NAT). Estudaremos NATs no Capítulo 4. Por enquanto, tudo do que precisamos saber é que um NAT impede que um hospedeiro de fora da rede doméstica inicie uma conexão com um hospedeiro dentro da rede doméstica. Caso ambos os chamadores do Skype tenham NATs, há um problema — nenhum deles pode aceitar uma chamada iniciada pelo outro, tornando aparentemente impossível uma chamada. O uso inteligente de superpares e retransmissores resolve muito bem esse problema. Suponha que quando Alice entra, ela recebe um superpar sem NAT. Alice pode iniciar uma sessão com seu superpar, visto que seu NAT apenas impede sessões iniciadas de fora de sua rede doméstica. Isso permite que Alice e seu superpar troquem mensagens de controle nessa sessão. O mesmo ocorre com Bob quando ele entra. Agora, quando Alice deseja telefonar para Bob, ela informa a seu superpar, que, por sua vez, informa ao superpar de Bob, que então informa a Bob sobre a chamada recebida de Alice. Caso Bob aceite a chamada, os dois superpares selecionam um terceiro superpar sem NAT — o nó de retransmissor — cujo trabalho será o de transmitir os dados entre Alice e Bob. Os superpares de Alice e de Bob, então, instruem Alice e Bob, respectivamente, a iniciarem uma sessão com o retransmissor. Alice, então, envia pacotes de voz ao retransmissor na conexão Alice-para-retransmissor (que foi iniciada por Alice), e o retransmissor transmite esses pacotes pela conexão retransmissor-para-Bob (que foi iniciada por Bob); pacotes de Bob para Alice fluem pelas mesmas duas conexões de retransmissor de forma inversa. E pronto, Bob e Alice têm uma conexão simultânea fim a fim mesmo que nenhum deles possa aceitar uma sessão originada fora de sua LAN. O uso de retransmissor ilustra o projeto cada vez mais sofisticado de sistemas P2P, em que pares realizam serviços de sistema central para outros (serviço de indexação e transmissão sendo dois exemplos) ao mesmo tempo em que usam o serviço de usuário final (por exemplo, download de arquivo, telefonia IP) fornecidos pelo sistema P2P.

O Skype é uma aplicação da Internet de amplo sucesso, que atinge literalmente dezenas de milhões de usuários. A incrivelmente rápida e ampla adoção de Skype, bem como de compartilhamento de arquivos

P2P, da Web e de programas de mensagens instantâneas antes dele, é um testamento da sabedoria do projeto de arquitetura geral da Internet, um projeto que não poderia ter previsto o conjunto rico e sempre em expansão de aplicações da Internet que seria desenvolvido nos próximos 30 anos. Os serviços de rede oferecidos a aplicações de Internet — transporte de datagrama sem conexão (UDP), transferência de datagrama orientado para conexão (TCP), a interface socket, endereçamento e nomes (DNS) entre outros — provaram ser suficientes para permitir que milhares de aplicações fossem desenvolvidas. Como todas essas aplicações foram colocadas sobre as quatro camadas inferiores existentes da pilha de protocolo de Internet, eles envolvem apenas o desenvolvimento de novos softwares cliente-servidor e peer-to-peer para uso nos sistemas finais. Isso, por sua vez, permitiu que essas aplicações fossem rapidamente empregadas e adotadas.

2.7 Programação e desenvolvimento de aplicações com TCP

Agora que já examinamos várias importantes aplicações de rede, vamos explorar como são escritos programas de aplicação de rede. Nesta seção escreveremos programas de aplicações que usam TCP; na seção seguinte, escreveremos programas que usam UDP.

Lembre-se de que na Seção 2.1 dissemos que muitas aplicações de rede consistem em um par de programas — um programa cliente e um programa servidor — que residem em dois sistemas finais diferentes. Quando esses programas são executados, criam-se um processo cliente e um processo servidor, que se comunicam entre si lendo de seus sockets e escrevendo através deles. Ao criar uma aplicação de rede, a tarefa principal do programador é escrever o código tanto para o programa cliente como para o programa servidor.

Há dois tipos de aplicações de rede. Um deles é uma implementação de um protocolo padrão definido, por exemplo, em um RFC. Para essa implementação, os programas, cliente e servidor, devem obedecer às regras ditas pelo RFC. Por exemplo, o programa cliente poderia ser uma implementação do lado do cliente do protocolo FTP descrito na Seção 2.3 e definido explicitamente no RFC 959 e o programa servidor, uma implementação do protocolo de servidor FTP também descrito explicitamente no RFC 959. Se um programador escrever codificação para o programa cliente e um outro programador independente escrever uma codificação para o programa servidor e ambos seguirem cuidadosamente as regras do RFC, então os dois programas poderão interagir. Realmente, muitas das aplicações de rede de hoje envolvem comunicação entre programas cliente e servidor que foram criados por programadores diferentes — por exemplo, um browser Netscape que se comunica com um servidor Web Apache, ou um cliente FTP em um PC que carrega um arquivo em um servidor FTP UNIX. Quando um programa, cliente ou servidor, implementa um protocolo definido em um RFC, deve usar o número de porta associado com o protocolo. (Números de portas foram discutidos brevemente na Seção 2.1. Serão examinados mais detalhadamente no Capítulo 3.)

O outro tipo de aplicação cliente-servidor é uma aplicação *proprietária*. Nesse caso, o protocolo de camada de aplicação utilizado pelos programas cliente e servidor não obedecem necessariamente a nenhum RFC existente. Um único programador (ou equipe de desenvolvimento) cria ambos os programas cliente e servidor, e tem completo controle sobre o que entra no código. Mas, como a codificação não implementa um protocolo de domínio público, outros programadores independentes não poderão desenvolver programas que interajam com a aplicação. Ao desenvolver uma aplicação proprietária, o programador deve ter o cuidado de não usar um dos números de porta bem conhecidos, definidos em RFCs.

Nesta seção e na próxima, examinaremos as questões fundamentais do desenvolvimento de uma aplicação cliente-servidor proprietária. Durante a fase de desenvolvimento, uma das primeiras decisões que o programador deve tomar é se a aplicação rodará em TCP ou UDP. Lembre-se de que o TCP é orientado para conexão e provê um *canal confiável de cadeia de bytes*, pelo qual fluem dados entre dois sistemas finais. O UDP não é orientado para conexão e envia pacotes de dados independentes de um sistema final ao outro, sem nenhuma garantia de entrega.

Nesta seção, desenvolveremos uma aplicação cliente simples que roda em TCP; na seção seguinte, desenvolveremos uma aplicação cliente simples que roda em UDP. Apresentaremos essas aplicações TCP e UDP simples em Java. Poderíamos escrevê-las em linguagem C ou C++, mas optamos por Java por diversas razões. Em primeiro

lugar, o código das aplicações é mais elegante e mais limpo em Java. Com Java, há menos linhas de codificação e cada uma delas pode ser explicada a programadores iniciantes sem muita dificuldade. Mas não precisa ficar assustado se não estiver familiarizado com a linguagem Java. Você conseguirá acompanhar a codificação se tiver experiência de programação em outra linguagem.

Para leitores interessados em programação cliente-servidor em linguagem C, há várias boas referências à disposição [Donahoo, 2001; Stevens, 1997; Frost, 1994; Kurose, 1996].

2.7.1 Programação de aplicações com TCP

Comentamos na Seção 2.1 que processos que rodam em máquinas diferentes se comunicam uns com os outros enviando mensagens para sockets. Dissemos que cada processo é análogo a uma casa e que o socket do processo é análogo a uma porta. Como ilustrado na Figura 2.28, o socket é a porta entre o processo da aplicação e o TCP. O desenvolvedor da aplicação controla tudo que está no lado da camada de aplicação da porta; contudo, tem pouco controle do lado da camada de transporte. (No máximo, poderá fixar alguns parâmetros do TCP, tais como tamanho máximo do buffer e tamanho máximo de segmentos.)

Agora, vamos examinar mais de perto a interação dos programas cliente e servidor. O cliente tem a tarefa de iniciar contato com o servidor. Para que o servidor possa reagir ao contato inicial do cliente, tem de estar pronto, o que implica duas coisas. Em primeiro lugar, o programa servidor não pode estar inativo, isto é, tem de estar rodando como um processo antes de o cliente tentar iniciar contato. Em segundo lugar, o programa tem de ter alguma porta — mais precisamente, um socket — que acolha algum contato inicial de um processo cliente que esteja rodando em uma máquina qualquer. Recorrendo à nossa analogia casa/porta para processo/socket, às vezes, nos referiremos ao contato inicial do cliente como ‘bater à porta’.

Com o processo servidor em execução, o processo cliente pode iniciar uma conexão TCP com o servidor, o que é feito no programa cliente pela criação de um socket. Quando cria seu socket, o cliente especifica o endereço do processo servidor, a saber, o endereço IP do hospedeiro servidor e o número de porta do processo servidor. Com a criação do socket no programa cliente, o TCP no cliente inicia uma apresentação de três vias e estabelece uma conexão TCP com o servidor. A apresentação de três vias é completamente transparente para os programas cliente e servidor.

Durante a apresentação de três vias, o processo cliente bate no socket de entrada do processo servidor. Quando o servidor ‘ouve’ a batida, cria uma nova porta (mais precisamente, um novo socket) dedicada àquele cliente específico. No exemplo a seguir, a porta de entrada é um objeto ServerSocket que denominamos welcomeSocket. Quando um cliente bate nesse socket, o programa chama o método accept() do welcomeSocket,

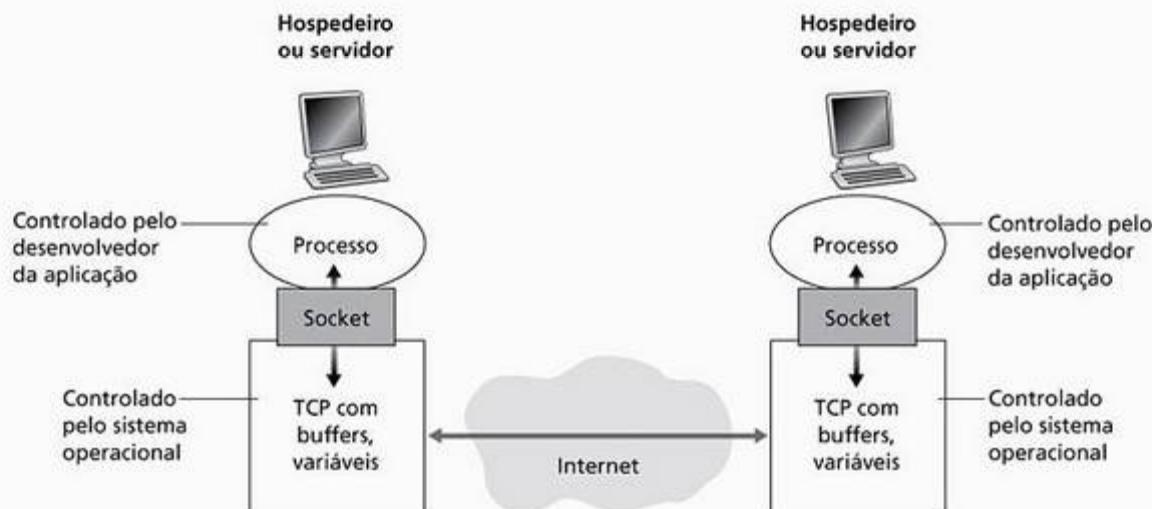


Figura 2.28 Processos que se comunicam através de sockets TCP

que cria um novo socket para o cliente. Ao final da fase de apresentação, existe uma conexão TCP entre o socket do cliente e o novo socket do servidor. Daqui em diante, vamos nos referir ao novo socket dedicado do servidor como **socket de conexão** do servidor.

Da perspectiva da aplicação, a conexão TCP é uma tubulação virtual direta entre o socket do cliente e o socket de conexão do servidor. O processo cliente pode enviar bytes para seu socket arbitrariamente; o TCP garante que o processo servidor receberá (através do socket de conexão) cada byte na ordem em que foram enviados. Assim, o TCP provê um **serviço confiável de corrente de bytes** entre os processos cliente e servidor. Além disso, exatamente como pessoas podem entrar e sair pela mesma porta, o processo cliente não somente envia bytes a seu socket, mas também os recebe dele; de modo semelhante, o processo servidor não só recebe bytes de seu socket de conexão, mas também os envia por ele. Isso é ilustrado na Figura 2.29. Como sockets desempenham um papel central em aplicações cliente-servidor, o desenvolvimento dessas aplicações também é denominado programação de sockets.

Antes de apresentarmos nosso exemplo de aplicação cliente-servidor, é útil discutirmos a noção de cadeia. Uma **cadeia** é uma sequência de caracteres que fluem para dentro ou para fora de um processo. Cada cadeia é, para o processo, uma cadeia de entrada ou uma cadeia de saída. Se a cadeia for de **entrada**, estará ligada a alguma fonte de entrada para o processo, tal como uma entrada-padrão (o teclado) ou um socket para o qual fluem dados vindos da Internet. Se a cadeia for de **saída**, estará ligada a alguma fonte de saída para o processo, tal como uma saída padrão (o monitor) ou um socket do qual fluem dados para a Internet.

2.7.2 Um exemplo de aplicação cliente-servidor em Java

Usaremos a seguinte aplicação cliente-servidor simples para demonstrar programação de sockets para TCP e UDP:

1. Um cliente lê uma linha a partir de sua **entrada padrão** (teclado) e a envia através de seu socket para o servidor.
2. O servidor lê uma linha a partir de seu socket de conexão.
3. O servidor converte a linha para letras maiúsculas.
4. O servidor envia a linha modificada ao cliente através de seu socket de conexão.
5. O cliente lê a linha modificada através de seu socket e apresenta a linha na sua **saída padrão** (monitor).

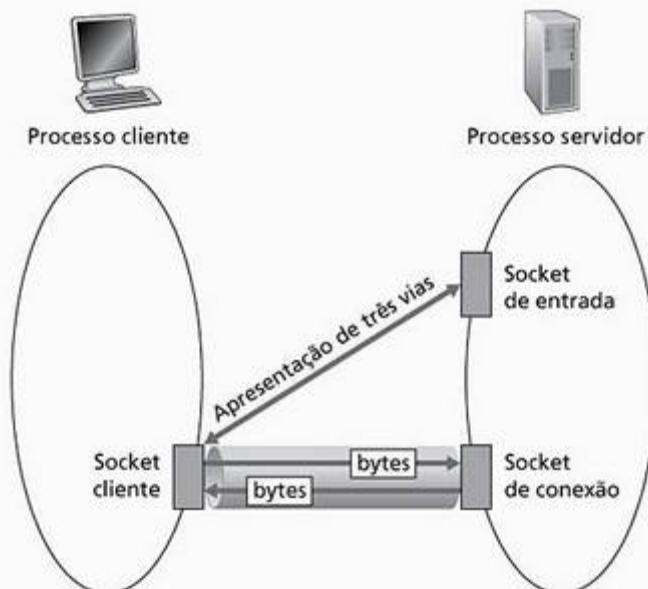


Figura 2.29 Socket cliente, socket de entrada e socket de conexão

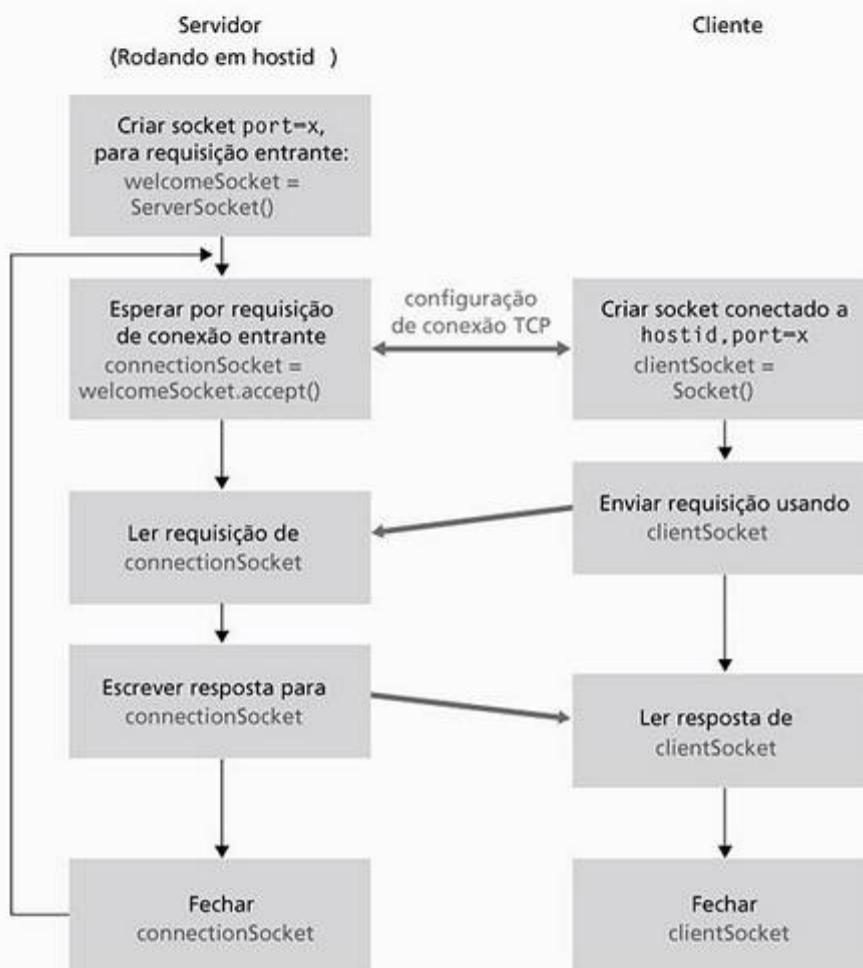


Figura 2.30 A aplicação cliente-servidor, usando serviços de transporte orientados à conexão

A Figura 2.30 ilustra a principal atividade do cliente e do servidor relacionada ao socket.

Em seguida, forneceremos o par de programas cliente-servidor para uma implementação da aplicação utilizando TCP. Apresentaremos também uma análise detalhada, linha a linha, após cada programa. O programa cliente será denominado `TCPClient.java` e o programa servidor, `TCPServer.java`. Para dar ênfase às questões fundamentais, forneceremos, intencionalmente, uma codificação objetiva, mas não tão à prova de fogo. Uma ‘boa codificação’ certamente teria algumas linhas auxiliares a mais.

Tão logo os dois programas estejam compilados em seus respectivos hospedeiros, o programa servidor será primeiramente executado no hospedeiro servidor, o que criará nele um processo servidor.

Como discutido anteriormente, o processo servidor espera para ser contatado por um processo cliente. Nesse exemplo de aplicação, quando o programa cliente é executado, um processo é criado no cliente, e esse processo imediatamente contata o servidor e estabelece uma conexão TCP com ele. O usuário no cliente pode então usar a aplicação para enviar uma linha e, em seguida, receber uma versão dessa linha em letras maiúsculas.

TCPClient.java

Eis a codificação para o lado cliente da aplicação:

```
import java.io.*;
import java.net.*;
```

```

class TCPCClient {
    public static void main(String argv[]) throws Exception
    {
        String sentence;
        String modifiedSentence;
        BufferedReader inFromUser = new BufferedReader(
            new InputStreamReader(System.in));
        Socket clientSocket = new Socket("hostname", 6789);
        DataOutputStream outToServer = new DataOutputStream(
            clientSocket.getOutputStream());
        BufferedReader inFromServer =
            new BufferedReader(new InputStreamReader(
                clientSocket.getInputStream()));
        sentence = inFromUser.readLine();
        outToServer.writeBytes(sentence + '\n');
        modifiedSentence = inFromServer.readLine();
        System.out.println("FROM SERVER: " +
            modifiedSentence);
        clientSocket.close();
    }
}

```

O programa `TCPCClient` cria três cadeias e um socket, como mostrado na Figura 2.31. O socket é denominado `clientSocket`. A cadeia `inFromUser` é uma cadeia de entrada para o programa ligada à entrada padrão (isto é, o teclado). Quando o usuário digita caracteres no teclado, eles fluem para dentro da cadeia `inFromUser`. A cadeia `inFromServer` é outra cadeia de entrada do programa ligada ao socket. Caracteres que chegam da rede fluem para dentro da cadeia `inFromServer`. Finalmente, a cadeia `outToServer` é uma cadeia de saída do programa que também é ligada ao socket. Caracteres que o cliente envia à rede fluem para dentro da cadeia `outToServer`.

Vamos agora examinar as várias linhas da codificação.

```

import java.io.*;
import java.net.*;

```

`java.io` e `java.net` são pacotes Java. O pacote `java.io` contém classes para cadeias de entrada e de saída. Em particular, contém as classes `BufferedReader` e `DataOutputStream` — classes que o programa usa para criar as três cadeias previamente ilustradas. O pacote `java.net` provê classes para suporte de rede. Em particular, contém as classes `Socket` e `ServerSocket`. O objeto `clientSocket` desse programa é derivado da classe `Socket`.

```

class TCPCClient {
    public static void main(String argv[]) throws Exception
    {
        .....
    }
}

```

Até aqui, o que vimos é material padronizado que você vê no início da maioria das codificações Java. A terceira linha é o começo de um bloco de definição de classe. A palavra-chave `class` inicia a definição de classe para a classe denominada `TCPCClient`. Uma classe contém variáveis e métodos, limitados pelas chaves `{ }` que iniciam e encerram o bloco de definição de classe. A classe `TCPCClient` não tem nenhuma variável de classe e possui exatamente um método, que é o método `main()`. Métodos são semelhantes às funções ou aos procedimentos em linguagens como C; o método `main()` na linguagem Java é semelhante à função `main()` em C e C++. Quando o interpretador Java executa uma aplicação (ao ser chamado pela classe de controle da aplicação), ele começa chamando o método `main()` da classe. O método `main()` então chama todos os outros métodos exigidos para

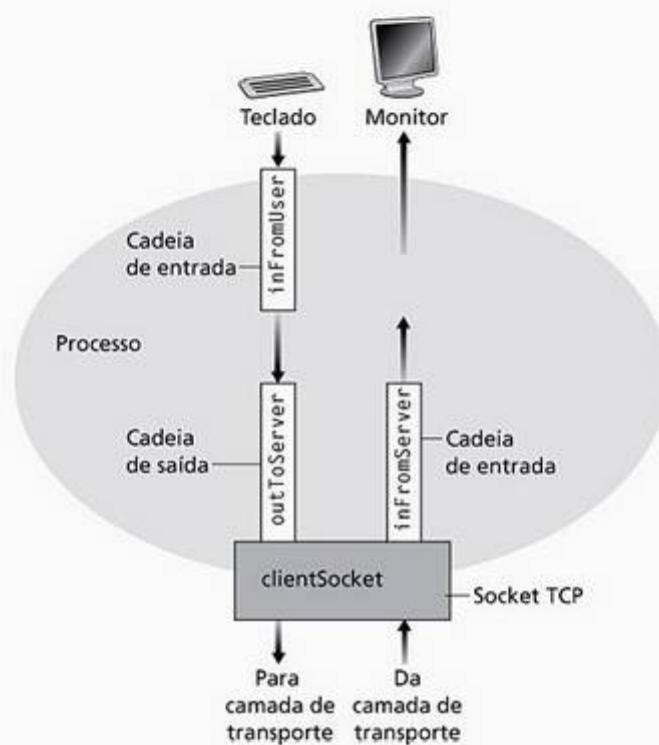


Figura 2.31 TCPClient tem três cadeias através das quais fluem caracteres

executar a aplicação. Para essa introdução à programação de portas em Java, você pode ignorar as palavras-chave `public`, `static`, `void`, `main` e `throws Exceptions` (embora deva incluí-las no código).

```
String sentence;
String modifiedSentence;
```

As duas linhas apresentadas acima declaram objetos do tipo `String` (cadeia). O objeto `sentence` é a cadeia digitada pelo usuário e enviada ao servidor. O objeto `modifiedSentence` é a cadeia obtida do servidor e enviada à saída padrão do usuário.

```
BufferedReader inFromUser = new BufferedReader(new InputStreamReader(System.in));
```

Essa linha cria o objeto de cadeia `inFromUser` do tipo `BufferedReader`. A cadeia de entrada é inicializada com `System.in`, que vincula a cadeia à entrada padrão. O comando permite que o cliente leia o texto de seu teclado.

```
Socket clientSocket = new Socket("hostname", 6789);
```

Essa linha cria o objeto `clientSocket` do tipo `Socket`. Ela também ativa a conexão TCP entre cliente e servidor. A cadeia `host-name` deve ser substituída pelo nome de hospedeiro do servidor (por exemplo, `apple.poly.edu`). Antes de a conexão TCP ser realmente iniciada, o cliente realiza um procedimento de consulta ao DNS do nome de hospedeiro para obter o endereço IP da máquina. O número 6789 é o número de porta. Você pode usar um número de porta diferente, mas precisa obrigatoriamente usar o mesmo do lado servidor da aplicação. Como discutimos anteriormente, o processo servidor é identificado pelo endereço IP do hospedeiro juntamente com o número de porta da aplicação.

```
DataOutputStream outToServer =
    new DataOutputStream(clientSocket.getOutputStream());
BufferedReader inFromServer =
    new BufferedReader(new InputStreamReader(
        clientSocket.getInputStream()));
```

Essas linhas criam objetos de cadeia que são ligados ao socket. A cadeia `outToServer` fornece a saída do processo para o socket. A cadeia `inFromServer` fornece ao processo a entrada do socket (veja a Figura 2.31).

```
sentence = inFromUser.readLine();
```

Essa linha coloca uma linha digitada pelo usuário na cadeia `sentence`. A cadeia `sentence` continua a juntar caracteres até que o usuário termine a linha digitando 'carriage return'. A linha passa da entrada padrão para dentro da cadeia `sentence` por meio da cadeia `inFromUser`.

```
outToServer.writeBytes(sentence + '\n');
```

Esta linha envia a cadeia `sentence` para dentro da corrente `outToServer` ampliada com um 'carriage return'. A sentença ampliada flui pelo socket do cliente para dentro da conexão TCP. O cliente então espera para receber caracteres do servidor.

```
modifiedSentence = inFromServer.readLine();
```

Quando chegam do servidor, os caracteres fluem através da cadeia `inFromServer` e são colocados dentro da cadeia `modifiedSentence`. Continuam a se acumular em `modifiedSentence` até que a linha termine com um caractere de 'carriage return'.

```
System.out.println("FROM SERVER " + modifiedSentence);
```

Essa linha envia para o monitor a cadeia `modifiedSentence` retornada pelo servidor.

```
clientSocket.close();
```

Essa última linha fecha o socket e, por conseguinte, a conexão TCP entre o cliente e o servidor. Ela faz com que o TCP no cliente envie uma mensagem para o TCP no servidor (veja a Seção 3.5).

TCPserver.java

Agora vamos examinar o programa servidor.

```
import java.io.*;
import java.net.*;
class TCPserver {
    public static void main(String argv[]) throws Exception
    {
        String clientSentence;
        String capitalizedSentence;
        ServerSocket welcomeSocket = new ServerSocket(
            6789);
        while(true) {
            Socket connectionSocket = welcomeSocket.
                accept();
            BufferedReader inFromClient =
                new BufferedReader(new InputStreamReader(
                    connectionSocket.getInputStream()));
            DataOutputStream outToClient =
                new DataOutputStream(
                    connectionSocket.getOutputStream());
            clientSentence = inFromClient.readLine();
            capitalizedSentence =
                clientSentence.toUpperCase() + '\n';
            outToClient.writeBytes(capitalizedSentence);
        }
    }
}
```

TCPServer tem muitas semelhanças com TCPClient. Vamos agora dar uma olhada nas linhas em TCPServer.java. Não comentaremos as linhas que são idênticas ou semelhantes aos comandos em TCPClient.java.

A primeira linha em TCPServer é substancialmente diferente da que vimos em TCPClient:

```
ServerSocket welcomeSocket = new ServerSocket(6789);
```

Essa linha cria o objeto welcomeSocket, que é do tipo ServerSocket. O welcomeSocket é uma espécie de porta que fica ‘ouvindo’, à espera que algum cliente bata. O welcomeSocket fica à escuta na porta número 6789. A linha seguinte é:

```
Socket connectionSocket = welcomeSocket.accept();
```

Essa linha cria um novo socket, denominado connectionSocket, quando algum cliente bate à porta welcomeSocket. O número de porta desse socket também é 6789. (Explicaremos por que ambos têm o mesmo número de porta no Capítulo 3). O TCP então estabelece uma conexão virtual direta entre clientSocket no cliente e connectionSocket no servidor. O cliente e o servidor podem então enviar bytes um para o outro pela conexão, e todos os bytes enviados chegam ao outro lado na ordem certa. Com connectionSocket estabelecido, o servidor pode continuar à escuta por outros clientes que requisitarão a aplicação usando welcomeSocket. (Essa versão do programa, na verdade, não fica à escuta por mais requisições de conexão, mas pode ser modificada com threads para fazer isso.) O programa então cria diversos objetos de cadeia, análogos aos objetos de cadeia em clientSocket. Considere agora:

```
capitalizedSentence = clientSentence.toUpperCase() + '\n';
```

Esse comando é o coração da aplicação. Ele toma a linha enviada pelo cliente, passa todas as letras para maiúsculas e adiciona um ‘carriage return’. Ele usa o método toUpperCase(). Todos os outros comandos no programa são periféricos, usados para a comunicação com o cliente.

Para testar o par de programas, instale e compile TCPClient.java em um hospedeiro e TCPServer.java em outro hospedeiro. Não se esqueça de incluir o nome de hospedeiro do servidor adequado em TCPClient.java. Então, execute TCPServer.class, o programa servidor compilado, no servidor. Isso cria um processo no servidor que fica ocioso até ser contatado por algum cliente. Então, execute TCPClient.class, o programa cliente compilado, no cliente. Isso cria um processo no cliente e estabelece uma conexão TCP entre os processos cliente e servidor. Por fim, para usar a aplicação, digite uma sentença seguida de um ‘carriage return’.

Para desenvolver sua própria aplicação cliente-servidor, você pode começar modificando ligeiramente os programas. Por exemplo, em vez de converter todas as letras para maiúsculas, o servidor poderia contar o número de vezes que a letra ‘s’ aparece no texto e retornar esse número.

2.8 Programação de aplicações com UDP

Aprendemos na seção anterior que, quando dois processos se comunicam por TCP, é como se houvesse uma tubulação entre eles, que permanece ativa até que um dos dois processos a feche. Quando um dos processos quer enviar alguns bytes para o outro processo, simplesmente insere os bytes na tubulação. O processo de envio não tem de acrescentar um endereço de destino aos bytes porque a tubulação está ligada logicamente ao destino. Além disso, a tubulação provê um canal confiável de cadeia de bytes — a sequência de bytes recebida pelo processo receptor é exatamente a mesma que o remetente inseriu na tubulação.

O UDP também permite que dois (ou mais) processos que rodam em hospedeiros diferentes se comuniquem. Contudo, é diferente do TCP de muitas maneiras fundamentais. Primeiramente, é um serviço não orientado para conexão — não há uma fase inicial de apresentação, durante a qual é estabelecida uma tubulação entre os dois processos. Como o UDP não tem uma tubulação, quando um processo quer enviar um conjunto de bytes a outro, o processo remetente deve anexar o endereço do processo destinatário ao conjunto de bytes. E isso precisa ser feito para cada conjunto de bytes que o processo remetente enviar. Como analogia, considere um grupo de 20 pessoas que toma cinco táxis para um mesmo destino; quando cada um dos grupos entra em um carro, tem de informar separadamente ao motorista o endereço de destino. Assim, o UDP é semelhante a um serviço de táxi. O endereço de destino é uma tupla que consiste no endereço IP do hospedeiro destinatário e no número de porta do

processo destinatário. Vamos nos referir ao conjunto de bytes de informação, juntamente com o endereço IP do destinatário e o número de porta, como 'pacote'. O UDP provê um modelo de serviço não confiável orientado para mensagem, no sentido de que faz o melhor esforço para entregar o conjunto de bytes ao destino. Esses conjuntos são bytes enviados em uma única operação na parte remetente, e serão entregues como um conjunto na parte destinatária; isso contrasta com o sentido da cadeia de bytes do TCP. O serviço UDP faz o melhor esforço já que o UDP não garante que o conjunto de bytes será realmente entregue. O serviço UDP contrasta acentuadamente (em vários aspectos) com o modelo de serviço confiável de cadeia de bytes do TCP.

Após ter criado um pacote, o processo remetente empurra-o para dentro da rede através de um socket. Continuando com nossa analogia do táxi, do outro lado do socket remetente há um táxi esperando pelo pacote. Esse táxi, então, leva o pacote até seu endereço de destino. Contudo, não garante que finalmente entregará o pacote no seu destino final, pois pode quebrar ou sofrer algum outro problema não previsto no caminho. Em outras palavras, o UDP provê serviço não confiável de transporte a seus processos de comunicação — não dá nenhuma garantia de que um datagrama ('pacote') alcançará seu destino final.

Nesta seção, ilustraremos programação de socket desenvolvendo novamente a mesma aplicação da seção anterior, mas, dessa vez, com UDP. Veremos que a codificação para UDP é diferente da codificação para TCP de muitas maneiras importantes. Em particular, (1) não há apresentação inicial entre os dois processos e, portanto, não há necessidade de um socket de entrada, (2) não há cadeias ligadas aos sockets, (3) os hospedeiros remetentes criam pacotes anexando o endereço IP do destinatário e o número de porta a cada conjunto de bytes que enviam e (4) o processo destinatário deve desmontar cada pacote recebido para obter os bytes de informação do pacote.

Lembre-se, mais uma vez, da nossa aplicação simples:

1. Um cliente lê uma linha a partir de sua entrada padrão (teclado) e a envia através de seu socket para o servidor.
2. O servidor lê uma linha a partir de seu socket.
3. O servidor converte a linha para letras maiúsculas.
4. O servidor envia a linha modificada através de seu socket ao cliente.
5. O cliente lê a linha modificada através de seu socket e apresenta a linha em sua saída padrão (monitor).

A Figura 2.32 destaca a principal atividade relacionada ao socket realizada pelo cliente e pelo servidor, que se comunicam por meio de um serviço de transporte (UDP) não orientado para conexão.

UDPClient.java

Eis a codificação para o lado cliente da aplicação:

```
import java.io.*;
import java.net.*;
class UDPClient {
    public static void main(String args[]) throws Exception
    {
        BufferedReader inFromUser =
            new BufferedReader(new InputStreamReader
                (System.in));
        DatagramSocket clientSocket = new DatagramSocket();
        InetAddress IPAddress =
            InetAddress.getByName("hostname");
        byte[] sendData = new byte[1024];
        byte[] receiveData = new byte[1024];
        String sentence = inFromUser.readLine();
        sendData = sentence.getBytes();
        DatagramPacket sendPacket =
```

```

        new DatagramPacket(sendData, sendData.length,
                           IPAddress, 9876);
    clientSocket.send(sendPacket);
    DatagramPacket receivePacket =
        new DatagramPacket(receiveData,
                           receiveData.length);
    clientSocket.receive(receivePacket);
    String modifiedSentence =
        new String(receivePacket.getData());
    System.out.println("FROM SERVER:" +
        modifiedSentence);
    clientSocket.close();
}
}

```

O programa UDPClient.java constrói uma cadeia e um socket, como mostra a Figura 2.33. O socket é denominado `clientSocket` e é do tipo `DatagramSocket`. Note que o UDP usa, no cliente, um tipo de socket diferente do usado no TCP. Em particular, com UDP nosso cliente usa um `DatagramSocket`, enquanto com TCP ele usou um `Socket`. A cadeia `inFromUser` é uma cadeia de entrada para o programa; está ligada à entrada padrão, isto é, ao teclado. Tínhamos uma cadeia equivalente em nossa versão TCP do programa. Quando o usuário digita caracteres no teclado, esses caracteres fluem para dentro da cadeia `inFromUser`. Porém, ao contrário do TCP, não há cadeias (de entrada e de saída) ligadas ao socket. Em vez de alimentar bytes à cadeia ligada a um objeto `Socket`, o UDP transmite pacotes individuais por meio do objeto `DatagramSocket`.

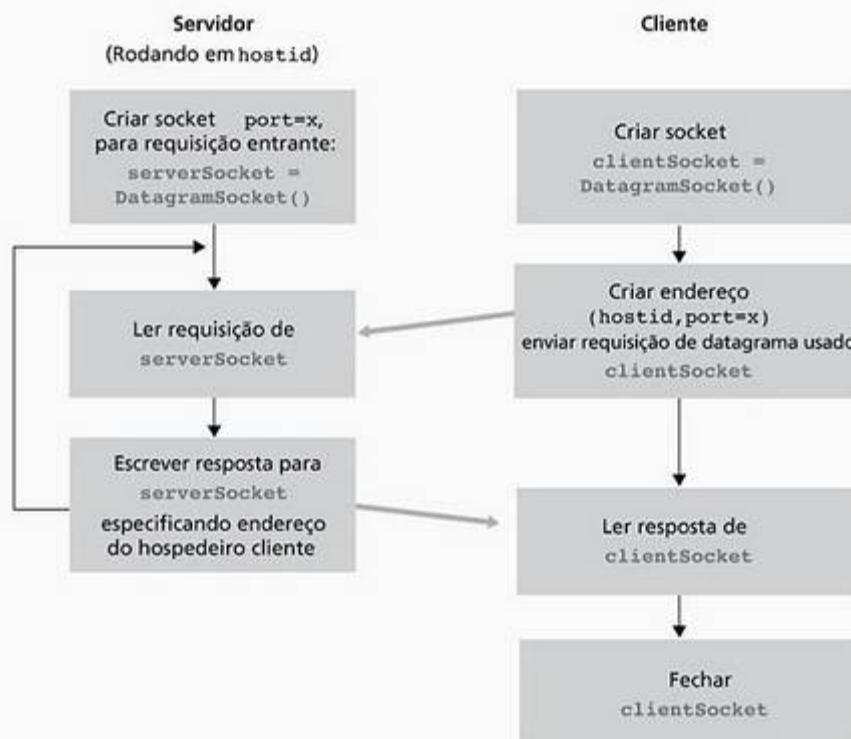


Figura 2.32 A aplicação cliente-servidor usando serviços de transporte não orientados para conexão

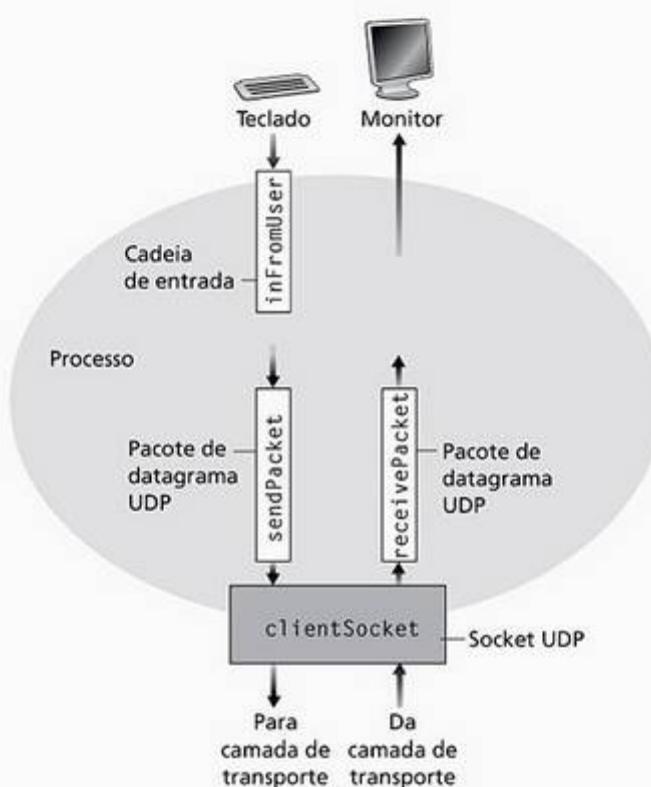


Figura 2.33 UDPClient tem uma corrente; o socket aceita pacotes do processo e entrega pacotes ao processo

Agora, vamos examinar as linhas da codificação de `TCPClient.java` que são significativamente diferentes.

```
DatagramSocket clientSocket = new DatagramSocket();
```

Essa linha cria o objeto `clientSocket` do tipo `DatagramSocket`. Ao contrário do `TCPClient.java`, ela não ativa uma conexão TCP. Em particular, o hospedeiro cliente não contata o hospedeiro servidor durante a execução desta linha. Por essa razão, o construtor `DatagramSocket()` não toma como argumento o nome do servidor ou o número da porta. Usando nossa analogia porta-tubulação, a execução da linha acima cria um socket para o processo cliente, mas não cria uma tubulação entre os dois processos.

```
InetAddress IPAddress = InetAddress.getByName("hostname");
```

Para enviar bytes a um processo destinatário, precisamos do endereço do processo. Parte desse endereço é o endereço IP do hospedeiro destinatário. A linha apresentada invoca uma consulta ao DNS que traduz o nome do destinatário (nesse exemplo, fornecido na codificação pelo programador) para um endereço IP. O DNS também foi chamado pela versão TCP do cliente, embora o tenha feito de maneira implícita, e não explícita. O método `getByName()` toma como argumento o nome de hospedeiro do servidor e retorna o endereço IP desse mesmo servidor. Coloca esse endereço no objeto `IPAddress` do tipo `InetAddress`.

```
byte[] sendData = new byte[1024];
byte[] receiveData = new byte[1024];
```

Os vetores de bytes `sendData` e `receiveData` reterão os dados que o cliente envia e recebe, respectivamente.
`sendData = sentence.getBytes();`

Essa linha realiza, essencialmente, uma conversão de tipo. Pega a cadeia `sentence` e a renomeia como `sendData`, que é um vetor de bytes.

```
DatagramPacket sendPacket = new DatagramPacket(
    sendData, sendData.length, IPAddress, 9876);
```

Essa linha constrói o pacote, `sendPacket`, que o cliente enviará para a rede através do seu socket. Esse pacote inclui os dados contidos nele, `sendData`, o comprimento desses dados, o endereço IP do servidor e o número de porta da aplicação (escolhemos 9876). Note que `sendPacket` é do tipo `DatagramPacket`.

```
clientSocket.send(sendPacket);
```

Nessa linha, o método `send()` do objeto `clientSocket` toma o pacote recém-construído e passa-o para a rede através de `clientSocket`. Mais uma vez, note que o UDP envia a linha de caracteres de maneira muito diferente do TCP. O TCP simplesmente inseriu a linha de caracteres em uma cadeia que tinha uma conexão lógica direta com o servidor; o UDP cria um pacote que inclui o endereço do servidor. Após enviar esse pacote, o cliente espera para receber um pacote do servidor.

```
DatagramPacket receivePacket =
    new DatagramPacket(receiveData, receiveData.length);
```

Nessa linha, enquanto espera pelo pacote do servidor, o cliente cria um lugar reservado para o pacote, `receivePacket`, um objeto do tipo `DatagramPacket`.

```
clientSocket.receive(receivePacket);
```

O cliente fica ocioso até receber um pacote; quando enfim o recebe, ele o coloca em `receivePacket`.

```
String modifiedSentence =
    new String(receivePacket.getData());
```

Essa linha extrai os dados de `receivePacket` e realiza uma conversão de tipo, convertendo um vetor de bytes na cadeia `modifiedSentence`.

```
System.out.println("FROM SERVER:" + modifiedSentence);
```

Essa linha, que também está presente no `TCPClient`, apresenta a cadeia `modifiedSentence` no monitor do cliente.

```
clientSocket.close();
```

Essa última linha fecha o socket. Como o UDP não é orientado para conexão, esta linha não faz com que o cliente envie uma mensagem de camada de transporte ao servidor (ao contrário do `TCPClient`).

UDPServer.java

Vamos agora dar uma olhada no lado servidor da aplicação:

```
import java.io.*;
import java.net.*;
class UDPServer {
    public static void main(String args[]) throws Exception
    {
        DatagramSocket serverSocket = new
            DatagramSocket(9876);
        byte[] receiveData = new byte[1024];
        byte[] sendData = new byte[1024];
        while (true)
        {
            DatagramPacket receivePacket =
                new DatagramPacket(receiveData,
                    receiveData.length);
            serverSocket.receive(receivePacket);
            String sentence = new String(
                receivePacket.getData());
            InetAddress IPAddress =
                receivePacket.getAddress();
```

```

        int port = receivePacket.getPort();
        String capitalizedSentence =
            sentence.toUpperCase();
        sendData = capitalizedSentence.getBytes();
        DatagramPacket sendPacket =
        new DatagramPacket (sendData,
                            sendData.length, IPAddress, port);
        serverSocket.send(sendPacket);
    }
}
}

```

O programa `UDPServer.java` constrói um socket, como mostra a Figura 2.34. O socket é denominado `serverSocket`. É um objeto do tipo `DatagramSocket`, como era o socket do lado cliente da aplicação. Mais uma vez, nenhuma cadeia está ligada ao socket.

Vamos agora examinar as linhas da codificação que são diferentes de `TCPServer.java`.

```
DatagramSocket serverSocket = new DatagramSocket (9876);
```

Essa linha constrói o `DatagramSocket` `serverSocket` na porta 9876. Todos os dados enviados e recebidos passarão através desse socket. Como o UDP não é orientado para conexão, não temos de criar um novo socket e continuar à escuta de novas requisições de conexão, como é feito no `TCPServer.java`. Se vários clientes acessarem essa aplicação, todos enviarão seus pacotes por esse único socket, `serverSocket`.

```

String sentence = new String(receivePacket.getData());
InetAddress IPAddress = receivePacket.getAddress();
int port = receivePacket.getPort();

```

Essas três linhas desmontam o pacote que chega do cliente. A primeira delas extrai os dados do pacote e os coloca no `String sentence`; há uma linha análoga em `UDPClient`. A segunda linha extrai o endereço IP. A terceira linha extrai o número de porta do cliente, que é escolhido pelo cliente e é diferente do número de porta 9876 do servidor. (Discutiremos números de porta do cliente com mais detalhes no capítulo seguinte.) É necessário que o servidor obtenha o endereço (endereço IP e número de porta) do cliente para que possa enviar a sentença em letras maiúsculas de volta para ele.

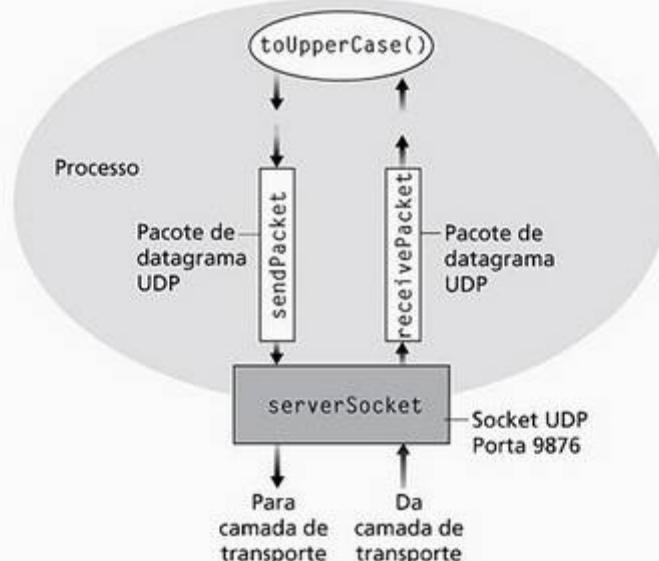


Figura 2.34 UDPServer não tem Cadeias; o socket aceita pacotes do processo e entrega pacotes ao processo

Isso conclui nossa análise sobre o par de programas UDP. Para testar a aplicação, instale e compile `UDPClient.java` em um hospedeiro e `UDPServer.java` em outro. (Não esqueça de incluir o nome apropriado do hospedeiro do servidor em `UDPClient.java`.) Em seguida, execute os dois programas em seus respectivos hospedeiros. Ao contrário do TCP, você pode executar primeiramente o lado cliente e depois o lado servidor. Isso acontece porque, quando você executa o programa cliente, o processo cliente não tenta iniciar uma conexão com o servidor. Assim que tiver executado os programas cliente e servidor, você poderá usar a aplicação digitando uma linha no cliente.

2.9 Resumo

Neste capítulo, estudamos os aspectos conceituais e os aspectos de implementação de aplicações de rede. Conhecemos a onipresente arquitetura cliente-servidor adotada por aplicações da Internet e examinamos sua utilização nos protocolos HTTP, FTP, SMTP, POP3 e DNS. Analisamos esses importantes protocolos de camada de aplicação e suas aplicações associadas (Web, transferência de arquivos, e-mail e DNS) com algum detalhe. Conhecemos também a arquitetura P2P, cada vez mais dominante, e examinamos sua utilização em muitas aplicações. Vimos como o API socket pode ser usado para construir aplicações de rede. Examinamos a utilização de portas para serviços de transporte fim a fim orientados para conexão (TCP) e não orientados para conexão (UDP) e também construímos um servidor Web simples usando sockets. A primeira etapa de nossa jornada de descida pela arquitetura das camadas da rede está concluída!

Logo no começo deste livro, na Seção 1.1, demos uma definição um tanto vaga e despojada de um protocolo. Dissemos que um protocolo é ‘o formato e a ordem das mensagens trocadas entre duas ou mais entidades comunicantes, bem como as ações realizadas na transmissão e/ou no recebimento de uma mensagem ou outro evento’. O material deste capítulo — em particular, o estudo detalhado dos protocolos HTTP, FTP, SMTP, POP3 e DNS — agregou considerável substância a essa definição. Protocolos são o conceito fundamental de redes. Nosso estudo sobre protocolos de aplicação nos deu agora a oportunidade de desenvolver uma noção mais intuitiva do que eles realmente são.

Na Seção 2.1, descrevemos os modelos de serviço que o TCP e o UDP oferecem às aplicações que os chamam. Examinamos esses modelos de serviço ainda mais de perto quando desenvolvemos, nas seções 2.7 e 2.8, aplicações simples que executam em TCP e UDP. Contudo, pouco dissemos sobre como o TCP e o UDP fornecem esses modelos de serviço. Por exemplo, sabemos que o TCP provê um serviço de dados confiável, mas ainda não mencionamos como ele o faz. No próximo capítulo, examinaremos cuidadosamente não apenas o que são protocolos de transporte, mas também o como e o porquê deles.

Agora que conhecemos a estrutura da aplicação da Internet e os protocolos de camada de aplicação, estamos prontos para continuar a descer a pilha de protocolos e examinar a camada de transporte no Capítulo 3.



Exercícios de fixação

Capítulo 2 Questões de revisão

Seção 2.1

1. Relacione cinco aplicações da Internet não proprietárias e os protocolos de camada de aplicação que elas usam.
2. Qual é a diferença entre arquitetura de rede e arquitetura de aplicação?
3. Para uma sessão de comunicação entre um par de processos, qual processo é o cliente e qual é o servidor?
4. Em uma aplicação de compartilhamento de arquivos P2P, você concorda com a afirmação: “não existe nenhuma noção de lados cliente e servidor de uma sessão de comunicação”? Por quê?
5. Que informação é usada por um processo que está rodando em um hospedeiro para identificar um processo que está rodando em outro hospedeiro?

6. Suponha que você queria fazer uma transação de um cliente remoto para um servidor da maneira mais rápida possível. Você usaria o UDP ou o TCP? Por quê?
7. Com referência à Figura 2.4, vemos que nenhuma das aplicações relacionadas nela requer ‘sem perda de dados’ e ‘temporização’. Você consegue imaginar uma aplicação que requeira ‘sem perda de dados’ e seja também altamente sensível ao atraso?
8. Relacione quatro classes de serviços que um protocolo de transporte pode prover. Para cada uma delas, indique se o UDP ou o TCP (ou ambos) fornece tal serviço.
9. Lembre-se de que o TCP pode ser aprimorado com o SSL para fornecer serviços de segurança processo a processo, incluindo a decodificação. O SSL opera na camada de transporte ou na camada de aplicação? Se o desenvolvedor da aplicação quer que o TCP seja aprimorado com o SSL, o que ele deve fazer?

Seções 2.2 a 2.5

10. O que significa protocolo de apresentação (handshaking protocol)?
11. Por que HTTP, FTP, SMTP, POP3 rodam sobre TCP e não sobre UDP?
12. Considere um site de comércio eletrônico que quer manter um registro de compras para cada um de seus clientes. Descreva como isso pode ser feito com cookies.
13. Descreva como o cache Web pode reduzir o atraso na recepção de um objeto desejado. O cache Web reduzirá o atraso para todos os objetos requisitados por um usuário ou somente para alguns objetos? Por quê?
14. Digite um comando Telnet em um servidor Web e envie uma mensagem de requisição com várias linhas. Inclua nessa mensagem a linha de cabeçalho `If-modified-since:` para forçar uma mensagem de resposta com a codificação de estado `304 Not Modified`.
15. Por que se diz que o FTP envia informações de controle ‘fora da banda’?
16. Suponha que Alice envie uma mensagem a Bob por meio de uma conta de e-mail da Web (como o Hotmail), e que Bob acesse seu e-mail por seu servidor de correio usando POP3. Descreva como a mensagem vai do hospedeiro de Alice até o hospedeiro de Bob. Não se esqueça de relacionar a série de protocolos de camada de aplicação usados para movimentar a mensagem entre os dois hospedeiros.
17. Imprima o cabeçalho de uma mensagem de e-mail que acabou de receber. Quantas linhas de cabeçalho `Received:` há nela? Analise cada uma das linhas.

18. Da perspectiva de um usuário, qual é a diferença entre o modo ler-e-apagar e o modo ler-e-guardar no POP3?
19. É possível que o servidor Web e o servidor de correio de uma organização tenham exatamente o mesmo apelido para um nome de hospedeiro (por exemplo, `foo.com`)? Qual seria o tipo de RR que contém o nome de hospedeiro do servidor de correio?

Seção 2.6

20. No BitTorrent, suponha que Alice forneça blocos para Bob durante um intervalo de 30 segundos. Bob retornará, necessariamente, o favor e fornecerá blocos para Alice no mesmo intervalo? Por quê?
21. Considere um novo par, Alice, que entra no BitTorrent sem possuir nenhum bloco. Sem qualquer bloco, ela não pode se tornar uma das quatro melhores exportadoras de dados para qualquer um dos outros pares, visto que ela não possui nada para enviar. Então, como Alice obterá seu primeiro bloco?
22. O que é uma rede de sobreposição em um sistema de compartilhamento de arquivos P2P? Ela inclui roteadores? O que são as arestas da rede de sobreposição? Como a rede de sobreposição de inundação de consultas é criada e como é mantida?
23. De que modo a aplicação mensagem instantânea é um híbrido das arquiteturas cliente-servidor e P2P?
24. Considere um DHT com uma topologia da rede de sobreposição (ou seja, cada par rastreia todos os pares no sistema). Quais são as vantagens e desvantagens de um DHT circular (sem atalhos)?
25. O Skype utiliza técnicas P2P para duas funções importantes. Quais são elas?
26. Relacione quatro diferentes aplicações que são apropriadas naturalmente para arquiteturas P2P. (Dica: Distribuição de arquivo e mensagem instantânea são duas.)

Seções 2.7-2.8

27. O servidor UDP descrito na Seção 2.8 precisava de uma porta apenas, ao passo que o servidor TCP descrito na Seção 2.7 precisava de duas portas. Por quê? Se um servidor TCP tivesse de suportar n conexões simultâneas, cada uma de um hospedeiro cliente diferente, de quantas portas precisaria?
28. Para a aplicação cliente-servidor por TCP descrita na Seção 2.7, por que o programa servidor deve ser executado antes do programa cliente? Para a aplicação cliente-servidor por UDP descrita na Seção 2.8, por que o programa cliente pode ser executado antes do programa servidor?



Problemas

1. Falso ou verdadeiro?
 - a. Um usuário requisita uma página Web que consiste em texto e três imagens. Para essa página, o cliente enviará uma mensagem de requisição e receberá quatro mensagens de resposta.
 - b. Duas páginas Web distintas (por exemplo, www.mit.edu/research.html e www.mit.edu/students.html) podem ser enviadas pela mesma conexão persistente.
 - c. Com conexões não persistentes entre browser e servidor de origem, é possível que um único segmento TCP transporte duas mensagens distintas de requisição HTTP.
 - d. O cabeçalho Date: na mensagem de resposta HTTP indica a última vez que o objeto da resposta foi modificado.
 - e. As mensagens de resposta HTTP nunca possuem um corpo de mensagem vazio.
2. Leia o RFC 959 para FTP. Relacione todos os comandos de cliente que são suportados pelo RFC.
3. Considere um cliente HTTP que queira obter um documento Web em um dado URL. Inicialmente, o endereço IP do servidor HTTP é desconhecido. Nesse cenário, quais protocolos de transporte e de camada de aplicação são necessários, além do HTTP?
4. Considere a seguinte cadeia de caracteres ASCII capturada pelo Wireshark quando o browser enviou uma mensagem HTTP GET (ou seja, o conteúdo real de uma mensagem HTTP GET). Os caracteres <cr><lf> são retorno de carro e avanço de linha (ou seja, a cadeia de caractere em itálico <cr> no texto abaixo representa o caractere único retorno de carro que estava contido, naquele momento, no cabeçalho HTTP). Responda às seguintes questões, indicando onde está a resposta na mensagem HTTP GET abaixo.


```
GET /cs453/index.html HTTP/1.1
<cr><lf>Host: gaia.cs.umass.edu<cr><lf>User Agent: Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US; rv:1.7.2) Gecko/20040804
Netscape/7.2 (ax) <cr><lf>Accept: text/xml, application/xml, application/xhtml+xml, text/html;q=0.9, text/plain;q=0.8, image/png,*/*;q=0.5 <cr><lf>Accept-Language: en-us,en;q=0.5<cr><lf>Accept-Encoding: zip,deflate<cr><lf>Accept-Charset: ISO -8859-1, utf-8;q=0.7,*;q=0.7 <cr><lf>Keep-Alive:
```
5. O texto a seguir mostra a resposta enviada do servidor em reação à mensagem HTTP GET na questão acima. Responda às seguintes questões, indicando onde está a resposta na mensagem abaixo.


```
HTTP/1.1 200 OK<cr><lf>Date:Tue, 07 Mar 2008
12:39:45 GMT<cr><lf>Server: Apache/2.0.52 (Fedora)
<cr><lf>Last-Modified: Sat, 10 Dec2005 18:27:46 GMT<cr><lf>ETag: "526c3-f22-a88a4c80"<cr><lf>Accept-Ranges: bytes<cr><lf>Content-Length: 3874<cr><lf>Keep-Alive:timeout=max=100
<cr><lf>Connection:Keep-Alive<cr><lf>Content-Type: text/html; charset=ISO-8859-1<cr><lf><cr><lf><!doctype html public "-//w3c//dtd html 4.0 transitional//en"><lf><html><lf><head><lf><meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1"><lf><meta name="GENERATOR" content="Mozilla/4.79 [en] (Windows NT 5.0; U) Netscape]"><lf><title>CMPSCI 453 / 591 / NTU-ST550A Spring 2005 homepage</title><lf></head><lf><much more document text following here (not shown)>
      
```

 - a. Qual é a URL do documento requisitado pelo browser?
 - b. Qual versão do HTTP o browser está rodando?
 - c. O browser requisita uma conexão não persistente ou persistente?
 - d. Qual é o endereço IP do hospedeiro no qual o browser está rodando?
 - e. Que tipo de browser inicia essa mensagem? Por que é necessário o tipo de browser em uma mensagem de requisição HTTP?

- d. Quais são os 5 primeiros bytes do documento que está retornando? O servidor aceitou uma conexão persistente?
6. Obtenha a especificação HTTP/1.1 (RFC 2616). Responda às seguintes perguntas:
- Explique o mecanismo de sinalização que cliente e servidor utilizam para indicar que uma conexão persistente está sendo fechada. O cliente, o servidor, ou ambos, podem sinalizar o encerramento de uma conexão?
 - Que serviços de criptografia são providos pelo HTTP?
 - O cliente é capaz de abrir três ou mais conexões simultâneas com um determinado servidor?
 - Um servidor ou um cliente pode abrir uma conexão de transporte entre eles se um dos dois descobrir que a conexão ficou lenta por um tempo. É possível que um lado comece a encerrar a conexão enquanto o outro está transmitindo dados por meio dessa conexão? Explique.
7. Suponha que você clique com seu browser Web sobre um ponteiro para obter uma página Web e que o endereço IP para o URL associado não esteja no cache de seu hospedeiro local. Portanto, será necessária uma consulta ao DNS para obter o endereço IP. Considere que n servidores DNS sejam visitados antes que seu hospedeiro receba o endereço IP do DNS; as visitas sucessivas incorrem em um RTT igual a RTT_1, \dots, RTT_n . Suponha ainda que a página Web associada ao ponteiro contenha exatamente um objeto que consiste em uma pequena quantidade de texto HTML. Seja RTT_o o RTT entre o hospedeiro local e o servidor que contém o objeto. Admitindo que o tempo de transmissão do objeto seja zero, quanto tempo passará desde que o cliente clica o ponteiro até que receba o objeto?
8. Com referência ao problema 7, suponha que o arquivo HTML refencie três objetos muito pequenos no mesmo servidor. Desprezando tempos de transmissão, quanto tempo passa, usando-se:
- HTTP não persistente sem conexões TCP paralelas?
 - HTTP não persistente com o browser configurado para 5 conexões paralelas?
 - HTTP persistente?
9. Considere a Figura 2.12, que mostra uma rede institucional conectada à Internet. Suponha que o tamanho médio do objeto seja 850 mil bits e que a taxa média de requisição dos browsers da instituição aos servidores de origem seja 1,6 requisição por segundo. Suponha também que a quantidade de tempo que leva desde o instante em que o roteador do lado da Internet do enlace de acesso transmite uma requisição HTTP até que receba a resposta seja 3 segundos em média (veja Seção 2.2.5). Modele o tempo total médio de resposta como a soma do atraso de acesso médio (isto é, o atraso entre o roteador da Internet e o roteador da instituição) e o tempo médio de atraso da Internet. Para a média de atraso de acesso, use $\Delta(1 - \Delta\beta)$, onde Δ é o tempo médio requerido para enviar um objeto pelo enlace de acesso e β é a taxa de chegada de objetos ao enlace de acesso.
- Determine o tempo total médio de resposta.
 - Agora, considere que um cache é instalado na LAN institucional e que a taxa de resposta local seja 0,4. Determine o tempo total de resposta.
10. Considere um enlace curto de 10 metros através do qual um remetente pode transmitir a uma taxa de 150 bits/s em ambas as direções. Suponha que os pacotes com dados tenham 100 mil bits de comprimento, e os pacotes que contêm controle (por exemplo, ACK ou apresentação) tenham 200 bits de comprimento. Admita que N conexões paralelas recebam cada 1/N da largura de banda do enlace. Agora, considere o protocolo HTTP e suponha que cada objeto baixado tenha 100 Kbits de comprimento e que o objeto inicial baixado contenha 10 objetos referenciados do mesmo remetente. Os downloads paralelos por meio de instâncias paralelas de HTTP não persistente fazem sentido nesse caso? Agora considere o HTTP persistente. Você espera ganhos significativos sobre o caso não persistente? Justifique sua resposta.
11. Considere o cenário apresentado na questão anterior. Agora suponha que o enlace é compartilhado por Bob e mais quatro usuários. Bob usa instâncias paralelas de HTTP não persistente, e os outros quatro usuários usam HTTP não persistente sem downloads paralelos.
- As conexões paralelas de Bob ajudam a acessar páginas Web mais rapidamente? Por quê? Por que não?
 - Se cinco usuários abrirem cinco instâncias paralelas de HTTP não persistente, então as conexões paralelas de Bob ainda seriam úteis? Por quê? Por que não?
12. Escreva um programa TCP simples para um servidor que aceite linhas de entrada de um cliente e envie as linhas para a saída padrão do servidor. (Você pode fazer isso modificando o programa TCPServer.java no texto.) Compile e execute seu programa. Em qualquer outra máquina que contenha um browser Web, defina o servidor proxy no browser para a máquina que está executando seu programa servidor e também configure o número de porta adequadamente. Seu browser deverá agora enviar suas mensagens.

gens de requisição GET a seu servidor, e este deverá apresentar as mensagens em sua saída padrão. Use essa plataforma para determinar se seu browser gera mensagens GET condicionais para objetos que estão em caches locais.

13. Qual é a diferença entre MAIL FROM: em SMTP e FROM: na mensagem de correio?
14. Como o SMTP marca o final de um corpo de mensagem? E o HTTP? O HTTP pode usar o mesmo método que o SMTP para marcar o fim de um corpo de mensagem? Explique.
15. Leia o RFC 5321 para SMTP. O que significa MTA? Considere a seguinte mensagem spam recebida (modificada de um spam verdadeiro). Admitindo que o criador desse spam seja malicioso e que os hospedeiros sejam honestos, identifique o hospedeiro malicioso que criou essa mensagem spam.

```
From - Fri Nov 07 13:41:30 2008
Return-Path: tennis5@pp33head.com
Received: from barmail.cs.umass.edu
(barmail.cs.umass.edu [128.119.240.3])
by cs.umass.edu
(8.13.1/8.12.6) for hg@cs.umass.edu;
Fri, 7 Nov 2008
13:27:10 -0500
Received: from asusus-4b96 (localhost
[127.0.0.1]) by barmail.cs.umass.edu
(Spam Firewall) for
<hg@cs.umass.edu>; Fri, 7 Nov 2008 13:27:
07 -0500 (EST)
Received: from asusus-4b96
([58.88.21.177]) by
barmail.cs.umass.edu for <hg@cs.umass.
edu>; Fri,
07 Nov 2008 13:27:07 -0500 (EST)
Received: from [58.88.21.177] by
Inbnd55.exchangedddd.com; Sat, 8 Nov
2008 01:27:07 +0700
From: "Jonny"<tennis5@pp33head.com>
To: <hg@cs.umass.edu>
Subject: How to secure your savings
```

16. Leia o RFC do POP3 [RFC 1939]. Qual é a finalidade do comando UIDL do POP3?
17. Considere acessar seu e-mail com POP3.

- a. Suponha que você configure seu cliente de correio POP para funcionar no modo ler-e-apagar. Conclua a seguinte transação:

```
C: list
S: 1 498
S: 2 912
S: .
```

```
C: retr 1
S: blah blah ...
S: .....blah
S: .
?
?
```

- b. Suponha que você configure seu cliente de correio POP para funcionar no modo ler-e-guardar. Conclua a seguinte transação:

```
C: list
S: 1 498
S: 2 912
S: .
C: retr 1
S: blah blah ...
S: .....blah
S: .
?
?
```

- c. Suponha que você configure seu cliente de correio POP para funcionar no modo ler-e-guardar. Usando sua solução na parte (b), suponha que você recupere as mensagens 1 e 2, saia do POP e então, 5 minutos mais tarde, acesse novamente o POP para obter um novo e-mail. Imagine que nenhuma outra mensagem foi enviada nesse intervalo. Elabore um transcript dessa segunda sessão POP.

18. a. O que é um banco de dados whois?
- b. Use vários bancos de dados whois da Internet para obter os nomes de dois servidores DNS. Cite quais bancos de dados whois você utilizou.
- c. Use nslookup em seu hospedeiro local para enviar consultas DNS a três servidores de nomes: seu servidor DNS local e os dois servidores DNS que encontrou na parte (b). Tente consultar registros dos tipos A, NS e MX. Faça um resumo do que encontrou.
- d. Use nslookup para encontrar um servidor Web que tenha vários endereços IP. O servidor Web de sua instituição (escola ou empresa) tem vários endereços IP?
- e. Use o banco de dados whois ARIN para determinar a faixa de endereços IP usados por sua universidade.
- f. Descreva como um invasor pode usar bancos de dados whois e a ferramenta nslookup para fazer o reconhecimento de uma instituição antes de lançar um ataque.
- g. Discuta por que bancos de dados whois devem estar disponíveis publicamente.

19. Neste problema, utilizamos a ferramenta funcional *dig* disponível em hospedeiros Unix e Linux para explorar a hierarquia dos servidores DNS. Lembre de que, na Figura 2.21, um servidor DNS de nível superior na hierarquia do DNS delega uma consulta DNS para um servidor DNS de nível inferior na hierarquia enviando de volta ao cliente DNS o nome daquele servidor DNS de nível inferior. Em primeiro lugar, leia a *man page* sobre a ferramenta *dig* e responda às seguintes questões:
- Iniciando com o servidor DNS raiz (de um dos servidores raiz [a-m].root-servers.net), construa uma sequência de consultas para o endereço IP para seu servidor Web de departamento utilizando o *dig*. Mostre a relação de nomes de servidores DNS na cadeia de delegação ao responder à sua consulta.
 - Repita o item "a" com vários sites da Internet populares, como google.com, yahoo.com ou amazon.com.
20. Suponha que você consiga acessar os caches nos servidores DNS locais do seu departamento. Você é capaz de propor uma maneira de determinar, em linhas gerais, os servidores Web (fora de seu departamento) que são mais populares entre os usuários do seu departamento? Explique.
21. Suponha que seu departamento possua um servidor DNS local para todos os computadores do departamento. Você é um usuário comum (ou seja, não é um administrador de rede/sistema). Você consegue encontrar um modo de determinar se um site da Internet externo foi muito provavelmente acessado de um computador do seu departamento alguns segundos atrás? Explique.
22. Considere um arquivo de distribuição de $F = 15$ Gbits para N pares. O servidor possui um taxa de upload de $u_s = 30$ Mbps e cada par possui uma taxa de download de $d_i = 2$ Mbps e uma taxa de upload de u_i . Para $N = 10, 100$ e 1.000 e $u = 300$ Kbps, 700 Kbps e 2 Mbps, prepare um gráfico apresentando o tempo mínimo de distribuição para cada uma das combinações de N e u para o modo cliente-servidor e para o modo distribuição P2P.
23. Considere distribuir um arquivo de F bits para N pares utilizando uma arquitetura cliente-servidor. Admita um modelo fluido no qual o servidor pode transmitir simultaneamente para diversos pares, a diferentes taxas, desde que a taxa combinada não ultrapasse u_s .
- Suponha que $u_s/N \leq d_{\min}$. Especifique um esquema de distribuição que possua o tempo de distribuição de NF/u_s .
 - Suponha que $u_s/N \geq d_{\min}$. Especifique um esquema de distribuição que possua o tempo de distribuição de F/d_{\min} .
 - Conclua que o tempo mínimo de distribuição é, geralmente, dado por $\max\{NF/u_s, F/d_{\min}\}$.
24. Considere distribuir um arquivo de F bits para N pares utilizando uma arquitetura P2P. Admita um modelo fluido e que d_{\min} é muito grande, de modo que a largura de banda do download do par nunca é um gargalo.
- Suponha que $us \leq (u_s + u_1 + \dots + u_N)/N$. Especifique um esquema de distribuição que possua o tempo de distribuição de F/u_s .
 - Suponha que $u_s \geq (u_s + u_1 + \dots + u_N)/N$. Especifique um esquema de distribuição que possua o tempo de distribuição de $NF/(u_s + u_1 + \dots + u_N)$.
 - Conclua que o tempo mínimo de distribuição é, geralmente, dado por $\max\{F/u_s, NF/(u_s + u_1 + \dots + u_N)\}$.
25. Considere uma rede de sobreposição com N pares ativos, sendo que cada dupla de pares possua uma conexão TCP. Além disso, suponha que as conexões TCP passem por um total de M roteadores. Quantos nós e arestas há na rede de sobreposição correspondente?
26. Suponha que Bob tenha entrado no BitTorrent, mas ele não quer fazer o upload de nenhum dado para qualquer outro par (denominado carona).
- Bob alega que consegue receber uma cópia completa do arquivo compartilhado pelo grupo. A alegação de Bob é possível? Por quê?
 - Bob alega que ele pode "pegar carona" de um modo mais eficiente usando um conjunto de diversos computadores (com endereços IP distintos) no laboratório de informática de seu departamento. Como ele pode fazer isso?
27. Neste problema, queremos descobrir a eficiência de um sistema de compartilhamento de arquivo P2P semelhante ao BitTorrent. Considere dois pares, Bob e Alice. Eles entram em um torrent com M pares no total (incluindo Bob e Alice) que estão compartilhando um arquivo que consiste em N blocos. Admita que em um tempo específico t , os blocos que um par possui são escolhidos aleatoriamente a partir de todos os N blocos, e nenhum par possui todos os N blocos. Responda às seguintes questões:
- Qual é a probabilidade de Bob ter todos os blocos de Alice, sabendo que os números de blocos que Bob e Alice têm são representados por n_b e n_a ?
 - Remova parte do condicionamento do item "a" para descobrir a probabilidade de Bob ter os mesmos blocos de Alice, sabendo que Alice possui n_a blocos.
 - Suponha que cada par no BitTorrent tenha 5 vizinhos. Qual é a probabilidade de Bob ter dados que sejam de interesse de pelo menos um dos cinco vizinhos?

28. No exemplo de DHT circular na Seção 2.6.2, suponha que o par 3 descobriu que o par 5 saiu. Como o par 3 atualiza informações sobre o estado de seu sucessor?
29. No exemplo de DHT circular na Seção 2.6.2, suponha que um novo par 6 queira entrar no DHT, sabendo, inicialmente, o endereço IP do par 15. Quais passos são tomados?
30. Considere um DHT circular com identificadores de nós e de chave na faixa [0, 63]. Suponha que haja oito pares com identificadores 0, 8, 16, 24, 32, 40, 48 e 56.
- Suponha que cada par possa ter um par no atalho. Para cada um dos oito atalhos, determine seu par no atalho para que o número de mensagens enviadas para qualquer consulta (iniciando em qualquer par) seja reduzido.
 - Repita o item "a", mas permita que cada par tenha dois pares no atalho.
31. Como um número inteiro em $[0, 2^n - 1]$ pode ser expresso como um número binário de n bit em um DHT, cada chave pode ser expressa como $k = (k_0, k_1, \dots, k_{n-1})$, e cada identificador de par pode ser expresso como $p = (p_0, p_1, \dots, p_{n-1})$. Vamos, agora, definir a distância XOR entre a chave k e o par p como

$$d(k, p) = \sum_{j=0}^{n-1} |k_j - p_j| 2^j$$

Descreva como essa métrica pode ser usada para determinar duplas (chave, valor) para pares. (Para aprender mais sobre como construir um DHT eficiente usando essa métrica natural, consulte [Maymounkov, 2002], no qual o DHT Kademia é descrito.)

32. Considere uma versão generalizada do esquema descrito no problema acima. Em vez de usar números binários, vamos tratar os identificadores de chave e de par como números de base b , sendo $b > 2$, e

então usamos a métrica do problema anterior para criar um DHT (com 2 substituído b). Compare esse DHT baseado nos números de base b com o DHT baseado nos números binários. Na pior das hipóteses, qual DHT gera mais mensagens por consulta? Por quê?

33. Como os DHTs são redes de sobreposição, eles não se adequam necessariamente bem à rede física de sobreposição no sentido de que dois pares vizinhos podem estar fisicamente muito distantes; por exemplo, um par poderia estar na Ásia e seu vizinho, na América do Norte. Se atribuirmos identificadores aleatória e uniformemente para pares recém-unidos, esse esquema de atribuição causaria essa incompatibilidade? Explique. E como tal incompatibilidade afetaria o desempenho do DHT?
34. Instale e compile os programas Java TCPClient e UDPClient em um hospedeiro e TCPServer e UDPServer em outro.
- Suponha que você execute TCPClient antes de executar TCPServer. O que acontece? Por quê?
 - Imagine que você execute UDPClient antes de UDPServer. O que acontece? Por quê?
 - O que acontece se você usar números de porta diferentes para os lados cliente e servidor?

35. Suponha que, em UDPClient.java, a linha
- ```
DatagramSocket clientSocket = new DatagramSocket();
```
- seja substituída por
- ```
DatagramSocket clientSocket = new DatagramSocket(5432);
```
- Será necessário mudar UDPServer.java? Quais são os números de porta para os sockets em UDPClient e UDPServer? Quais eram esses números antes dessa mudança?



Questões dissertativas

- Na sua opinião, por que as aplicações de compartilhamento de arquivos P2P são tão populares? Será por que distribuem música e vídeo gratuitamente (o que é legalmente discutível) ou por que seu número imenso de servidores atende eficientemente uma demanda maciça por megabytes? Ou será pelas duas razões?
- Leia o artigo "The Darknet and the Future of Content Distribution" de Biddle, England, Peinado e Willman [Biddle, 2003]. Você concorda com a opinião dos autores? Por quê? Por que não?
- Sites de comércio eletrônico e outros sites Web frequentemente têm bancos de dados "de apoio". Como servidores HTTP se comunicam com esses bancos de dados?
- Como você pode configurar seu browser para cache local? Que opções de cache você tem?
- Você pode configurar seu browser para abrir várias conexões simultâneas com um site Web? Quais são as vantagens e as desvantagens de ter um grande número de conexões TCP simultâneas?

6. Vimos que sockets TCP da Internet tratam os dados que estão sendo enviados como uma cadeia de bytes, mas que sockets UDP reconhecem fronteiras de mensagens. Cite uma vantagem e uma desvantagem da API orientada para bytes em relação à API que reconhece e preserva explicitamente as fronteiras das mensagens definidas por aplicações.
7. O que é o servidor Web Apache? Quanto custa? Que funcionalidade tem atualmente?
8. Muitos clientes BitTorrent utilizam DHTs para criar um rastreador distribuído. Para esses DHTs, qual é a "chave" e qual é o "valor"?
9. Imagine que as organizações responsáveis pela padronização da Web decidam modificar a convenção de nomeação de modo que cada objeto seja nomeado e referenciado por um nome exclusivo que independa de localização (um URN). Discuta algumas questões que envolveriam tal modificação.
10. Há empresas distribuindo transmissões televisivas ao vivo por meio da Internet hoje? Se sim, essas empresas estão usando arquiteturas cliente-servidor e P2P?
11. As empresas, hoje, estão oferecendo um serviço de vídeo ao vivo através da Internet usando uma arquitetura P2P?
12. Como o Skype provê um serviço PC para telefone a vários países de destino?
13. Quais são os clientes mais populares do BitTorrent atualmente?

Tarefas de programação de sockets

Tarefa 1: servidor Web multithread

Ao final desta tarefa de programação, você terá desenvolvido, em Java, um servidor Web multithread, que seja capaz de atender várias requisições em paralelo. Você implementará a versão 1.0 do HTTP como definida no RFC 1945.

O HTTP/1.0 cria uma conexão TCP separada para cada par requisição/resposta. Cada uma dessas conexões será manipulada por um thread. Haverá também um thread principal, no qual o servidor ficará à escuta de clientes que quiserem estabelecer conexões. Para simplificar o trabalho de programação, desenvolveremos a codificação em dois estágios. No primeiro estágio, você escreverá um servidor multithread que simplesmente apresenta o conteúdo da mensagem de requisição HTTP que recebe. Depois que esse programa estiver executando normalmente, você adicionará a codificação necessária para gerar uma resposta apropriada.

Ao desenvolver a codificação, você poderá testar seu servidor com um browser Web. Mas lembre-se de que você não estará atendendo através da porta padrão 80, portanto, precisará especificar o número de porta dentro do URL que der a seu browser. Por exemplo, se o nome de seu hospedeiro for host.someschool.edu, seu servidor estiver à escuta na porta 6789 e você quiser obter o arquivo index.html, então deverá especificar o seguinte URL dentro do browser:

<http://host.someschool.edu:6789/index.html>

Quando seu servidor encontrar um erro, deverá enviar uma mensagem de resposta com uma fonte HTML adequada, de modo que a informação de erro

seja apresentada na janela do browser. Você pode encontrar mais detalhes sobre esta tarefa, assim como trechos importantes em código java no site <http://www.aw.com/kurose.br>

Tarefa 2: cliente de correio

Nesta tarefa, você desenvolverá um agente de usuário de correio em Java com as seguintes características:

Que provê uma interface gráfica para o remetente com campos para o servidor de correio local, para o endereço de e-mail do remetente, para o endereço de e-mail do destinatário, para o assunto da mensagem e para a própria mensagem.

Que estabelece uma conexão TCP entre o cliente de correio e o servidor de correio local. Que envia comandos SMTP para o servidor de correio local. Que recebe e processa comandos SMTP do servidor de correio local.

Esta será a aparência de sua interface:



Você desenvolverá o agente de usuário de modo que ele envie uma mensagem de e-mail para no máximo um destinatário por vez. Além disso, o agente de usuário admitirá que a parte de domínio do endereço de e-mail do destinatário será o nome canônico do servidor SMTP do destinatário. (O agente de usuário não realizará uma busca no DNS para um registro MX; portanto, o remetente deverá fornecer o nome real do servidor de correio.)

Tarefa 3: UDP Pinger Lab

Neste laboratório, você implementará um cliente e um servidor Ping simples, em UDP. A funcionalidade que esses programas oferecem é similar à do programa Ping padronizado, disponível em sistemas operacionais modernos. O Ping padronizado funciona enviando à Internet uma mensagem ECHO do Protocolo de Mensagens de Controle da Internet (ICMP), que a máquina remota devolve ao remetente como se fosse um eco. Então, o remetente pode

determinar o tempo de viagem de ida e volta entre ele mesmo e o computador para o qual enviou o Ping.

O Java não provê nenhuma funcionalidade para enviar ou receber mensagens ICMP e é por isso que neste laboratório você implementará o Ping na camada de aplicação com sockets e mensagens UDP padronizados.

Tarefa 4: servidor Proxy Web

Neste laboratório, você desenvolverá um servidor proxy Web simples que também poderá fazer cache de páginas Web. Esse servidor aceitará uma mensagem GET de um browser, transmitirá essa mensagem ao servidor Web destinatário, receberá a mensagem de resposta HTTP do servidor destinatário e transmitirá a mensagem de resposta ao browser. Esse servidor proxy é muito simples: entende apenas requisições GET simples. Contudo, pode manipular todos os tipos de objetos; não somente páginas HTML, mas também imagens.



Wireshark Labs

Você encontrará detalhes completos, em inglês, sobre os Wireshark Labs no site www.aw.com/kurose_br.

Wireshark Lab: HTTP

Como já tivemos uma primeira experiência com o analisador de pacotes Wireshark no Lab 1, estamos prontos para usar o Wireshark para investigar protocolos em operação. Neste laboratório, exploraremos diversos aspectos do protocolo HTTP: a interação básica GET/resposta, formatos de mensagens HTTP, extração de grandes arquivos HTML, extração de arquivos HTML com URLs inseridos, conexões persistentes e não persistentes e autenticação e segurança do HTTP.

Wireshark Lab: DNS

Neste laboratório, examinaremos mais de perto o lado cliente do DNS, o protocolo que transforma nomes

de hospedeiros da Internet em endereços IP. Lembre-se de que, na Seção 2.5, dissemos que o papel desempenhado pelo cliente no DNS é relativamente simples — um cliente envia uma consulta a seu servidor de nomes local e recebe uma resposta. Mas há muita coisa que pode acontecer nos bastidores e que é invisível para os clientes DNS enquanto os servidores DNS hierárquicos se comunicam uns com os outros para resolver a consulta DNS do cliente de modo recursivo ou iterativo. Do ponto de vista do cliente DNS, todavia, o protocolo é bastante simples — é formulada uma consulta ao servidor de nomes local e é recebida uma resposta deste. Nesse laboratório observamos o DNS em ação.

Entrevista

Bram Cohen

Bram Cohen é Cientista-chefe e cofundador da BitTorrent, Inc. e criador do protocolo de distribuição de arquivo P2P BitTorrent. Bram é também o cofundador da CodeCon e coautor do Codeville. Antes de criar o BitTorrent, Bram trabalhou no MojoNation, que permitia que as pessoas separassem arquivos confidenciais em blocos codificados e distribuissem essas partes para outros computadores que rodavam o software MojoNation. Esse conceito serviu de inspiração para Bram desenvolver o BitTorrent. Antes do MojoNation, Bram era um grande profissional da área de informática, trabalhando para várias empresas de Internet do meio ao fim dos anos 90. Ele cresceu em Nova York, se formou na Stuyvesant High School e frequentou a Universidade de Buffalo.



Como surgiu a ideia de desenvolver o BitTorrent?

Eu adquiri experiência criando redes (protocolos acima TCP/UDP), e implementar enxames parecia o problema irresoluto mais interessante da época, então resolvi trabalhar nisso.

O cálculo por trás do núcleo do BitTorrent é simples: Existe muita capacidade para upload lá fora. Muitas outras pessoas também fizeram essa mesma observação. Mas implementar algo que pudesse lidar com a logística envolvida é um outro problema.

Quais foram os aspectos mais desafiadores ao desenvolver o BitTorrent?

A parte principal foi acertar todo o projeto e a estrutura do protocolo. Assim que estivesse funcionando, sua implementação era "uma mera questão de programação". Com relação à implementação, de longe a parte mais difícil foi implementar um sistema confiável. Ao lidar com pares desconhecidos, você tem de imaginar que, a qualquer momento, qualquer um deles pode fazer qualquer coisa e ter um tipo de resposta estabelecida para todos os casos extremos. Eu tinha de ficar reescrevendo uma longa seção sobre o BitTorrent quando o estava criando pela primeira vez porque apareceram novos problemas e o projeto geral se tornou mais claro.

Inicialmente, como as pessoas descobriram o BitTorrent?

Normalmente as pessoas descobrem o BitTorrent fazendo o seu download. Elas queriam um certo tipo de conteúdo, que era encontrado somente usando o BitTorrent, então o baixavam. Um editor resolveu usar o BitTorrent porque simplesmente não tinha a largura de banda para distribuir seu conteúdo de outra maneira.

Comente sua opinião sobre as ações legais da RIAA e da MPAA contra as pessoas que utilizam programas de compartilhamento de arquivo, como o BitTorrent, para dividir filmes e música? Você já foi processado por desenvolver tecnologias que distribuem ilegalmente material protegido por direitos autorais?

A violação dos direitos autorais é ilegal, mas a tecnologia, não. Nunca fui processado, pois nunca me envolvi em nenhuma violação dos direitos autorais. Se você tem interesse em produzir tecnologia, precisa se agarrar a ela.

Você acha que, em um futuro próximo, outros sistemas de distribuição de arquivo podem substituir o BitTorrent? Por exemplo, a Microsoft pode incluir seu próprio protocolo de distribuição de arquivo em um lançamento futuro de um sistema operacional?

Pode haver protocolos no futuro, mas é improvável que os princípios fundamentais de como agrupar dados, esclarecidos no protocolo BitTorrent, mudem. A maneira mais provável de ocorrer uma substituição é se houver uma mudança em toda a estrutura da Internet em razão das proporções entre algumas das constantes fundamentais que mudam radicalmente à medida que a velocidade aumenta. Mas projeções para os próximos anos apenas reforçam ainda mais o modelo atual.

De forma geral, onde você vê a Internet liderando? Na sua opinião, quais são, ou serão, os desafios técnicos mais importantes? Você visualiza alguma nova "aplicação inovadora" que está por vir?

A Internet e os computadores em geral estão se tornando cada vez mais onipresentes. O iPod nano parece uma lembrancinha de festa, pois inevitavelmente um dia ele o será, em razão da baixa dos preços. O desafio técnico atual mais interessante é reunir o maior número de dados possível de todos os dispositivos conectados e disponibilizar esses dados de uma forma útil e acessível. Por exemplo, quase todos os aparelhos portáteis poderiam conter um GPS, e cada objeto que você possui, incluindo roupas, brinquedos, eletrodomésticos e mobília, poderiam informá-lo onde se encontram ao perdê-los e fazer um resumo completo sobre seu histórico atual, incluindo manutenção necessária, utilidade futura esperada, detecção de maus tratos etc. Você não só poderia obter informações sobre suas propriedades como também, digamos, a vida útil de um produto específico poderia ser determinada de forma precisa, e a coordenação com outras pessoas se tornaria mais fácil, além do simples, mas surpreendente, aperfeiçoamento de as pessoas poderem se encontrar facilmente quando possuem um telefone celular.

Profissionalmente, alguém lhe serviu de inspiração? Em que sentidos?

Nenhuma parábola me vem à cabeça, mas os mitos do Vale do Silício foram algo que segui à risca.

Você tem algum conselho para os alunos ingressantes na área de rede/Internet?

Encontre algo que não esteja em alta no momento, mas que você ache que possam dar origem a coisas empolgantes e que seja, particularmente, interessante para você e comece a trabalhar nisso. Tente também obter experiência profissional na área em que deseja trabalhar. Experiências do mundo real ensinam você o que é importante no mundo real, e isso é algo que é sempre bem distorcido quando visto somente de dentro do ambiente acadêmico.



Capítulo 3

Camada de transporte



Posicionada entre as camadas de aplicação e de rede, a camada de transporte é uma peça central da arquitetura de rede em camadas. Ela desempenha o papel fundamental de fornecer serviços de comunicação diretamente aos processos de aplicação que rodam em hospedeiros diferentes. A abordagem pedagógica que adotamos neste capítulo é alternar entre discussões de princípios de camada de transporte e o modo como esses princípios são implementados em protocolos existentes; como de costume, daremos particular ênfase aos protocolos da Internet, em especial aos protocolos de camada de transporte TCP e UDP.

Começaremos discutindo a relação entre as camadas de transporte e de rede, preparando o cenário para o exame da primeira função importante da camada de transporte — ampliar o serviço de entrega da camada de rede entre dois sistemas finais para um serviço de entrega entre dois processos de camada de aplicação que rodam nos sistemas finais. Ilustraremos essa função quando abordarmos o UDP, o protocolo de transporte não orientado para conexão da Internet.

Então retornaremos aos princípios e enfrentaremos um dos problemas mais fundamentais de redes de computadores — como duas entidades podem se comunicar de maneira confiável por um meio que pode perder e corromper dados. Mediante uma série de cenários cada vez mais complicados (e realistas!), construiremos um conjunto de técnicas que os protocolos de transporte utilizam para resolver esse problema. Então, mostraremos como esses princípios estão incorporados no TCP, o protocolo de transporte orientado para conexão da Internet.

Em seguida, passaremos para um segundo problema fundamentalmente importante em redes — o controle da taxa de transmissão de entidades de camada de transporte para evitar ou se recuperar de congestionamentos dentro da rede. Consideraremos as causas e consequências do congestionamento, bem como técnicas de controle de congestionamento comumente usadas. Após adquirir um sólido entendimento das questões que estão por trás do controle de congestionamento, estudaremos como o TCP o aborda.

3.1 Introdução e serviços de camada de transporte

Nos dois capítulos anteriores, citamos o papel da camada de transporte e os serviços que ela fornece. Vamos revisar rapidamente o que já aprendemos sobre a camada de transporte.

Um protocolo de camada de transporte fornece **comunicação lógica** entre processos de aplicação que rodam em hospedeiros diferentes. *Comunicação lógica* nesse contexto significa que, do ponto de vista de uma aplicação, tudo se passa como se os hospedeiros que rodam os processos estivessem conectados diretamente; na

verdade, esses hospedeiros poderão estar em lados opostos do planeta, conectados por numerosos roteadores e uma ampla variedade de tipos de enlace. Processos de aplicação usam a comunicação lógica provida pela camada de transporte para enviar mensagens entre si, livres da preocupação dos detalhes da infraestrutura física utilizada para transportar essas mensagens. A Figura 3.1 ilustra a noção de comunicação lógica.

Nela, vemos que protocolos de camada de transporte são implementados nos sistemas finais, mas não em roteadores de rede. No lado remetente, a camada de transporte converte as mensagens que recebe de um

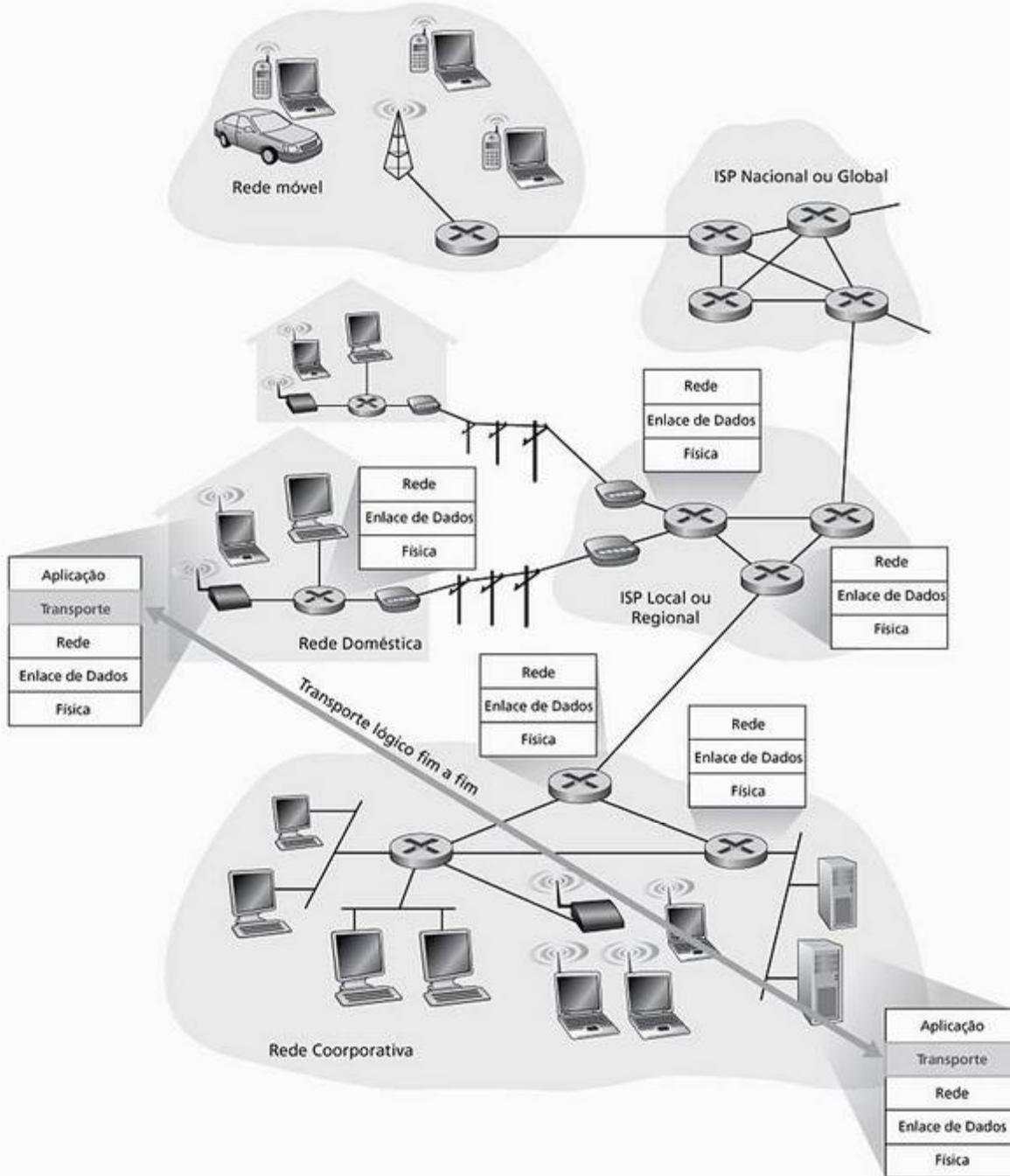


Figura 3.1 A camada de transporte fornece comunicação lógica, e não física, entre processos de aplicações. Pode haver mais de um protocolo de camada de transporte disponível para aplicações de rede. Por exemplo, a Internet tem dois protocolos — TCP e UDP. Cada um deles provê um conjunto diferente de serviços de camada de transporte à aplicação que o está chamando

processo de aplicação remetente em pacotes de camada de transporte, denominados **segmentos** de camada de transporte na terminologia da Internet. Isso é (possivelmente) feito fragmentando-se as mensagens da aplicação em pedaços menores e adicionando-se um cabeçalho de camada de transporte a cada pedaço para criar o segmento de camada de transporte. A camada de transporte, então, passa o segmento para a camada de rede no sistema final remetente, onde ele é encapsulado em um pacote de camada de rede (um datagrama) e enviado ao destinatário. É importante notar que roteadores de rede agem somente nos campos de camada de rede do datagrama; isto é, não examinam os campos do segmento de camada de transporte encapsulado com o datagrama. No lado destinatário, a camada de rede extraí do datagrama o segmento de camada de transporte e passa-o para a camada de transporte. Em seguida, essa camada processa o segmento recebido, disponibilizando os dados para a aplicação destinatária.

3.1.1 Relação entre as camadas de transporte e de rede

Lembre-se de que a camada de transporte se situa logo acima da camada de rede na pilha de protocolos. Enquanto um protocolo de camada de transporte fornece comunicação lógica entre *processos* que rodam em hospedeiros diferentes, um protocolo de camada de rede fornece comunicação lógica entre *hospedeiros*. Essa distinção é sutil, mas importante. Vamos examiná-la com o auxílio de uma analogia com moradias.

Considere duas casas, uma na Costa Leste e outra na Costa Oeste dos Estados Unidos e que, em cada uma delas, há uma dúzia de crianças. As crianças da Costa Leste são primas das crianças da Costa Oeste e todas adoram escrever cartas umas para as outras — cada criança escreve a cada primo uma vez por semana e cada carta é entregue pelo serviço de correio tradicional dentro de um envelope separado. Assim, cada moradia envia 144 cartas por semana para a outra. (Essas crianças economizariam muito dinheiro se tivessem e-mail!) Em cada moradia há uma criança responsável pela coleta e distribuição da correspondência — Ann, na casa da Costa Oeste, e Bill, na da Costa Leste. Toda semana, Ann coleta a correspondência de seus irmãos e irmãs e a coloca no correio. Quando as cartas chegam à casa da Costa Oeste, também é Ann quem tem a tarefa de distribuir a correspondência a seus irmãos e irmãs. Bill realiza o mesmo trabalho na casa da Costa Leste.

Nesse exemplo, o serviço postal provê uma comunicação lógica entre as duas casas — ele movimenta a correspondência de uma casa para outra, e não de uma pessoa para outra. Por outro lado, Ann e Bill proveem comunicação lógica entre os primos — eles coletam e entregam a correspondência de seus irmãos e irmãs. Note que, da perspectiva dos primos, Ann e Bill são o serviço postal, embora sejam apenas uma parte do sistema (a parte do sistema final) do processo de entrega fim a fim. Esse exemplo das moradias é uma analogia interessante para explicar como a camada de transporte se relaciona com a camada de rede:

mensagens de aplicação = cartas em envelopes

processos = primos

hospedeiros (também denominados sistemas finais) = casas

protocolo de camada de transporte = Ann e Bill

protocolo de camada de rede = serviço postal (incluindo os carteiros)

Continuando com essa analogia, observe que Ann e Bill fazem todo o seu trabalho dentro de suas respectivas casas; eles não estão envolvidos, por exemplo, com a classificação da correspondência em nenhuma central intermediária dos correios ou com o transporte da correspondência de uma central a outra. De maneira semelhante, protocolos de camada de transporte moram nos sistemas finais, onde movimentam mensagens de processos de aplicação para a borda da rede (isto é, para a camada de rede) e vice-versa, mas não interferem no modo como as mensagens são movimentadas dentro do núcleo da rede. Na verdade, como ilustrado na Figura 3.1, roteadores intermediários não reconhecem nenhuma informação que a camada de transporte possa ter anexado às mensagens da aplicação nem agem sobre ela.

Prosseguindo com nossa saga familiar, suponha agora que, quando Ann e Bill saem de férias, outro par de primos — digamos, Susan e Harvey — substitua-os e encarregue-se da coleta interna da correspondência e de sua entrega. Infelizmente para as duas famílias, eles não desempenham essa tarefa do mesmo modo que Ann e Bill.

Por serem crianças mais novas, Susan e Harvey recolhem e entregam a correspondência com menos frequência e, ocasionalmente, perdem cartas (que às vezes acabam mastigadas pelo cão da família). Assim, o par de primos Susan e Harvey não provê o mesmo conjunto de serviços (isto é, o mesmo modelo de serviço) oferecido por Ann e Bill. De uma maneira análoga, uma rede de computadores pode disponibilizar vários protocolos de transporte, em que cada um oferece um modelo de serviço diferente às aplicações.

Os possíveis serviços que Ann e Bill podem fornecer são claramente limitados pelos possíveis serviços que os correios fornecem. Por exemplo, se o serviço postal não estipula um prazo máximo para entregar a correspondência entre as duas casas (digamos, três dias), então não há nenhuma possibilidade de Ann e Bill definirem um atraso máximo para a entrega da correspondência entre qualquer par de primos. De maneira semelhante, os serviços que um protocolo de transporte pode fornecer são frequentemente limitados pelo modelo de serviço do protocolo subjacente da camada de rede. Se o protocolo de camada de rede não puder dar garantias contra atraso ou garantias de largura de banda para segmentos de camada de transporte enviados entre hospedeiros, então o protocolo de camada de transporte não poderá dar essas mesmas garantias para mensagens de aplicação enviadas entre processos.

No entanto, certos serviços podem ser oferecidos por um protocolo de transporte mesmo quando o protocolo de rede subjacente não oferece o serviço correspondente na camada de rede. Por exemplo, como veremos neste capítulo, um protocolo de transporte pode oferecer serviço confiável de transferência de dados a uma aplicação mesmo quando o protocolo subjacente da rede não é confiável, isto é, mesmo quando o protocolo de rede perde, embaralha ou duplica pacotes. Como outro exemplo (que exploraremos no Capítulo 8, quando discutirmos segurança de rede), um protocolo de transporte pode usar criptografia para garantir que as mensagens da aplicação não sejam lidas por intrusos mesmo quando a camada de rede não puder garantir o sigilo de segmentos de camada de transporte.

3.1.2 Visão geral da camada de transporte na Internet

Lembre-se de que a Internet — e, de maneira mais geral, a rede TCP/IP — disponibiliza dois protocolos de transporte distintos para a camada de aplicação. Um deles é o **UDP** (User Datagram Protocol — Protocolo de Datagrama de Usuário), que provê à aplicação solicitante um serviço não confiável, não orientado para conexão. O segundo desses protocolos é o **TCP** (Transmission Control Protocol — Protocolo de Controle de Transmissão), que provê à aplicação solicitante um serviço confiável, orientado para conexão. Ao projetar uma aplicação de rede, o criador da aplicação deve especificar um desses dois protocolos de transporte. Como vimos nas seções 2.7 e 2.8, o desenvolvedor da aplicação escolhe entre o UDP e o TCP ao criar sockets.

Para simplificar a terminologia, quando no contexto da Internet, faremos alusão ao pacote de camada de transporte como um *segmento*. Devemos mencionar, contudo, que a literatura da Internet (por exemplo, os RFCs) também se refere ao pacote de camada de transporte com/para TCP como um segmento, mas muitas vezes se refere ao pacote com/para UDP como um *datagrama*. Porém, essa mesma literatura também usa o termo *datagrama* para o pacote de camada de rede! Como este é um livro de introdução a redes de computadores, acreditamos que será menos confuso se nos referirmos a ambos os pacotes TCP e UDP como segmentos; reservaremos o termo *datagrama* para o pacote de camada de rede.

Antes de continuarmos com nossa breve apresentação do UDP e do TCP, é útil dizer algumas palavras sobre a camada de rede da Internet. (A camada de rede é examinada detalhadamente no Capítulo 4.) O protocolo de camada de rede da Internet tem um nome — IP, que quer dizer *Internet Protocol*. O IP provê comunicação lógica entre hospedeiros. O modelo de serviço do IP é um **serviço de entrega de melhor esforço**, o que significa que o IP faz o ‘melhor esforço’ para levar segmentos entre hospedeiros comunicantes, mas não dá nenhuma garantia. Em especial, o IP não garante a entrega de segmentos, a entrega ordenada de segmentos e tampouco a integridade dos dados nos segmentos. Por essas razões, ele é denominado um **serviço não confiável**. Mencionamos também neste livro que cada hospedeiro tem, no mínimo, um endereço de camada de rede, denominado endereço IP. Examinaremos endereçamento IP detalhadamente no Capítulo 4. Para este capítulo, precisamos apenas ter em mente que *cada hospedeiro tem um endereço IP*.

Agora que abordamos ligeiramente o modelo de serviço IP, vamos resumir os modelos de serviço providos por UDP e TCP. A responsabilidade fundamental do UDP e do TCP é ampliar o serviço de entrega IP entre dois sistemas finais para um serviço de entrega entre dois processos que rodam nos sistemas finais. A ampliação da entrega hospedeiro a hospedeiro para entrega processo a processo é denominada **multiplexação/demultiplexação de camada de transporte**. Discutiremos multiplexação e demultiplexação de camada de transporte na próxima seção. O UDP e o TCP também fornecem verificação de integridade ao incluir campos de detecção de erros nos cabeçalhos de seus segmentos. Esses dois serviços mínimos de camada de transporte — entrega de dados processo a processo e verificação de erros — são os únicos que o UDP fornece! Em especial, como o IP, o UDP é um serviço não confiável — ele não garante que os dados enviados por um processo cheguem (quando chegam!) intactos ao processo destinatário. O UDP será discutido detalhadamente na Seção 3.3.

O TCP, por outro lado, oferece vários serviços adicionais às aplicações. Primeiramente, e mais importante, ele provê **transferência confiável de dados**. Usando controle de fluxo, números de sequência, reconhecimentos e temporizadores (técnicas que exploraremos detalhadamente neste capítulo), o protocolo assegura que os dados sejam entregues do processo remetente ao processo destinatário corretamente e em ordem. Assim, o TCP converte o serviço não confiável do IP entre sistemas finais em um serviço confiável de transporte de dados entre processos. Ele também provê **controle de congestionamento**. O controle de congestionamento não é tanto um serviço fornecido à aplicação solicitante; é mais um serviço dirigido à Internet como um todo — para o bem geral. Em termos genéricos, o controle de congestionamento do TCP evita que qualquer outra conexão TCP abarrote os enlaces e comutadores entre hospedeiros comunicantes com uma quantidade excessiva de tráfego. Em princípio, o TCP permite que conexões TCP trafegando por um enlace de rede congestionado compartilhem em pé de igualdade a largura de banda daquele enlace. Isso é feito pela regulagem da taxa com a qual o lado remetente do TCP pode enviar tráfego para dentro da rede. O tráfego UDP, por outro lado, não é regulado. Uma aplicação que usa transporte UDP pode enviar tráfego à taxa que quiser, pelo tempo que quiser.

Um protocolo que fornece transferência confiável de dados e controle de congestionamento é, necessariamente, complexo. Precisaremos de várias seções para detalhar os princípios da transferência confiável de dados e do controle de congestionamento, bem como de seções adicionais para explicar o protocolo TCP. Esses tópicos são analisados nas seções 3.4 a 3.8. A abordagem escolhida neste capítulo é alternar entre princípios básicos e o protocolo TCP. Por exemplo, discutiremos primeiramente a transferência confiável de dados em âmbito geral e, em seguida, como o TCP fornece especificamente a transferência confiável de dados. De maneira semelhante, discutiremos inicialmente o controle de congestionamento em âmbito geral e, em seguida, como o TCP realiza o controle de congestionamento. Mas, antes de chegarmos a essa parte boa, vamos examinar, primeiramente, multiplexação/demultiplexação na camada de transporte.

3.2 Multiplexação e demultiplexação

Nesta seção, discutiremos multiplexação e demultiplexação na camada de transporte, isto é, a ampliação do serviço de entrega hospedeiro a hospedeiro provido pela camada de rede para um serviço de entrega processo a processo para aplicações que rodam nesses hospedeiros. Para manter a discussão em nível concreto, vamos examinar esse serviço básico da camada de transporte no contexto da Internet. Enfatizamos, contudo, que o serviço de multiplexação/demultiplexação é necessário para todas as redes de computadores.

No hospedeiro de destino, a camada de transporte recebe segmentos da camada de rede logo abaixo dela e tem a responsabilidade de entregar os dados desses segmentos ao processo de aplicação apropriado que roda no hospedeiro. Vamos examinar um exemplo. Suponha que você esteja sentado à frente de seu computador, descarregando páginas Web enquanto roda uma sessão FTP e duas sessões Telnet. Por conseguinte, você tem quatro processos de aplicação de rede em execução — dois processos Telnet, um processo FTP e um processo HTTP. Quando a camada de transporte em seu computador receber dados da camada de rede abaixo dela, precisará direcionar os dados recebidos a um desses quatro processos. Vamos ver agora como isso é feito.

Em primeiro lugar, lembre-se de que dissemos, nas seções 2.7 e 2.8, que um processo (como parte de uma aplicação de rede) pode ter um ou mais **sockets**, portas pelas quais dados passam da rede para o processo e do

processo para a rede. Assim, como mostra a Figura 3.2, a camada de transporte do hospedeiro destinatário na verdade não entrega dados diretamente a um processo, mas a um socket intermediário. Como, a qualquer dado instante, pode haver mais de um socket no hospedeiro destinatário, cada um tem um identificador exclusivo. O formato do identificador depende de o socket ser UDP ou TCP, como discutiremos em breve.

Agora, vamos considerar como um hospedeiro destinatário direciona à porta correta um segmento de camada de transporte que chega. Cada segmento de camada de transporte tem um conjunto de campos para essa finalidade. Na extremidade receptora, a camada de transporte examina esses campos para identificar a porta receptora e direcionar o segmento a esse socket. A tarefa de entregar os dados contidos em um segmento da camada de transporte à porta correta é denominada **demultiplexação**. O trabalho de reunir, no hospedeiro de origem, porções de dados provenientes de diferentes portas, encapsular cada porção de dados com informações de cabeçalho (que mais tarde serão usadas na demultiplexação) para criar segmentos, e passar esses segmentos para a camada de rede é denominada **multiplexação**. Note que a camada de transporte do hospedeiro que está no meio da Figura 3.2 tem de demultiplexar segmentos que chegam da camada de rede abaixo para os processos P1 ou P2 acima; isso é feito direcionando à porta correspondente do processo os dados contidos no segmento que está chegando. A camada de transporte desse hospedeiro também tem de juntar dados de saída dessas portas, formar segmentos de camada de transporte e passá-los à camada de rede. Embora tenhamos apresentado multiplexação e demultiplexação no contexto dos protocolos de transporte da Internet, é importante entender que essas operações estarão presentes sempre que um único protocolo em uma camada (na de transporte ou em qualquer outra) for usado por vários protocolos na camada mais alta seguinte.

Para ilustrar o serviço de demultiplexação, lembre-se da metáfora das moradias apresentada na seção anterior. Cada uma das crianças é identificada por seu nome próprio. Quando Bill recebe uma grande quantidade de correspondência do carteiro, realiza uma operação de demultiplexação ao examinar para quem as cartas estão endereçadas e, em seguida, entregar a correspondência a seus irmãos e irmãs. Ann realiza uma operação de multiplexação quando coleta as cartas de seus irmãos e irmãs e entrega a correspondência na agência do correio.

Agora que entendemos os papéis da multiplexação e da demultiplexação na camada de transporte, vamos examinar como isso é feito em um hospedeiro. Sabemos, pela nossa discussão anterior, que multiplexação na camada de rede requer (1) que as portas tenham identificadores exclusivos e (2) que cada segmento tenha campos especiais que indiquem a porta para a qual o segmento deve ser entregue. Esses campos especiais, ilustrados na Figura 3.3, são o **campo de número de porta da fonte** e o **campo de número de porta do destino**. (Os segmentos UDP e TCP têm outros campos também, que serão examinados nas seções subsequentes deste capítulo.) Cada número de porta é um número de 16 bits na faixa de 0 a 65535. Os números de porta entre 0 e 1023 são denominados **números de porta bem conhecidos**; eles são restritos, o que significa que estão reservados para utilização por protocolos de aplicação bem conhecidos, como HTTP (que usa a porta número 80) e FTP.

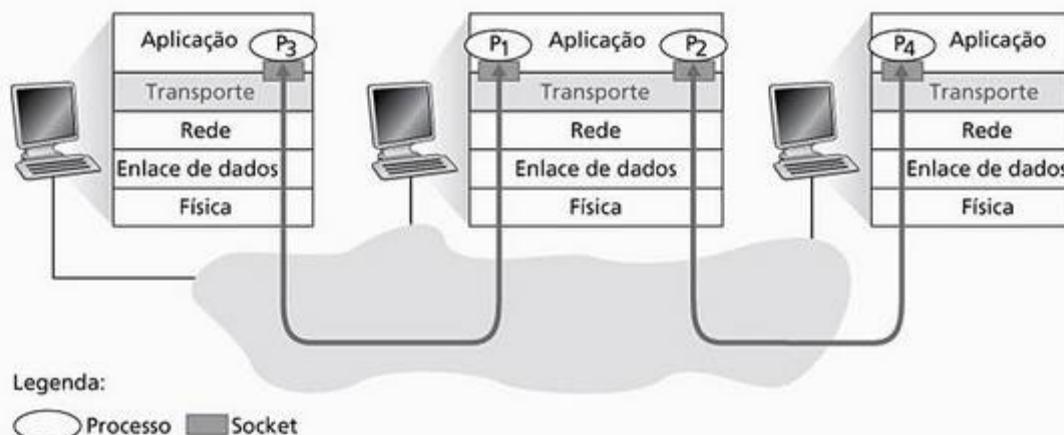


Figura 3.2 Multiplexação e demultiplexação na camada de transporte

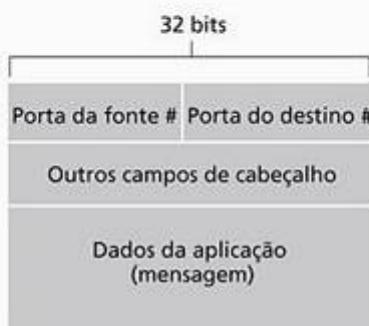


Figura 3.3 Campos de número de porta da fonte e do destino em um segmento de camada de transporte

(que usa a porta número 21). A lista dos números de porta bem conhecidos é apresentada no RFC 1700 e atualizada em <http://www.iana.org> [RFC 3232]. Quando desenvolvemos uma nova aplicação (como as desenvolvidas nas seções 2.7 a 2.8), devemos atribuir a ela um número de porta.

Agora já deve estar claro como a camada de transporte poderia implementar o serviço de demultiplexação: cada porta do hospedeiro pode receber um número designado; quando um segmento chega ao hospedeiro, a camada de transporte examina seu número de porta de destino e o direciona à porta correspondente. Então, os dados do segmento passam através da porta e entram no processo ligado a ela. Como veremos, é assim que o UDP faz demultiplexação. Todavia, veremos também que multiplexação/demultiplexação em TCP é ainda mais sutil.

Multiplexação e demultiplexação não orientadas para conexão

Lembre-se, na Seção 2.8, de que um programa em Java que roda em um hospedeiro pode criar uma porta UDP com a linha

```
DatagramSocket mySocket = new DatagramSocket();
```

Quando uma porta UDP é criada dessa maneira, a camada de transporte automaticamente designa um número de porta ao socket. Em especial, a camada de transporte designa um número de porta na faixa de 1024 a 65535 que não esteja sendo usado naquele instante por qualquer outra porta UDP no hospedeiro. Alternativamente, um programa Java poderia criar uma porta com a linha

```
DatagramSocket mySocket = new DatagramSocket(19157);
```

Nesse caso, a aplicação designa um número de porta específico — a saber, 19157 — à porta UDP. Se o desenvolvedor responsável por escrever o código da aplicação estivesse implementando o lado servidor de um ‘protocolo bem conhecido’, ele teria de designar o número de porta bem conhecido correspondente. O lado cliente da aplicação normalmente permite que a camada de transporte designe o número de porta automaticamente — e transparentemente — ao passo que o lado servidor da aplicação designa um número de porta específico.

Agora que os sockets UDP já têm seus números de porta designados, podemos descrever multiplexação/demultiplexação UDP com precisão. Suponha que um processo no hospedeiro A, cujo número de porta UDP é 19157, queira enviar uma porção de dados de aplicação a um processo cujo número de porta UDP seja 46428 no hospedeiro B. A camada de transporte no hospedeiro A cria um segmento de camada de transporte que inclui os dados de aplicação, o número da porta da fonte (19157), o número da porta de destino (46428) e mais outros dois valores (que serão discutidos mais adiante, mas que não são importantes para a discussão em curso). Então, a camada de transporte passa o segmento resultante para a camada de rede. Essa camada encapsula o segmento em um datagrama IP e faz uma tentativa de melhor esforço para entregar o segmento ao hospedeiro destinatário. Se o segmento chegar à máquina de destino B, a camada de destino no hospedeiro destinatário examinará o número da porta de destino no segmento (46428) e o entregará a seu socket identificado pelo número 46428. Note que a máquina B poderá estar rodando vários processos, cada um com sua própria porta UDP e número de porta

associado. À medida que segmentos UDP chegassem da rede, a máquina B direcionaria (demultiplexaria) cada segmento à porta apropriada examinando o número de porta de destino do segmento.

É importante notar que um socket UDP é totalmente identificado por uma tupla com dois elementos, consistindo em um endereço IP de destino e um número de porta de destino. Consequentemente, se dois segmentos UDP tiverem endereços IP de fonte e/ou números de porta de fonte diferentes, porém o mesmo endereço IP de destino e o mesmo número de porta de destino, eles serão direcionados ao mesmo processo de destino por meio do mesmo socket de destino.

É possível que agora você esteja imaginando qual é a finalidade do número de porta da fonte. Como mostra a Figura 3.4, no segmento A-B, o número de porta da fonte serve como parte de um “endereço de retorno” — quando B quer enviar um segmento de volta para A, a porta de destino no segmento B-A tomará seu valor do valor da porta de fonte do segmento A-B. (O endereço de retorno completo é o endereço IP e o número de porta de fonte de A). Como exemplo, lembre-se do programa servidor UDP que estudamos na Seção 2.8. Em `UDPServer.java`, o servidor usa um método para extrair o número de porta da fonte do segmento que recebe do cliente; então envia um novo segmento ao cliente, com o número de porta que extraiu servindo como o número de porta de destino desse novo segmento.

Multiplexação e demultiplexação orientadas para conexão

Para entender demultiplexação TCP, temos de examinar de perto sockets TCP e estabelecimento de conexão TCP. Há uma diferença sutil entre um socket UDP e um socket TCP: o socket TCP é identificado por uma tupla de quatro elementos: (endereço IP da fonte, número de porta da fonte, endereço IP de destino, número de porta do destino). Assim, quando um segmento TCP que vem da rede chega a um hospedeiro, este usa todos os quatro valores para direcionar (demultiplexar) o segmento para o socket apropriado. Em especial, e ao contrário do UDP, dois segmentos TCP chegando com endereços IP de fonte ou números de porta de fonte diferentes serão direcionados para dois sockets diferentes (com exceção de um TCP que esteja carregando a requisição de estabelecimento de conexão original). Para perceber melhor, vamos considerar novamente o exemplo de programação cliente-servidor TCP apresentado na Seção 2.7:

A aplicação servidor TCP tem um socket de entrada que espera requisições de estabelecimento de conexão vindas de clientes TCP (ver Figura 2.29) na porta número 6789.

O cliente TCP gera um segmento de estabelecimento de conexão com a linha

```
Socket clientSocket = new Socket("serverHostName", 6789);
```

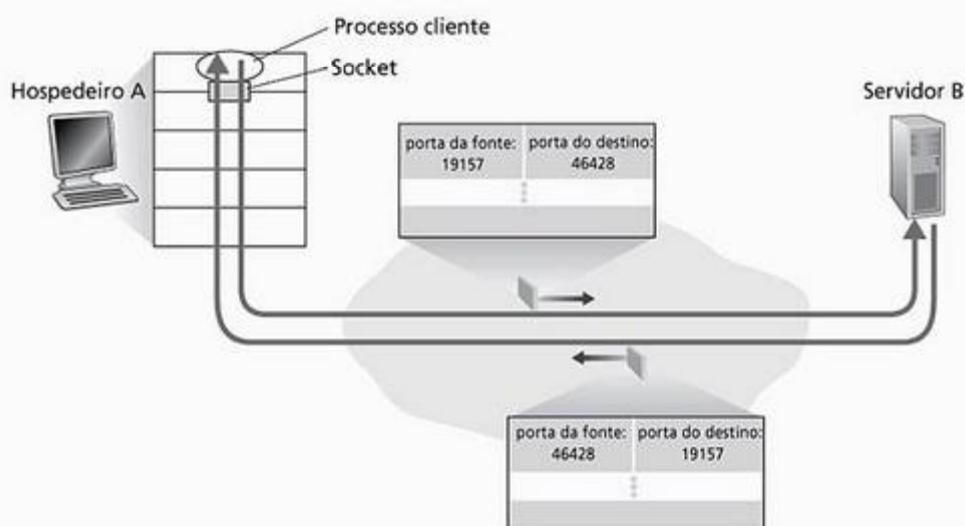


Figura 3.4 Campos de número de porta da fonte e do destino em um segmento de camada de transporte

Uma requisição de estabelecimento de conexão nada mais é do que um segmento TCP com número de porta de destino 6789 e um bit especial de estabelecimento de conexão marcado no cabeçalho TCP (que será discutido na Seção 3.5). O segmento inclui também um número de porta de fonte que foi escolhido pelo cliente. A linha acima cria ainda um socket TCP para o processo cliente, através do qual dados podem entrar e sair do processo cliente.

Quando o sistema operacional do computador que está rodando o processo servidor recebe o segmento de requisição de conexão que está chegando e cuja porta de destino é 6789, ele localiza o processo servidor que está esperando para aceitar uma conexão na porta número 6789. Então, o processo servidor cria um novo socket:

```
Socket connectionSocket = welcomeSocket.accept();
```

A camada de transporte no servidor também nota os quatro valores seguintes no segmento de requisição de conexão: (1) o número de porta da fonte no segmento, (2) o endereço IP do hospedeiro da fonte, (3) o número de porta do destino no segmento e (4) seu próprio endereço IP. O socket de conexão recém-criado é identificado por esses quatro valores; todos os segmentos subsequentes que chegarem, cuja porta da fonte, endereço IP da fonte, porta de destino e endereço IP de destino combinarem com esses quatro valores, serão demultiplexados para essa porta. Com a conexão TCP agora ativa, o cliente e o servidor podem enviar dados um para o outro.

O hospedeiro servidor pode suportar vários sockets TCP simultâneos, sendo cada um ligado a um processo e identificado por sua própria tupla de quatro elementos. Quando um segmento TCP chega ao hospedeiro, todos os quatro campos (endereço IP da fonte, porta da fonte, endereço IP de destino, porta de destino) são usados para direcionar (demultiplexar) o segmento para o socket apropriado.

A situação é ilustrada na Figura 3.5, na qual o hospedeiro C inicia duas sessões HTTP para o servidor B, e o hospedeiro A inicia uma sessão HTTP para o servidor B.

Os hospedeiros A e C e o servidor B possuem, cada um, seu próprio endereço IP exclusivo: A, C e B, respectivamente. O hospedeiro C atribui dois números diferentes (26145 e 7532) de porta da fonte às suas duas conexões HTTP. Como o hospedeiro A está escolhendo números de porta independentemente de C, ele poderia

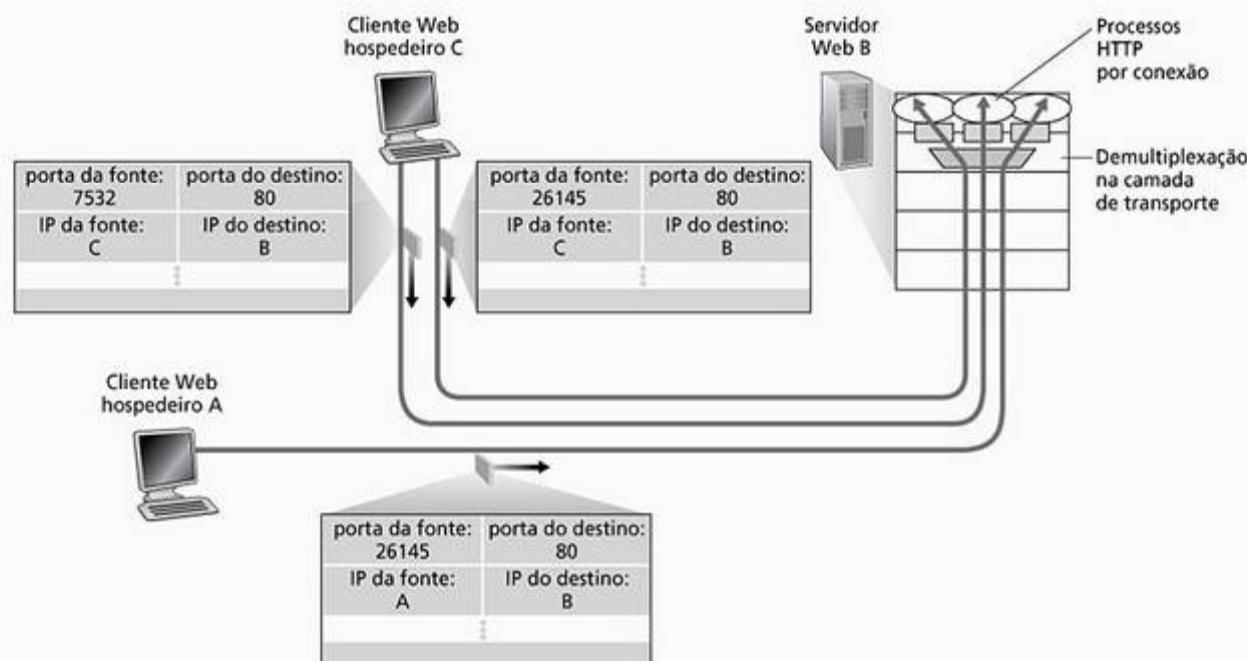


Figura 3.5 Dois clientes que usam o mesmo número de porta de destino (80) para se comunicar com a mesma aplicação de servidor Web

também atribuir um número de porta da fonte 26145 à sua conexão HTTP. Apesar disso, o servidor B ainda será capaz de demultiplexar corretamente as duas conexões que têm o mesmo número de porta de fonte, já que elas têm endereços IP de fonte diferentes.

Servidores Web e TCP

Antes de encerrar essa discussão, é instrutivo falar um pouco mais sobre servidores Web e como eles usam números de porta. Considere um hospedeiro rodando um servidor Web, tal como um Apache, na porta 80. Quando clientes (por exemplo, browsers) enviam segmentos ao servidor, todos os segmentos terão a porta de destino 80. Em especial, os segmentos que estabelecem a conexão inicial e os segmentos que carregam as mensagens de requisição HTTP, ambos terão a porta de destino 80. Como acabamos de descrever, o servidor distingue os segmentos dos diferentes clientes pelos endereços IP e pelos números de porta da fonte.

Como mostra a Figura 3.5, cada um desses processos tem seu próprio socket de conexão através do qual chegam requisições HTTP e são enviadas respostas HTTP. Mencionamos, contudo, que nem sempre existe uma correspondência unívoca entre sockets de conexão e processos. Na verdade, os servidores Web de alto desempenho atuais muitas vezes utilizam somente um processo, mas criam uma nova thread com um novo socket de conexão para cada nova conexão cliente. (Uma thread pode ser considerada um subprocesso leve.) Se você fez a primeira tarefa de programação do Capítulo 2, construiu um servidor Web que faz exatamente isso. Para um servidor desses, a qualquer dado instante podem haver muitos sockets de conexão (com identificadores diferentes) ligados ao mesmo processo.

Se o cliente e o servidor estiverem usando HTTP persistente, então, durante toda a conexão persistente, eles trocarão mensagens HTTP através do mesmo socket do servidor. Todavia, se o cliente e o servidor usarem HTTP

Segurança em foco

Varredura de Porta

Vimos que um processo servidor espera pacientemente em uma porta aberta para contato por um cliente remoto. Algumas portas são reservadas para aplicações familiares (por exemplo, Web, FTP, DNS e os servidores SMTP); outras portas são utilizadas por convenção por aplicações populares (por exemplo, o Microsoft SQL Server 2000 ouve as solicitações na porta 1434 do UDP). Desta forma, se determinarmos que uma porta está aberta no hospedeiro, talvez possamos mapear essa porta para uma aplicação específica sendo executada no hospedeiro. Isso é muito útil para administradores de sistema, que frequentemente têm interesse em saber quais aplicações de rede estão sendo executadas nos hospedeiros em suas redes. Porém, os atacantes, a fim de "examinarem o local", também querem saber quais portas estão abertas nos hospedeiros direcionados. Se o hospedeiro estiver sendo executado em uma aplicação com uma falha de segurança conhecida (por exemplo, um SQL Server ouvindo em uma porta 1434 estava sujeito a esgotamento de buffer, permitindo que um usuário remoto execute um código arbitrário no hospedeiro vulnerável, uma falha explorada pelo worm Slammer [CERT, 2003-04]), então esse hospedeiro está pronto para o ataque.

Determinar quais aplicações estão ouvindo em quais portas é relativamente uma tarefa fácil. De fato há um número de programas de domínio público, denominados varredores de porta, que fazem exatamente isso. Talvez o mais utilizado seja o nmap, disponível gratuitamente em <http://insecure.org/nmap> e está incluso na maioria das distribuições Linux. Para o TCP, o nmap varre portas sequencialmente, procurando por portas que aceitam conexões TCP. Para o UDP, o nmap novamente varre portas sequencialmente, procurando por portas UDP que respondem aos segmentos UDP transmitidos. Em ambos os casos, o nmap retorna uma lista de portas abertas, fechadas ou inalcançáveis. Um hospedeiro executando o nmap pode tentar varrer qualquer hospedeiro direcionado em qualquer lugar da Internet. Voltaremos a falar sobre o nmap na Seção 3.5.6, ao discutirmos gerenciamento da conexão TCP.

não persistente, então uma nova conexão TCP é criada e encerrada para cada requisição/resposta e, portanto, um novo socket é criado e mais tarde encerrado para cada requisição/resposta. Essa criação e encerramento frequentes de sockets podem causar sério impacto sobre o desempenho de um servidor Web movimentado (embora o sistema operacional possa usar vários estratagemas para atenuar o problema). Aconselhamos o leitor interessado em questões de sistema operacional referentes a HTTP persistente e não persistente a consultar [Nielsen, 1997; Nahum, 2002].

Agora que já discutimos multiplexação e demultiplexação na camada de transporte, passemos à discussão de um dos protocolos da Internet, o UDP. Na próxima seção, veremos que a contribuição desse protocolo não é muito mais do que um serviço de multiplexação/demultiplexação.

3.3 Transporte não orientado para conexão: UDP

Nesta seção, examinaremos o UDP mais de perto, como ele funciona e o que ele faz. Aconselhamos o leitor a rever o material apresentado na Seção 2.1, que inclui uma visão geral do modelo de serviço UDP, e o da Seção 2.8, que discute a programação de portas por UDP.

Para motivar nossa discussão sobre UDP, suponha que você esteja interessado em projetar um protocolo de transporte simples, bem básico. Como você faria isso? De início, você deve considerar a utilização de um protocolo de transporte vazio. Em especial, do lado do remetente, considere pegar as mensagens do processo de aplicação e passá-las diretamente para a camada de rede; do lado do destinatário, considere pegar as mensagens que chegam da camada de rede e passá-las diretamente ao processo da aplicação. Mas, como aprendemos na seção anterior, o que teremos de fazer é praticamente nada. No mínimo, a camada de transporte tem de fornecer um serviço de multiplexação/demultiplexação para passar os dados da camada de rede ao processo de aplicação correto.

O UDP, definido no [RFC 768], faz apenas quase tão pouco quanto um protocolo de transporte pode fazer. À parte sua função de multiplexação/demultiplexação e de alguma verificação de erros, ele nada adiciona ao IP. Na verdade, se o criador da aplicação escolher o UDP, em vez do TCP, a aplicação estará ‘falando’ quase diretamente com o IP. O UDP pega as mensagens do processo de aplicação, anexa os campos de número de porta da fonte e do destino para o serviço de multiplexação/demultiplexação, adiciona dois outros pequenos campos e passa o segmento resultante à camada de rede, que encapsula o segmento dentro de um datagrama IP e, em seguida, faz uma tentativa de melhor esforço para entregar o segmento ao hospedeiro receptor. Se o segmento chegar ao hospedeiro receptor, o UDP usará o número de porta de destino para entregar os dados do segmento ao processo de aplicação correto. Note que, com o UDP, não há apresentação entre as entidades remetente e destinatária da camada de transporte antes de enviar um segmento. Por essa razão, dizemos que o UDP é não orientado para conexão.

O DNS é um exemplo de protocolo de camada de aplicação que usa o UDP. Quando a aplicação DNS em um hospedeiro quer fazer uma consulta, constrói uma mensagem de consulta DNS e passa a mensagem para o UDP. Sem realizar nenhuma apresentação com a entidade UDP que está funcionando no sistema final de destino, o UDP do lado do hospedeiro adiciona campos de cabeçalho à mensagem e passa o segmento resultante à camada de rede, que encapsula o segmento UDP em um datagrama e o envia a um servidor de nomes. A aplicação DNS no hospedeiro requisitante então espera por uma resposta à sua consulta. Se não receber uma resposta (possivelmente porque a rede subjacente perdeu a consulta ou a resposta), ela tentará enviar a consulta a outro servidor de nomes ou informará à aplicação consultante que não pode obter uma resposta.

É possível que agora você esteja imaginando por que um criador de aplicação escolheria construir uma aplicação sobre UDP, em vez de sobre TCP. O TCP não é sempre preferível ao UDP, já que fornece serviço confiável de transferência de dados e o UDP não? A resposta é ‘não’, pois muitas aplicações se adaptam melhor ao UDP pelas seguintes razões:

Melhor controle no nível da aplicação sobre quais dados são enviados e quando. Com UDP, tão logo um processo de aplicação passe dados ao UDP, o protocolo empacotará esses dados dentro de um segmento UDP e os passará imediatamente à camada de rede. O TCP, por outro lado, tem um mecanismo de controle de congestionamento que limita o remetente TCP da camada de transporte quando um ou mais

enlaces entre os hospedeiros da fonte e do destinatário ficam excessivamente congestionados. O TCP também continuará a reenviar um segmento até que o hospedeiro destinatário reconheça a recepção desse segmento, independentemente do tempo que a entrega confiável levar. Visto que aplicações de tempo real requerem uma taxa mínima de envio, não querem atrasar excessivamente a transmissão de segmentos e podem tolerar uma certa perda de dados, o modelo de serviço do TCP não é particularmente compatível com essas necessidades das aplicações. Como discutiremos mais adiante, essas aplicações podem usar UDP e implementar, como parte da aplicação, qualquer funcionalidade adicional necessária além do serviço de entrega de segmentos simples e básico do UDP.

Não há estabelecimento de conexão. Como discutiremos mais adiante, o TCP usa uma apresentação de três vias antes de começar a transferir dados. O UDP simplesmente envia mensagens sem nenhuma preliminar formal e, assim, não introduz nenhum atraso para estabelecer uma conexão. Provavelmente esta é a principal razão pela qual o DNS roda sobre UDP, e não sobre TCP — o DNS seria muito mais lento se rodasse em TCP. O HTTP usa o TCP, e não o UDP, porque a confiabilidade é fundamental para páginas Web com texto. Mas, como discutimos brevemente na Seção 2.2, o atraso de estabelecimento de uma conexão TCP é uma contribuição importante aos atrasos associados à recepção de documentos Web.

Não há estados de conexão. O TCP mantém o estado de conexão nos sistemas finais. Esse estado inclui buffers de envio e recebimento, parâmetros de controle de congestionamento e parâmetros numéricos de sequência e de reconhecimento. Veremos na Seção 3.5 que essa informação de estado é necessária para implementar o serviço de transferência confiável de dados do TCP e para prover controle de congestionamento. O UDP, por sua vez, não mantém o estado de conexão e não monitora nenhum desses parâmetros. Por essa razão, um servidor devotado a uma aplicação específica pode suportar um número muito maior de clientes ativos quando a aplicação roda sobre UDP e não sobre TCP.

Pequena sobrecarga de cabeçalho de pacote. O segmento TCP tem 20 bytes de sobrecarga de cabeçalho além dos dados para cada segmento, enquanto o UDP tem somente 8 bytes de sobrecarga.

A Figura 3.6 relaciona aplicações populares da Internet e os protocolos de transporte que elas usam. Como era de esperar, o e-mail, o acesso a terminal remoto, a Web e a transferência de arquivos rodam sobre TCP — todas essas aplicações necessitam do serviço confiável de transferência de dados do TCP. Não obstante, muitas aplicações importantes executam sobre UDP, e não sobre TCP. O UDP é usado para atualização das tabelas de roteamento com o protocolo RIP (*routing information protocol* — protocolo de informação de roteamento) (veja Seção 4.6.1). Como as atualizações RIP são enviadas periodicamente (em geral, a cada 5 minutos), atualizações perdidas serão substituídas por atualizações mais recentes, tornando inútil a recuperação das atualizações perdidas. O UDP também é usado para levar dados de gerenciamento de rede (SNMP; veja o Capítulo 9). Nesse caso, o UDP é preferível ao TCP, já que aplicações de gerenciamento de rede frequentemente devem funcionar quando a rede está em estado sobrecarregado — exatamente quando é difícil conseguir transferência confiável de dados com congestionamento controlado. E também, como mencionamos anteriormente, o DNS roda sobre UDP, evitando, desse modo, atrasos de estabelecimento de conexões TCP.

Como mostra a Figura 3.6, hoje o UDP e o TCP também é comumente usado para aplicações de multimídia, como telefone por Internet, videoconferência em tempo real e recepção de áudio e vídeo armazenados. Examinaremos essas aplicações mais de perto no Capítulo 7. No momento, mencionamos apenas que todas essas aplicações podem tolerar uma pequena quantidade de perda de pacotes, de modo que a transferência confiável de dados não é absolutamente crítica para o sucesso da aplicação. Além disso, aplicações em tempo real, como telefone por Internet e videoconferência, reagem muito mal ao controle de congestionamento do TCP. Por essas razões, os desenvolvedores de aplicações de multimídia muitas vezes optam por rodar suas aplicações sobre UDP em vez de sobre TCP. Entretanto, o TCP está sendo utilizado cada vez mais para transporte de mídia. Por exemplo, [Sripanidkulchai, 2004] descobriu que aproximadamente 75% do fluxo em tempo real e gravado utilizaram TCP. Quando as taxas de perda de pacote são baixas, junto com algumas empresas que bloqueiam o tráfego UDP por razões de segurança (veja Capítulo 8), o TCP se torna um protocolo cada vez mais atrativo para o transporte de mídia.

Aplicação	Protocolo de camada de aplicação	Protocolo de transporte subjacente
Correio eletrônico	SMTP	TCP
Acesso a terminal remoto	Telnet	TCP
Web	HTTP	TCP
Transferência de arquivo	FTP	TCP
Servidor remoto de arquivo	NFS	tipicamente UDP
Recepção de multimídia	tipicamente proprietária	UDP ou TCP
Telefonia por Internet	tipicamente proprietária	UDP ou TCP
Gerenciamento de rede	SNMP	tipicamente UDP
Protocolo de roteamento	RIP	tipicamente UDP
Tradução de nome	DNS	tipicamente UDP

Figura 3.6 Aplicações populares da Internet e seus protocolos de transporte subjacentes

Embora atualmente seja comum rodar aplicações de multimídia sobre UDP, isso é controvertido. Como já mencionamos, o UDP não tem controle de congestionamento. Mas esse controle é necessário para evitar que a rede entre em um estado no qual pouquíssimo trabalho útil é realizado. Se todos começassem a enviar vídeo com alta taxa de bits sem usar nenhum controle de congestionamento, haveria tamanho transbordamento de pacotes nos roteadores que poucos pacotes UDP conseguiram atravessar com sucesso o caminho da fonte ao destino. Além do mais, as altas taxas de perda induzidas pelos remetentes UDP sem controle fariam com que os remetentes TCP (que, como veremos mais adiante, *reduzem* suas taxas de envio em face de congestionamento) reduzissem drasticamente suas taxas. Assim, a falta de controle de congestionamento no UDP pode resultar em altas taxas de perda entre um remetente e um destinatário UDP e no acúmulo de sessões TCP — um problema potencialmente sério [Floyd, 1999]. Muitos pesquisadores propuseram novos mecanismos para forçar todas as fontes, inclusive as fontes UDP, a realizar um controle de congestionamento adaptativo [Mahdavi, 1997; Floyd, 2000; Kohler, 2006: RFC 4340].

Antes de discutirmos a estrutura do segmento UDP, mencionaremos que é possível que uma aplicação tenha transferência confiável de dados usando UDP. Isso pode ser feito se a confiabilidade for embutida na própria aplicação (por exemplo, adicionando mecanismos de reconhecimento e de retransmissão, tais como os que estudaremos na próxima seção). Mas esta é uma tarefa não trivial que manteria o desenvolvedor ocupado com a depuração por um longo tempo. Não obstante, embutir confiabilidade diretamente na aplicação permite que ela tire proveito de ambas as alternativas. Em outras palavras, os processos de aplicação podem se comunicar de maneira confiável sem ter de se sujeitar às limitações da taxa de transmissão impostas pelo mecanismo de controle de congestionamento do TCP.

3.3.1 Estrutura do segmento UDP

A estrutura do segmento UDP, mostrada na Figura 3.7, é definida no RFC 768. Os dados da aplicação ocupam o campo de dados do segmento UDP. Por exemplo, para o DNS, o campo de dados contém uma mensagem de consulta ou uma mensagem de resposta. Para uma aplicação de recepção de áudio, parcelas de áudio preenchem o campo de dados. O cabeçalho UDP tem apenas quatro campos, cada um consistindo em 2 bytes. Como já discutido na seção anterior, os números de porta permitem que o hospedeiro destinatário passe os dados da aplicação ao processo correto que está funcionando na máquina destinatária (isto é, realize a função de demultiplexação). A soma de verificação é usada pelo hospedeiro receptor para verificar se foram introduzidos erros no segmento. Na verdade, a soma de verificação também é calculada para alguns dos campos no cabeçalho IP, além do segmento UDP. Mas ignoramos esse detalhe para podermos enxergar a floresta por entre as árvores. Discutiremos o cálculo

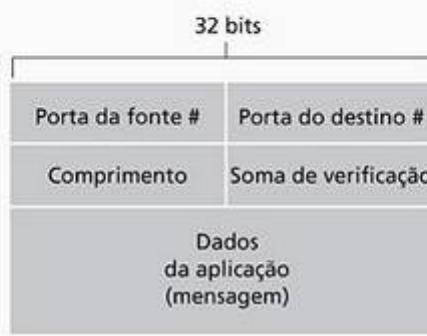


Figura 3.7 Estrutura do segmento UDP

da soma de verificação mais adiante. Os princípios básicos da detecção de erros estão descritos na Seção 5.2. O campo de comprimento especifica o comprimento do segmento UDP, incluindo o cabeçalho, em bytes.

3.3.2 Soma de verificação UDP

A soma de verificação UDP serve para detectar erros. Em outras palavras, é usada para determinar se bits dentro do segmento UDP foram alterados (por exemplo, por ruído nos enlaces ou enquanto armazenados em um roteador) durante sua movimentação da fonte até o destino. O UDP no lado remetente realiza o complemento de 1 da soma de todas as palavras de 16 bits do segmento levando em conta o “vai um” em toda a soma. Esse resultado é colocado no campo de soma de verificação no segmento UDP. Damos aqui um exemplo simples do cálculo da soma de verificação. Se quiser saber detalhes sobre a implementação eficiente do algoritmo de cálculo e sobre o desempenho com dados reais, consulte o RFC 1071 e [Stone, 1998 e 2000], respectivamente. Como exemplo, suponha que tenhamos as seguintes três palavras de 16 bits:

```
0110011001100000
0101010101010101
1000111100001100
```

A soma das duas primeiras dessas palavras de 16 bits é:

```
0110011001100000
0101010101010101
1011101110110101
```

Adicionando a terceira palavra à soma acima, temos:

```
1011101110110101
1000111100001100
0100101011000010
```

Note que essa última adição teve “vai um” no bit mais significativo que foi somado ao bit menos significativo. O complemento de 1 é obtido pela conversão de todos os 0 em 1 e de todos os 1 em 0. Desse modo, o complemento de 1 da soma 0100101011000010 é 1011010100111101, que passa a ser a soma de verificação. No destinatário, todas as quatro palavras de 16 bits são somadas, inclusive a soma de verificação. Se nenhum erro for introduzido no pacote, a soma no destinatário será, obviamente, 1111111111111111. Se um dos bits for um zero, saberemos então que houve introdução de erro no pacote.

Provavelmente você está imaginando por que o UDP fornece uma soma de verificação em primeiro lugar, visto que muitos protocolos de camada de enlace (entre os quais, o popular protocolo Ethernet) também for-

necem verificação de erros. A razão é que não há garantia de que todos os enlaces entre a origem e o destino forneçam verificação de erros — um dos enlaces pode usar um protocolo de camada de enlace que não forneça verificação de erros. Além disso, mesmo que os segmentos sejam corretamente transmitidos por um enlace, é possível haver introdução de erros de bits quando um segmento é armazenado na memória de um roteador. Dado que não são garantidas nem a confiabilidade enlace a enlace, nem a detecção de erro na memória, o UDP deve prover detecção de erro *sí e só* na camada de transporte se quisermos que o serviço de transferência de dados sí e só forneça detecção de erro. Esse é um exemplo do famoso **princípio sí e só** do projeto de sistemas [Saltzer, 1984]. Esse princípio afirma que, uma vez que é dado como certo que funcionalidades (detecção de erro, neste caso) devem ser implementadas sí e só, “funções colocadas nos níveis mais baixos podem ser redundantes ou de pouco valor em comparação com o custo de fornecer-las no nível mais alto”.

Como se pretende que o IP rode sobre qualquer protocolo de camada 2, é útil que a camada de transporte forneça verificação de erros como medida de segurança. Embora o UDP forneça verificação de erros, ele nada faz para recuperar um erro. Algumas implementações do UDP simplesmente descartam o segmento danificado; outras passam o segmento errado à aplicação acompanhado de um aviso.

Isso encerra nossa discussão sobre o UDP. Logo veremos que o TCP oferece transferência confiável de dados às suas aplicações, bem como outros serviços que o UDP não oferece. Naturalmente, o TCP também é mais complexo do que o UDP. Contudo, antes de discutirmos o TCP, primeiramente devemos examinar os princípios subjacentes da transferência confiável de dados.

3.4 Princípios da transferência confiável de dados

Nesta seção, consideraremos o problema conceitual da transferência confiável de dados. Isso é apropriado, já que o problema de implementar transferência confiável de dados ocorre não somente na camada de transporte, mas também na camada de enlace e na camada de aplicação. Assim, o problema geral é de importância central para o trabalho em rede. Na verdade, se tivéssemos de fazer uma lista dos dez maiores problemas fundamentalmente importantes para o trabalho em rede, o da transferência confiável de dados seria o candidato número um da lista. Na seção seguinte, examinaremos o TCP e mostraremos, em especial, que ele utiliza muitos dos princípios que descreveremos aqui.

A Figura 3.8 ilustra a estrutura para nosso estudo de transferência confiável de dados. A abstração do serviço fornecido às entidades das camadas superiores é a de um canal confiável através do qual dados podem ser transferidos. Com um canal confiável, nenhum dos dados transferidos é corrompido (trocado de 0 para 1 ou vice-versa) nem perdido, e todos são entregues na ordem em que foram enviados. Este é exatamente o modelo de serviço oferecido pelo TCP às aplicações de Internet que recorrem a ele.

É responsabilidade de um **protocolo de transferência confiável de dados** implementar essa abstração de serviço. A tarefa é dificultada pelo fato de que a camada abaixo do protocolo de transferência confiável de dados pode ser não confiável. Por exemplo, o TCP é um protocolo confiável de transferência de dados que é implementado sobre uma camada de rede sí e só não confiável (IP). De modo mais geral, a camada abaixo das duas extremidades que se comunicam confiavelmente pode consistir em um único enlace físico (como é o caso, por exemplo, de um protocolo de transferência de dados na camada de enlace) ou em uma rede global interligada (como é o caso de um protocolo de camada de transporte). Para nossa finalidade, contudo, podemos considerar essa camada mais baixa simplesmente como um canal ponto a ponto não confiável.

Nesta seção, desenvolveremos gradualmente os lados remetente e destinatário de um protocolo confiável de transferência de dados, considerando modelos progressivamente mais complexos do canal subjacente. A Figura 3.8(b) ilustra as interfaces de nosso protocolo de transferência de dados. O lado remetente do protocolo de transferência de dados será invocado de cima, por uma chamada a `rdt_send()`. Ele passará os dados a serem entregues à camada superior no lado destinatário. (Aqui, `rdt` representa o protocolo de transferência confiável de dados [*reliable data transfer*] e `_send` indica que o lado remetente do `rdt` está sendo chamado. O primeiro

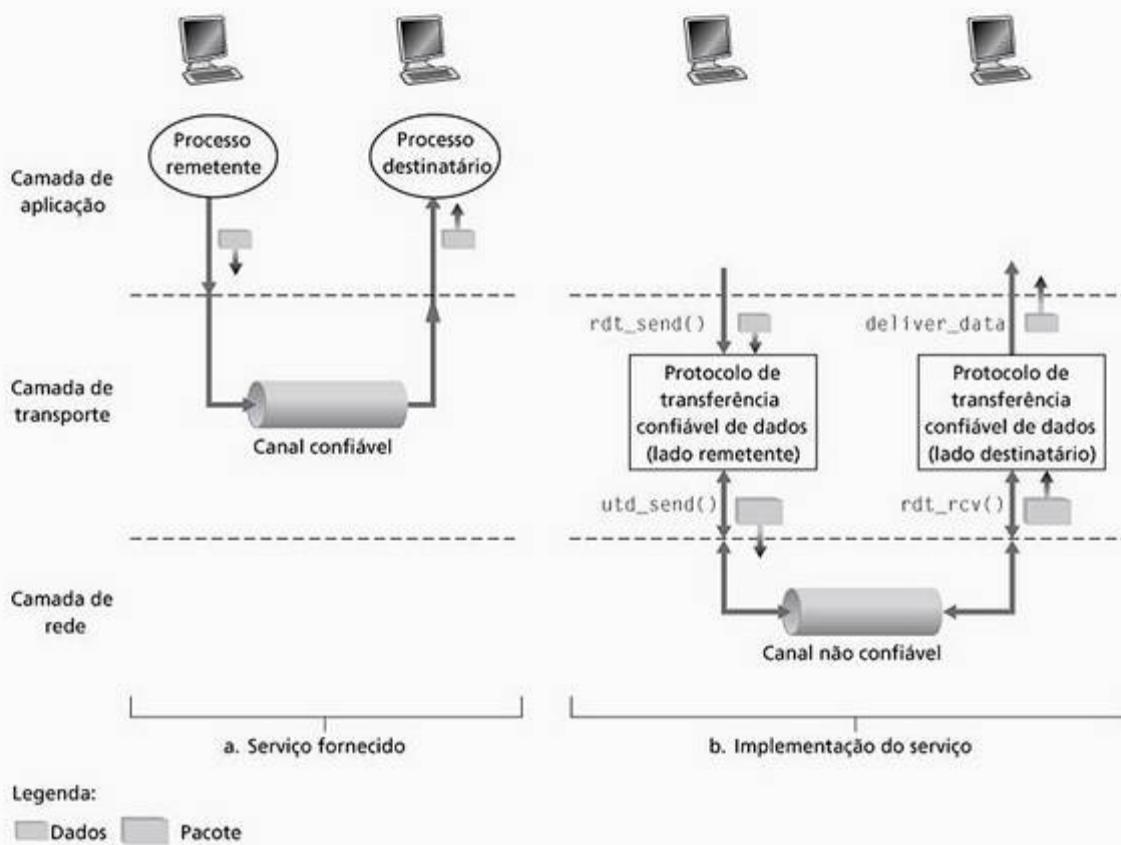


Figura 3.8 Transferência confiável de dados: modelo do serviço e implementação do serviço

passo no desenvolvimento de qualquer protocolo é dar-lhe um bom nome!) Do lado destinatário, `rdt_rcv()` será chamado quando um pacote chegar do lado destinatário do canal. Quando o protocolo `rdt` quiser entregar dados à camada superior, ele o fará chamando `deliver_data()`. No que se segue, usamos a terminologia 'pacote' em vez de 'segmento' de camada de transporte. Como a teoria desenvolvida nesta seção se aplica a redes de computadores em geral, e não somente à camada de transporte da Internet, o termo genérico 'pacote' talvez seja mais apropriado aqui.

Nesta seção, consideraremos apenas o caso de **transferência unidirecional de dados**, isto é, transferência de dados do lado remetente ao lado destinatário. O caso de **transferência bidirecional confiável de dados** (isto é, *full-duplex*) não é conceitualmente mais difícil, mas é bem mais tedioso de explicar. Embora consideremos apenas a transferência unidirecional de dados, é importante notar que, apesar disso, os lados remetente e destinatário de nosso protocolo terão de transmitir pacotes em ambas as direções, como mostra a Figura 3.8. Logo veremos que, além de trocar pacotes contendo os dados a transferir, os lados remetente e destinatário do `rdt` também precisarão trocar pacotes de controle entre si. Ambos os lados de envio e destino do `rdt` enviam pacotes para o outro lado por meio de uma chamada a `utd_send()` (em que `utd` representa transferência não confiável de dados — *unreliable data transfer*).

3.4.1 Construindo um protocolo de transferência confiável de dados

Vamos percorrer agora uma série de protocolos que vão se tornando cada vez mais complexos, até chegar a um protocolo de transferência confiável de dados impecável.

Transferência confiável de dados sobre um canal perfeitamente confiável: rdt1.0

Vamos considerar primeiramente o caso mais simples, em que o canal subjacente é completamente confiável. O protocolo em si, que denominaremos rdt1.0, é trivial. As definições de **máquina de estado finito** (*finite-state machine* — FSM) para o remetente e o destinatário rdt1.0 são apresentadas na Figura 3.9. A FSM da Figura 3.9(a) define a operação do remetente, enquanto a FSM da Figura 3.9(b) define a operação do destinatário. É importante notar que há FSM *separadas* para o remetente e o destinatário. Ambas as FSM da Figura 3.9 têm apenas um estado. As setas na descrição da FSM indicam a transição do protocolo de um estado para outro. (Uma vez que cada FSM da Figura 3.9 tem apenas um estado, uma transição é, necessariamente, de um dado estado para ele mesmo; examinaremos diagramas de estados mais complicados em breve.) O evento que causou a transição é mostrado acima da linha horizontal que a rotula, e as ações realizadas quando ocorre o evento são mostradas abaixo da linha horizontal. Quando nenhuma ação é realizada em um evento, ou quando não ocorre nenhum evento e uma ação é realizada, usaremos o símbolo Λ , acima ou abaixo da linha horizontal, para indicar explicitamente a falta de uma ação ou de um evento, respectivamente. O estado inicial da FSM é indicado pela seta tracejada. Embora as FSMs da Figura 3.9 tenham somente um estado, as outras que veremos em breve têm vários estados, portanto, será importante identificar o estado inicial de cada FSM.

O lado remetente do rdt simplesmente aceita dados da camada superior pelo evento `rdt_send(data)`, cria um pacote que contém os dados (pela ação `make_pkt(data)`) e envia-o para dentro do canal. Na prática, o evento `rdt_send(data)` resultaria de uma chamada de procedimento (por exemplo, para `rdt_send()`) pela aplicação da camada superior.

No lado destinatário, rdt recebe um pacote do canal subjacente pelo evento `rdt_rcv(packet)`, extraí os dados do pacote (pela ação `extract(packet, data)`) e os passa para a camada superior (pela ação `deliver_data(data)`). Na prática, o evento `rdt_rcv(packet)` resultaria de uma chamada de procedimento (por exemplo, para `rdt_rcv()`) do protocolo da camada inferior.

Nesse protocolo simples, não há diferença entre a unidade de dados e um pacote. E, também, todo o fluxo de pacotes corre do remetente para o destinatário; com um canal perfeitamente confiável, não há necessidade de o lado destinatário fornecer qualquer informação ao remetente, já que nada pode dar errado! Note que também admitimos que o destinatário está capacitado a receber dados seja qual for a velocidade em que o remetente os envie. Assim, não há necessidade de pedir para o remetente desacelerar!



a. rdt1.0: lado remetente



b. rdt1.0: lado destinatário

Figura 3.9 rdt1.0 — Um protocolo para um canal completamente confiável

Transferência confiável de dados por um canal com erros de bits: rdt2.0

Um modelo mais realista de canal subjacente é um canal em que os bits de um pacote podem ser corrompidos. Esses erros de bits ocorrem normalmente nos componentes físicos de uma rede enquanto o pacote é transmitido, propagado ou armazenado. Continuaremos a admitir, por enquanto, que todos os pacotes transmitidos sejam recebidos (embora seus bits possam estar corrompidos) na ordem em que foram enviados.

Antes de desenvolver um protocolo para se comunicar de maneira confiável com esse canal, considere primeiramente como as pessoas enfrentariam uma situação como essa. Considere como você ditaria uma mensagem longa pelo telefone. Em um cenário típico, quem estivesse anotando a mensagem diria 'o.k.' após cada sentença que ouvisse, entendesse e anotasse. Se a pessoa ouvisse uma mensagem truncada, pediria que você a repetisse. Esse protocolo de ditado de mensagem usa **reconhecimentos positivos** ('o.k.') e **reconhecimentos negativos** ('Repita, por favor'). Essas mensagens de controle permitem que o destinatário faça o remetente saber o que foi recebido corretamente e o que foi recebido com erro e, portanto, exige repetição. Em um arranjo de rede de computadores, protocolos de transferência confiável de dados baseados nesse tipo de retransmissão são conhecidos como **protocolos ARQ (Automatic Repeat reQuest — solicitação automática de repetição)**.

Essencialmente, são exigidas três capacitações adicionais dos protocolos ARQ para manipular a presença de erros de bits:

Detectão de erros. Primeiramente, é preciso um mecanismo que permita ao destinatário detectar quando ocorrem erros. Lembre-se de que dissemos na seção anterior que o UDP usa o campo de soma de verificação da Internet exatamente para essa finalidade. No Capítulo 5, examinaremos, com mais detalhes, técnicas de detecção e de correção de erros. Essas técnicas permitem que o destinatário detecte e possivelmente corrija erros de bits de pacotes. Por enquanto, basta saber que essas técnicas exigem que bits extras (além dos bits dos dados originais a serem transferidos) sejam enviados do remetente ao destinatário. Esses bits são colocados no campo de soma de verificação do pacote de dados do protocolo rdt2.0.

Realimentação do destinatário. Uma vez que remetente e destinatário normalmente estejam rodando em sistemas finais diferentes, possivelmente separados por milhares de quilômetros, o único modo de o remetente saber qual é a visão de mundo do destinatário (neste caso, se um pacote foi recebido corretamente ou não) é o destinatário fornecer realimentação explícita ao remetente. As respostas de reconhecimento positivo (ACK) ou negativo (NAK) no cenário do ditado da mensagem são exemplos dessa realimentação. Nossa protocolo rdt2.0 devolverá, dessa mesma maneira, pacotes ACK e NAK do destinatário ao remetente. Em princípio, esses pacotes precisam apenas ter o comprimento de um bit; por exemplo, um valor 0 poderia indicar um NAK e um valor 1 poderia indicar um ACK.

Retransmissão. Um pacote que é recebido com erro no destinatário será retransmitido pelo remetente.

A Figura 3.10 mostra a representação por FSM do rdt2.0, um protocolo de transferência de dados que emprega detecção de erros, reconhecimentos positivos e reconhecimentos negativos.

O lado remetente do rdt2.0 tem dois estados. No estado mais à esquerda, o protocolo do lado remetente está esperando que os dados sejam passados pela camada superior. Quando o evento `rdt_send(data)` ocorrer, o remetente criará um pacote (`sndpkt`) contendo os dados a serem enviados, juntamente com uma soma de verificação do pacote (por exemplo, como discutimos na Seção 3.3.2 para o caso de um segmento UDP) e, então, enviará o pacote pela operação `udt_send(sndpkt)`. No estado mais à direita, o protocolo remetente está esperando por um pacote ACK ou NAK da parte do destinatário. Se um pacote ACK for recebido (a notação `rdt_rcv(rcvpkt) && isACK(rcvpkt)` na Figura 3.10 corresponde a esse evento), o remetente saberá que o pacote transmitido mais recentemente foi recebido corretamente. Assim, o protocolo volta ao estado de espera por dados vindos da camada superior. Se for recebido um NAK, o protocolo retransmitirá o último pacote e esperará por um ACK ou NAK a ser devolvido pelo destinatário em resposta ao pacote de dados retransmitido. É importante notar que, quando o destinatário está no estado de espera por ACK ou NAK, não pode receber mais dados da camada superior; isto é, o evento `rdt_send()` não pode ocorrer; isso somente acontecerá após o remetente receber um ACK e sair desse estado. Assim, o remetente não enviará novos dados até ter certeza de que o

destinatário recebeu corretamente o pacote em questão. Devido a esse comportamento, protocolos como o rdt2.0 são conhecidos como protocolos pare e espere (*stop-and-wait*).

A FSM do lado destinatário para o rdt2.0 tem um único estado. Quando o pacote chega, o destinatário responde com um ACK ou um NAK, dependendo de o pacote recebido estar ou não corrompido. Na Figura 3.10, a notação `rdt_rcv(rcvpkt) && corrupt(rcvpkt)` corresponde ao evento em que um pacote é recebido e existe um erro.

Pode parecer que o protocolo rdt2.0 funciona, mas infelizmente ele tem um defeito fatal. Em especial, ainda não tratamos da possibilidade de o pacote ACK ou NAK estar corrompido! (Antes de continuar, é bom você começar a pensar em como esse problema pode ser resolvido.) Lamentavelmente, nossa pequena omissão não é tão inofensiva quanto possa parecer. No mínimo, precisaremos adicionar aos pacotes ACK/NAK bits de soma de verificação para detectar esses erros. A questão mais difícil é como o protocolo deve se recuperar de erros em pacotes ACK ou NAK. Nesse caso, a dificuldade é que, se um ACK ou um NAK estiver corrompido, o remetente não terá como saber se o destinatário recebeu ou não corretamente a última parcela de dados transmitidos.

Considere três possibilidades para manipular ACKs ou NAKs corrompidos:

Para a primeira possibilidade, imagine o que um ser humano faria no cenário do ditado da mensagem. Se quem estiver ditando não entender o 'o.k.' ou o 'Repita, por favor' do destinatário, provavelmente perguntará: "O que foi que você disse?" (introduzindo assim um novo tipo de pacote remetente-destinatário em nosso protocolo). O locutor então repetiria a resposta. Mas e se a frase "O que foi que

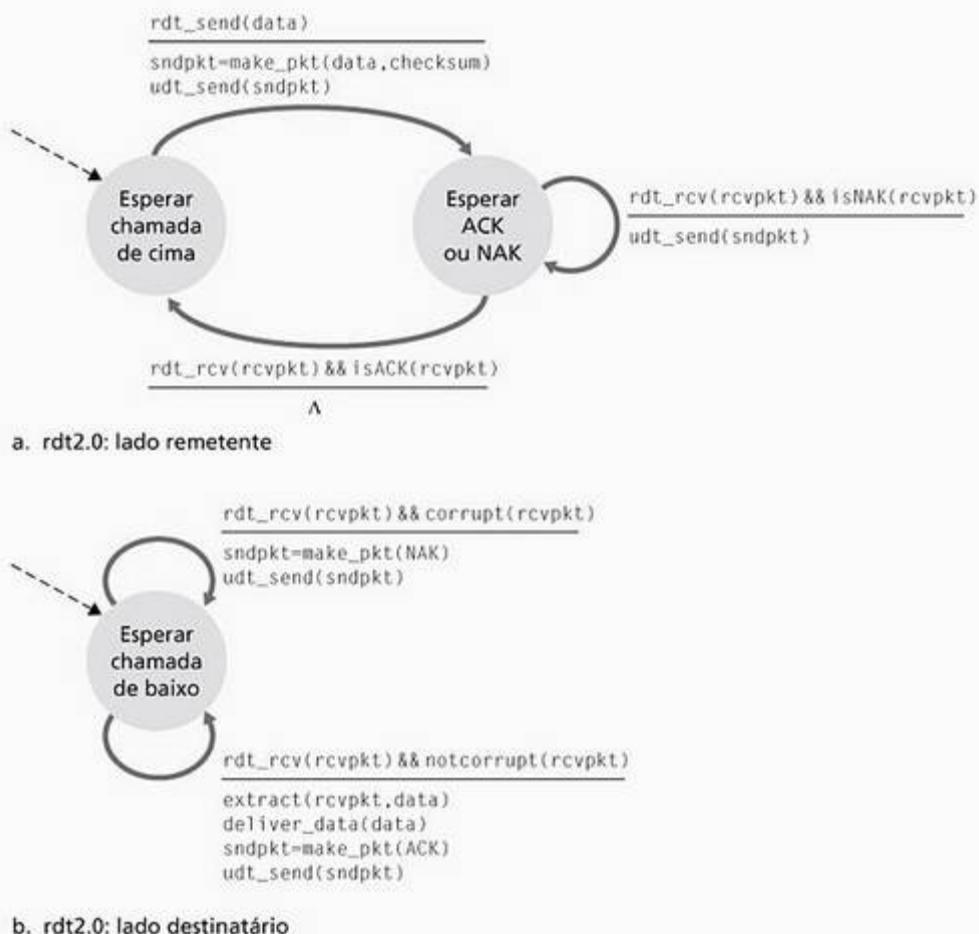


Figura 3.10 rdt2.0 — Um protocolo para um canal com erros de bits

você disse?" estivesse corrompida? O destinatário, sem ter nenhuma noção se a sentença corrompida era parte do ditado ou um pedido para repetir a última resposta, provavelmente responderia: "O que foi que você disse?" E então, é claro, essa resposta também poderia estar truncada. É óbvio que estamos entrando em um caminho difícil.

Uma segunda alternativa é adicionar um número suficiente de bits de soma de verificação para permitir que o remetente não somente detecte, mas também se recupere de erros de bits. Isso resolve o problema imediato para um canal que pode corromper pacotes, mas não perdê-los.

Uma terceira abordagem é o remetente simplesmente reenviar o pacote de dados corrente quando receber um pacote ACK ou NAK truncado. Esse método, no entanto, introduz pacotes duplicados no canal remetente-destinatário. A dificuldade fundamental com pacotes duplicados é que o destinatário não sabe se o último ACK ou NAK que enviou foi recebido corretamente no remetente. Assim, ele não pode saber *a priori* se um pacote que chega contém novos dados ou se é uma retransmissão!

Uma solução simples para esse novo problema (e que é adotada em quase todos os protocolos de transferência de dados existentes, inclusive o TCP) é adicionar um novo campo ao pacote de dados e fazer com que o remetente numere seus pacotes de dados colocando um número de sequência nesse campo. O destinatário então teria apenas de verificar esse número de sequência para determinar se o pacote recebido é ou não uma retransmissão. Para esse caso simples de protocolo pare e espere, um número de sequência de um bit é suficiente, já que permitirá que o destinatário saiba se o remetente está reenviando o pacote previamente transmitido (o número de sequência do pacote recebido é o mesmo do pacote recebido mais recentemente) ou um novo pacote (o número de sequência muda, indo "para a frente" em progressão aritmética de módulo 2). Como estamos admitindo que este é um canal que não perde pacotes, os pacotes ACK e NAK em si não precisam indicar o número de sequência do pacote que estão reconhecendo. O remetente sabe que um pacote ACK ou NAK recebido (truncado ou não) foi gerado em resposta ao seu pacote de dados transmitidos mais recentemente.

As figuras 3.11 e 3.12 mostram a descrição da FSM para o rdt2.1, nossa versão corrigida do rdt2.0. Cada um dos rdt2.1 remetente e destinatário da FSM agora tem um número duas vezes maior de estados do que antes. Isso acontece porque o estado do protocolo deve agora refletir se o pacote que está sendo correntemente

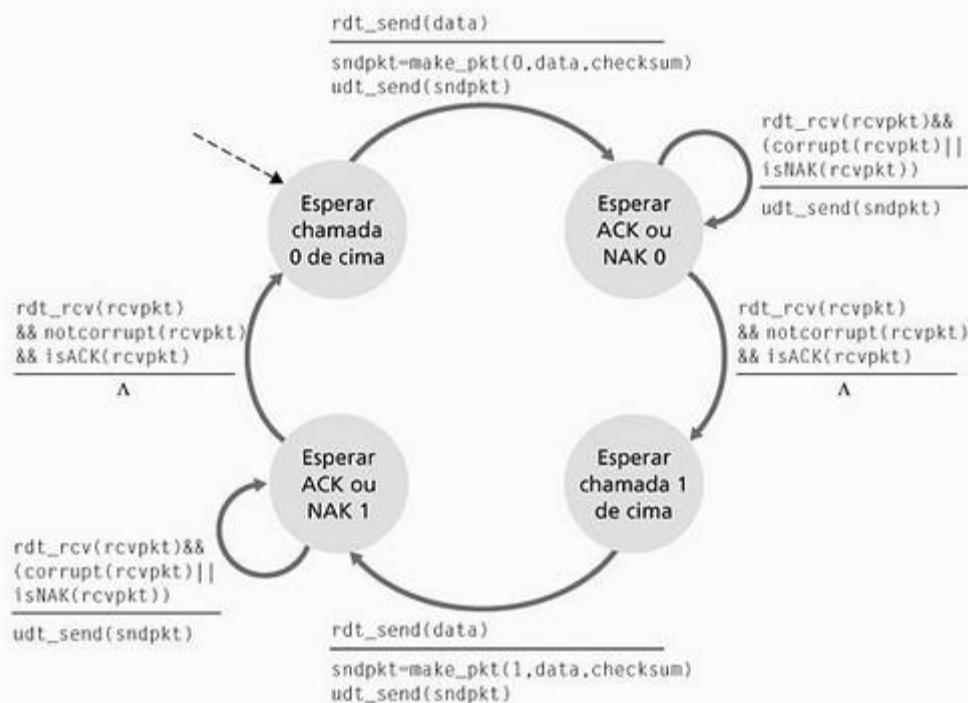


Figura 3.11 rdt2.1 remetente

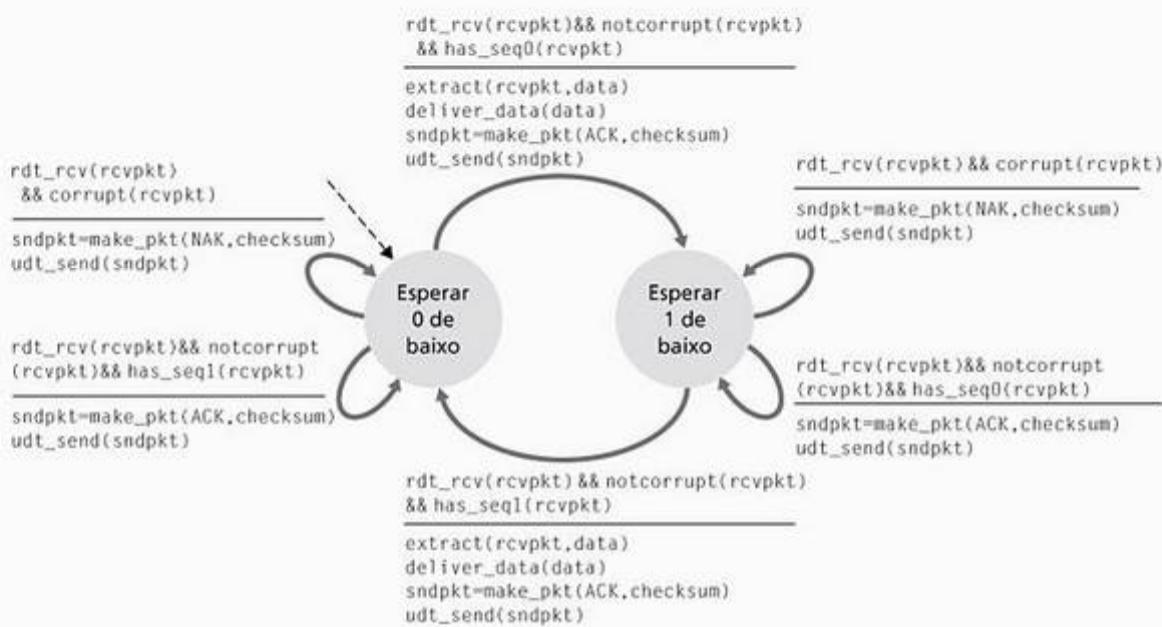


Figura 3.12 rdt2.1 destinatário

enviado (pelo remetente) ou aguardado (no destinatário) deveria ter um número de sequência 0 ou 1. Note que as ações nos estados em que um pacote numerado com 0 está sendo enviado ou aguardado são imagens especulares daquelas que devem funcionar quando estiver sendo enviado ou aguardado um pacote numerado com 1; as únicas diferenças têm a ver com a manipulação do número de sequência.

O protocolo rdt2.1 usa tanto o reconhecimento positivo como o negativo do remetente ao destinatário. Quando um pacote fora de ordem é recebido, o destinatário envia um reconhecimento positivo para o pacote que recebeu; quando um pacote corrompido é recebido, ele envia um reconhecimento negativo. Podemos conseguir o mesmo efeito de um pacote NAK se, em vez de enviarmos um NAK, enviarmos um ACK em seu lugar para o último pacote corretamente recebido. Um remetente que recebe dois ACKs para o mesmo pacote (isto é, **ACKs duplicados**) sabe que o destinatário não recebeu corretamente o pacote seguinte àquele para o qual estão sendo dados dois ACKs.

Nosso protocolo de transferência confiável de dados sem NAK para um canal com erros de bits é o rdt2.2, mostrado nas figuras 3.13 e 3.14. Uma modificação sutil entre rdt2.1 e rdt2.2 é que o destinatário agora deve incluir o número de sequência do pacote que está sendo reconhecido por uma mensagem ACK (o que é feito incluindo o argumento ACK, 0 ou ACK, 1 em make_pkt() na FSM destinatária) e o remetente agora deve verificar o número de sequência do pacote que está sendo reconhecido por uma mensagem ACK recebida (o que é feito incluindo o argumento 0 ou 1 em isACK() na FSM remetente).

Transferência confiável de dados por um canal com perda e com erros de bits: rdt3.0

Suponha agora que, além de corromper bits, o canal subjacente possa perder pacotes, um acontecimento que não é incomum nas redes de computadores de hoje (incluindo a Internet). Duas preocupações adicionais devem agora ser tratadas pelo protocolo: como detectar perda de pacote e o que fazer quando isso ocorre. A utilização de soma de verificação, números de sequência, pacotes ACK e retransmissões — as técnicas já desenvolvidas em rdt2.2 — nos permitirão atender a última preocupação. Lidar com a primeira preocupação, por sua vez, exigirá a adição de um novo mecanismo de protocolo.

Há muitas abordagens possíveis para lidar com a perda de pacote (e diversas delas serão estudadas nos exercícios ao final do capítulo). Aqui, atribuiremos ao remetente o encargo de detectar e se recuperar das perdas de pacote. Suponha que o remetente transmita um pacote de dados e que esse pacote, ou o ACK do seu destinatário, seja perdido. Em qualquer um dos casos, nenhuma resposta chegará ao remetente vinda do destinatário. Se o remetente

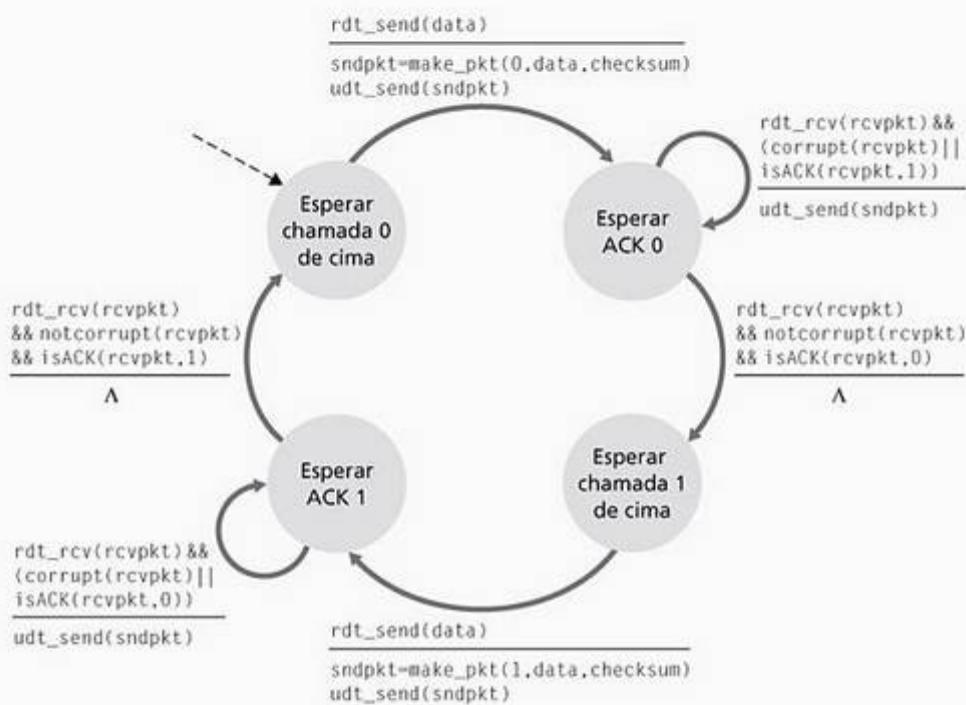


Figura 3.13 rdt2.2 remetente

estiver disposto a esperar o tempo suficiente para ter certeza de que o pacote foi perdido, ele poderá simplesmente retransmitir o pacote de dados. É preciso que você se convença de que esse protocolo funciona mesmo.

Mas quanto tempo o remetente precisa esperar para ter certeza de que algo foi perdido? É claro que deve aguardar no mínimo o tempo de um atraso de ida e volta entre ele e o destinatário (o que pode incluir buffers em roteadores ou equipamentos intermediários) e mais o tempo que for necessário para processar um pacote no destinatário. Em muitas redes, o atraso máximo para esses piores casos é muito difícil até de estimar, quanto mais

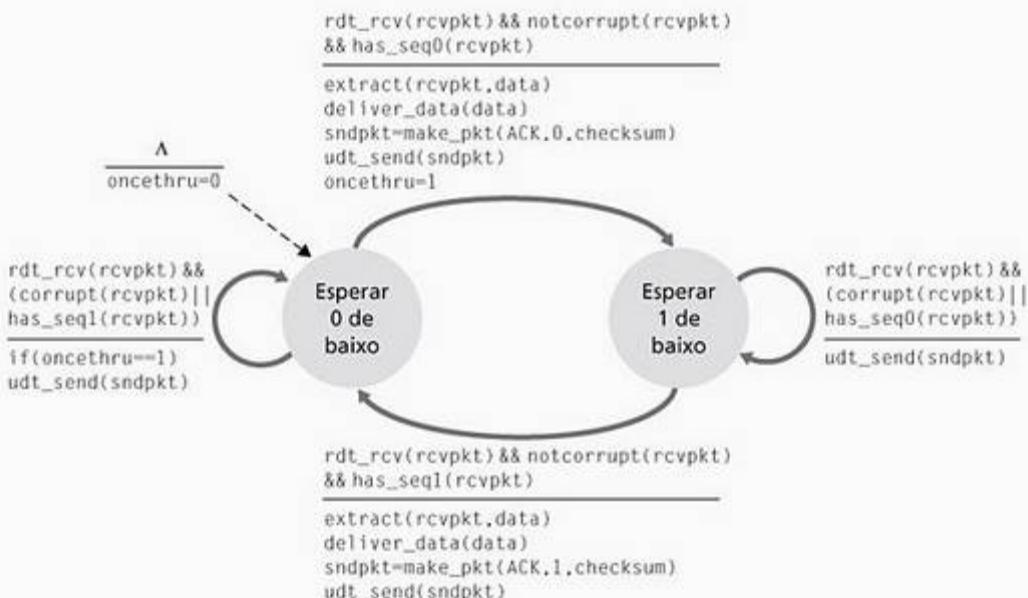


Figura 3.14 rdt2.2 destinatário

saber com certeza. Além disso, o ideal seria que o protocolo se recuperasse da perda de pacotes logo que possível; esperar pelo atraso do pior dos casos pode significar um longo tempo até que a recuperação do erro seja iniciada. Assim, a abordagem adotada na prática é a seguinte: o remetente faz uma escolha ponderada de um valor de tempo dentro do qual seria provável, mas não garantido, que a perda tivesse acontecido. Se não for recebido um ACK nesse período, o pacote é retransmitido. Note que, se um pacote sofrer um atraso particularmente longo, o remetente poderá retransmiti-lo mesmo que, nem o pacote de dados, nem o seu ACK tenham sido perdidos. Isso introduz a possibilidade de **pacotes de dados duplicados** no canal remetente-destinatário. Felizmente, o protocolo rdt2.2 já dispõe de funcionalidade suficiente (isto é, números de sequência) para tratar dos casos de pacotes duplicados.

Do ponto de vista do remetente, a retransmissão é uma panaceia. O remetente não sabe se um pacote de dados foi perdido, se um ACK foi perdido ou se o pacote ou o ACK simplesmente estavam muito atrasados. Em todos os casos, a ação é a mesma: retransmitir. Para implementar um mecanismo de retransmissão com base no tempo, é necessário um **temporizador de contagem regressiva** que interrompa o processo remetente após ter decorrido um dado tempo. Assim, será preciso que o remetente possa (1) acionar o temporizador todas as vezes que um pacote for enviado (quer seja a primeira vez, quer seja uma retransmissão), (2) responder a uma interrupção feita pelo temporizador (realizando as ações necessárias) e (3) parar o temporizador.

A Figura 3.15 mostra a FSM remetente para o rdt3.0, um protocolo que transfere confiavelmente dados por um canal que pode corromper ou perder pacotes; nos “Exercícios de fixação” pediremos a você que projete a FSM destinatária para rdt3.0. A Figura 3.16 mostra como o protocolo funciona sem pacotes perdidos ou atrasados e como manipula pacotes de dados perdidos. Nessa figura, a passagem do tempo ocorre do topo do diagrama para baixo. Note que o instante de recebimento de um pacote é necessariamente posterior ao instante de envio de um pacote, como resultado de atrasos de transmissão e de propagação. Nas figuras 3.16(b-d), os colchetes do lado remetente indicam os instantes em que o temporizador foi acionado e, mais tarde, os instantes em que ele parou. Vários dos aspectos mais sutis desse protocolo são examinados nos exercícios ao final deste capítulo. Como os números de sequência se alternam entre 0 e 1, o protocolo rdt3.0 às vezes é conhecido como **protocolo bit alternante**.

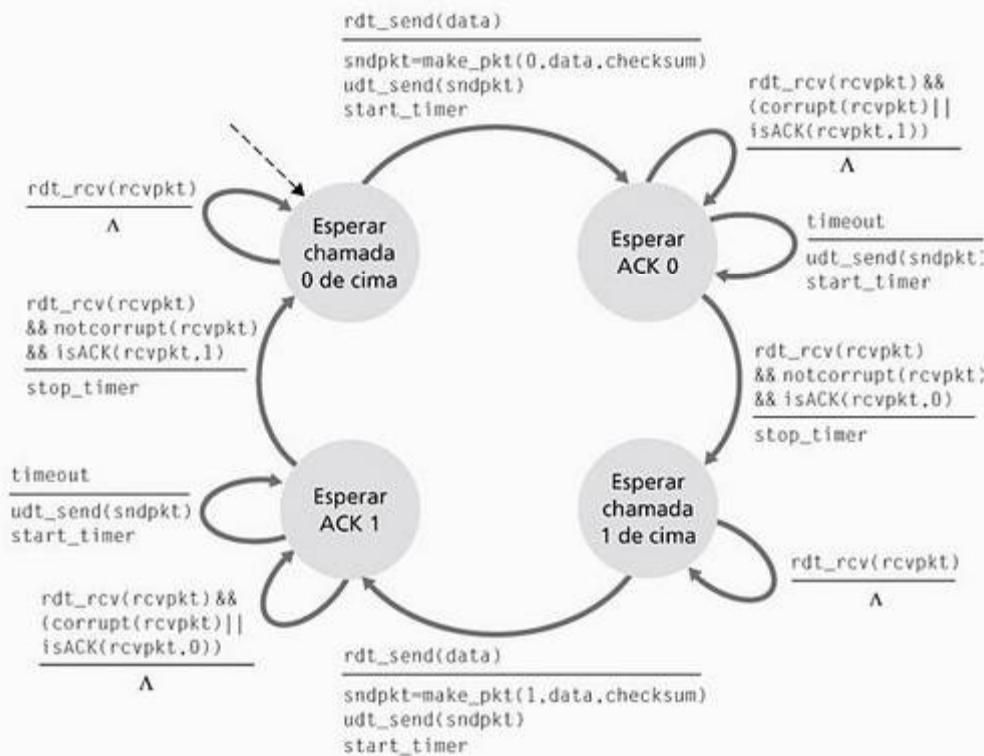


Figura 3.15 rdt3.0 remetente

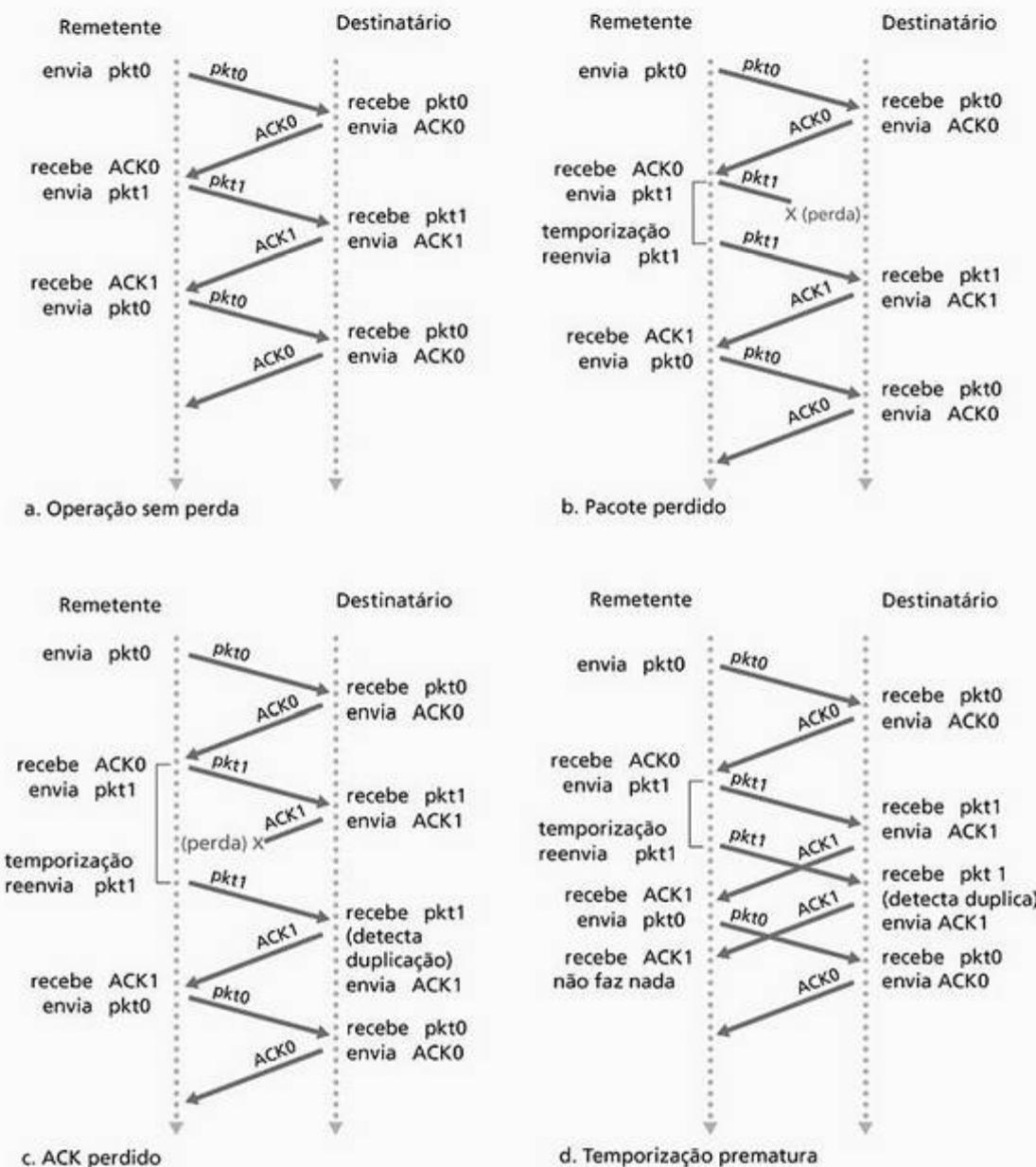


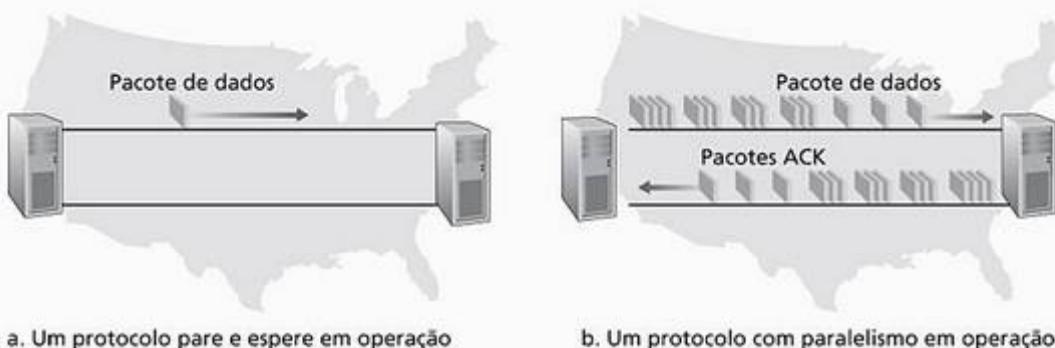
Figura 3.16 Operação do rdt3.0, o protocolo bit alternante

Agora já reunimos os elementos fundamentais de um protocolo de transferência de dados. Somas de verificação, números de sequência, temporizadores e pacotes de reconhecimento negativo e positivo — cada um desempenha um papel crucial e necessário na operação do protocolo. Temos agora em funcionamento um protocolo de transferência confiável de dados!

3.4.2 Protocolos de transferência confiável de dados com paralelismo

O protocolo rdt3.0 é correto em termos funcionais, mas é pouco provável que alguém fique contente com o desempenho dele, particularmente nas redes de alta velocidade de hoje. No coração do problema do desempenho do rdt3.0 está o fato de ele ser um protocolo do tipo pare e espere.

Para avaliar o impacto sobre o desempenho causado pelo comportamento “pare e espere”, considere um caso ideal de dois hospedeiros, um localizado na Costa Oeste dos Estados Unidos e outro na Costa Leste, como mostra a Figura 3.17.

**Figura 3.17** Protocolo pare e espere versus protocolo com paralelismo

O atraso de propagação de ida e volta à velocidade da luz, T_{prop} , entre esses dois sistemas finais é de aproximadamente 30 milissegundos. Suponha que eles estejam conectados por um canal com capacidade de transmissão, R , de 1 gigabit (10^9 bits) por segundo. Para um tamanho de pacote, L , de 1 kbyte (8 mil bits), incluindo o campo de cabeçalho e também o de dados, o tempo necessário para realmente transmitir o pacote para o enlace de 1 Gbps é:

$$t_{trans} = \frac{L}{R} = \frac{8.000 \text{ bits/pacote}}{10^9 \text{ bits/seg}} = 8 \text{ microssegundos}$$

A Figura 3.18(a) mostra que, com nosso protocolo pare e espere, se o remetente começar a enviar o pacote em $t = 0$, então em $t = L/R = 8$ microssegundos, o último bit entrará no canal do lado remetente. O pacote então faz sua jornada de 15 milissegundos atravessando o país, com o último bit do pacote emergindo no destinatário em $t = RTT/2 + L/R = 15,008$ milissegundos. Supondo, para simplificar, que pacotes ACK sejam extremamente pequenos (para podermos ignorar seu tempo de transmissão) e que o destinatário pode enviar um ACK logo que receber o último bit de um pacote de dados, o ACK emergirá de volta no remetente em $t = RTT + L/R = 30,008$ milissegundos. Nesse ponto, o remetente agora poderá transmitir a próxima mensagem. Assim, em 30,008 milissegundos, o remetente esteve enviando por apenas 0,008 milissegundo. Se definirmos a **utilização** do remetente (ou do canal) como a fração de tempo em que o remetente está realmente ocupado enviando bits para dentro do canal, a análise da Figura 3.18(a) mostra que o protocolo pare e espere tem uma utilização do remetente U_{remet} bastante desanimadora, de:

$$U_{remet} = \frac{L/R}{RTT + L/R} = \frac{0,008}{30,008} = 0,00027$$

Portanto, o remetente ficou ocupado apenas 2,7 centésimos de 1 por cento do tempo! Visto de outra maneira, ele só foi capaz de enviar 1.000 bytes em 30,008 milissegundos, uma vazão efetiva de apenas 267 kbps — mesmo estando disponível um enlace de 1 gigabit por segundo! Imagine o infeliz gerenciador de rede que acabou de pagar uma fortuna para ter capacidade de enlace da ordem de gigabits, mas consegue uma vazão de apenas 267 quilobits por segundo! Este é um exemplo gráfico de como protocolos de rede podem limitar as capacidades oferecidas pelo hardware subjacente de rede. Além disso, desprezamos também os tempos de processamento de protocolo das camadas inferiores no remetente e no destinatário, bem como os atrasos de processamento e de fila que ocorreriam em quaisquer roteadores intermediários existentes entre o remetente e o destinatário. Incluir esses efeitos serviria apenas para aumentar ainda mais o atraso e piorar ainda mais o fraco desempenho.

A solução para esse problema de desempenho em especial é simples: em vez de operar em modo pare e espere, o remetente é autorizado a enviar vários pacotes sem esperar por reconhecimentos, como mostra a Figura 3.17(b). A Figura 3.18(b) mostra que, se um remetente for autorizado a transmitir três pacotes antes de ter de

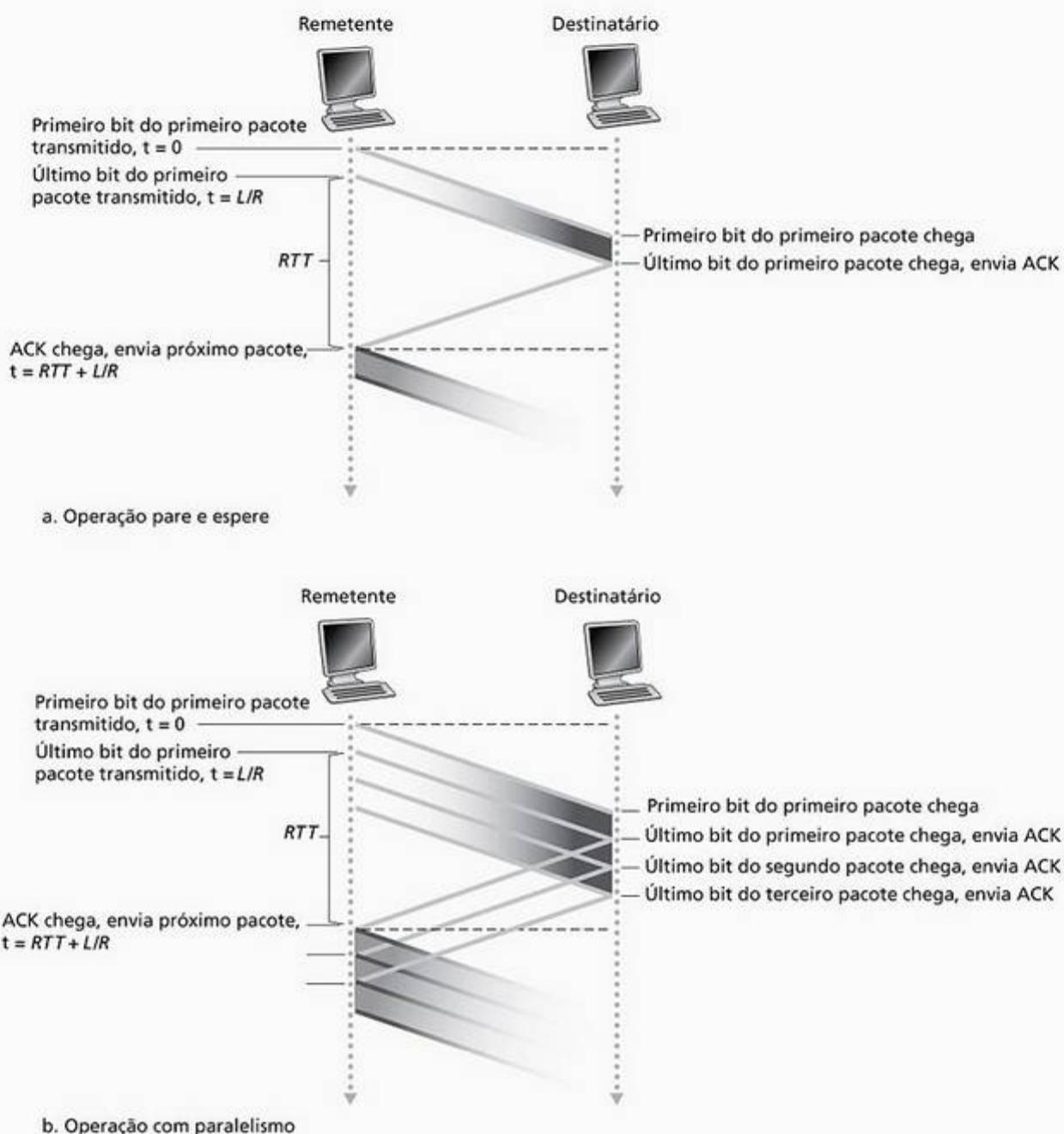


Figura 3.18 Envio com pare e espere e com paralelismo

esperar por reconhecimentos, sua utilização será essencialmente triplicada. Uma vez que os muitos pacotes em trânsito entre remetente e destinatário podem ser visualizados como se estivessem enchendo uma tubulação, essa técnica é conhecida, em inglês, como pipelining (tubulação). Porém, como essa expressão é difícil de traduzir para o português, preferimos usar ‘paralelismo’, embora a transmissão de dados seja de fato sequencial. O paralelismo gera as seguintes consequências para protocolos de transferência confiável de dados:

A faixa de números de sequência tem de ser ampliada, uma vez que cada pacote em trânsito (sem contar as retransmissões) precisa ter um número de sequência exclusivo e pode haver vários pacotes não reconhecidos em trânsito.

Os lados remetente e destinatário dos protocolos podem ter de reservar buffers para mais de um pacote. No mínimo, o remetente terá de providenciar buffers para pacotes que foram transmitidos, mas que ainda não foram reconhecidos. O buffer de pacotes corretamente recebidos pode também ser necessário no destinatário, como discutiremos a seguir.

A faixa de números de sequência necessária e as necessidades de buffer dependerão da maneira como um protocolo de transferência de dados responde a pacotes perdidos, corrompidos e demasiadamente atrasados. Duas abordagens básicas em relação à recuperação de erros com paralelismo podem ser identificadas: **Go-Back-N** e **repetição seletiva**.

3.4.3 Go-Back-N

Em um protocolo **Go-Back-N** (GBN), o remetente é autorizado a transmitir múltiplos pacotes (se disponíveis) sem esperar por um reconhecimento, mas fica limitado a ter não mais do que algum número máximo permitido, N , de pacotes não reconhecidos. Nesta seção, descreveremos o protocolo GBN com detalhes. Mas antes de continuar a leitura, convidamos você para se divertir com o applet GBN (incrível!) no Companion site Web.

A Figura 3.19 mostra a visão que o remetente tem da faixa de números de sequência em um protocolo GBN.

Se definirmos **base** como o número de sequência do mais antigo pacote não reconhecido e **nextseqnum** como o menor número de sequência não utilizado (isto é, o número de sequência do próximo pacote a ser enviado), então quatro intervalos na faixa de números de sequência poderão ser identificados. Os números de sequência no intervalo $[0, \text{base}-1]$ correspondem aos pacotes que já foram transmitidos e reconhecidos. O intervalo $[\text{base}, \text{nextseqnum}-1]$ corresponde aos pacotes que foram enviados, mas ainda não foram reconhecidos. Os números de sequência no intervalo $[\text{nextseqnum}, \text{base}+N-1]$ podem ser usados para pacotes que podem ser enviados imediatamente, caso cheguem dados vindos da camada superior. Finalmente, números de sequência maiores ou iguais a $\text{base}+N$ não podem ser usados até que um pacote não reconhecido que esteja pendente seja reconhecido (especificamente, o pacote cujo número de sequência é base).

Como sugere a Figura 3.19, a faixa de números de sequência permitidos para pacotes transmitidos mas ainda não reconhecidos pode ser vista como uma 'janela' de *tamanho N* sobre a faixa de números de sequência. À medida que o protocolo opera, essa janela se desloca para a frente sobre o espaço de números de sequência. Por essa razão, N é frequentemente denominado **tamanho de janela** e o protocolo GBN em si, **protocolo de janela deslizante** (*sliding-window protocol*). É possível que você esteja pensando que razão teríamos, em primeiro lugar, para limitar o número de pacotes pendentes não reconhecidos a um valor N . Por que não permitir um número ilimitado desses pacotes? Veremos na Seção 3.5 que o controle de fluxo é uma das razões para impor um limite ao remetente. Examinaremos outra razão para isso na Seção 3.7, quando estudarmos o controle de congestionamento do TCP.

Na prática, o número de sequência de um pacote é carregado em um campo de comprimento fixo no cabeçalho do pacote. Se k for o número de bits no campo de número de sequência do pacote, a faixa de números de sequência será então $[0, 2^k - 1]$. Com uma faixa finita de números de sequência, toda a aritmética que envolver números de sequência deverá ser feita usando aritmética de módulo 2^k . (Em outras palavras, o espaço do número de sequência pode ser imaginado como um anel de tamanho 2^k , em que o número de sequência $2^k - 1$

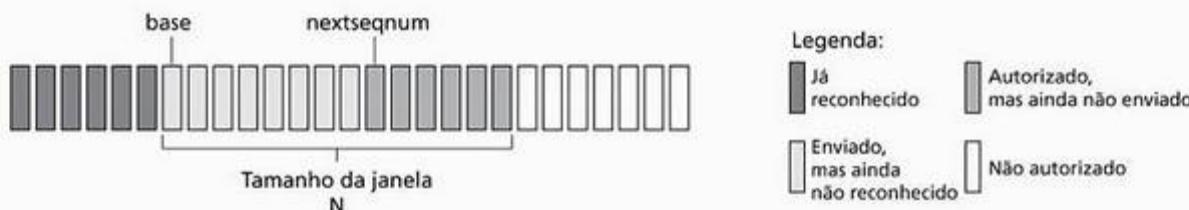


Figura 3.19 Visão do remetente para os números de sequência no protocolo Go-Back-N

é imediatamente seguido pelo número de sequência 0.) Lembre-se de que `rdt3.0` tem um número de sequência de 1 bit e uma faixa de números de sequência de [0,1]. Vários problemas ao final deste capítulo tratam das consequências de uma faixa finita de números de sequência. Veremos na Seção 3.5 que o TCP tem um campo de número de sequência de 32 bits, onde os números de sequência do TCP contam bytes na cadeia de bytes, em vez de pacotes.

As figuras 3.20 e 3.21 descrevem uma FSM estendida dos lados remetente e destinatário de um protocolo GBN baseado em ACK, mas sem NAK.

Referimo-nos a essa descrição de FSM como *FSM estendida* porque adicionamos variáveis (semelhantes às variáveis de linguagem de programação) para `base` e `nextseqnum`; também adicionamos operações sobre essas variáveis e ações condicionais que as envolvem. Note que a especificação da FSM estendida agora está começando a parecer um pouco com uma especificação de linguagem de programação. [Bochman, 1984] fornece um excelente levantamento sobre extensões adicionais às técnicas FSM, bem como sobre outras técnicas para especificação de protocolos baseadas em linguagens.

O remetente GBN deve responder a três tipos de eventos:

Chamada vinda de cima. Quando `rdt_send()` é chamado de cima, o remetente primeiramente verifica se a janela está cheia, isto é, se há N pacotes pendentes não reconhecidos. Se a janela não estiver cheia, um pacote é criado e enviado e as variáveis são adequadamente atualizadas. Se estiver cheia, o remetente apenas devolve os dados à camada superior — uma indicação implícita de que a janela está cheia. Presumivelmente, a camada superior então teria de tentar outra vez mais tarde. Em uma implementação real, o remetente muito provavelmente teria colocado esses dados em um buffer (mas não os teria enviado imediatamente) ou teria um mecanismo de sincronização (por exemplo, um semáforo ou uma flag) que permitiria que a camada superior chamassem `rdt_send()` somente quando as janelas não estivessem cheias.

Recebimento de um ACK. Em nosso protocolo GBN, um reconhecimento de pacote com número de sequência n seria tomado como um **reconhecimento cumulativo**, indicando que todos os pacotes com número de sequência até e inclusive n tinham sido corretamente recebidos no destinatário. Voltaremos a esse assunto em breve, quando examinarmos o lado destinatário do GBN.

Um esgotamento de temporização. O nome ‘Go-Back-N’ deriva do comportamento do remetente em relação a pacotes perdidos ou demasiadamente atrasados. Como no protocolo pare e espere, um temporizador é usado para recuperar a perda de dados ou reconhecer pacotes. Se ocorrer o esgotamento da temporização, o remetente reenvia *todos* os pacotes que tinham sido previamente enviados mas que ainda não tinham sido reconhecidos. Nossa remetente da Figura 3.20 usa apenas um único temporizador, que pode ser imaginado como um temporizador para o mais antigo pacote já transmitido mas que ainda não foi reconhecido. Se for recebido um ACK e ainda houver pacotes adicionais transmitidos mas ainda não reconhecidos, o temporizador será reiniciado. Se não houver nenhum pacote pendente não reconhecido, o temporizador será desligado.

As ações do destinatário no GBN também são simples. Se um pacote com número de sequência n for recebido corretamente e estiver na ordem (isto é, os últimos dados entregues à camada superior vierem de um pacote com número de sequência $n - 1$), o destinatário enviará um ACK para o pacote n e entregará a porção dos dados do pacote à camada superior. Em todos os outros casos, o destinatário descarta o pacote e reenvia um ACK para o pacote mais recente que foi recebido na ordem correta. Dado que os pacotes são entregues à camada superior um por vez, se o pacote k tiver sido recebido e entregue, então todos os pacotes com número de sequência menores do que k também terão sido entregues. Assim, o uso de reconhecimentos cumulativos é uma escolha natural para o GBN.

Em nosso protocolo GBN, o destinatário descarta os pacotes que chegam fora de ordem. Embora pareça bobagem e perda de tempo descartar um pacote corretamente recebido (mas fora de ordem), existem justificativas para isso. Lembre-se de que o destinatário deve entregar dados na ordem certa à camada superior. Suponha agora que o pacote n esteja sendo esperado, mas quem chega é o pacote $n + 1$. Como os dados devem ser entregues na ordem certa, o destinatário *poderia* conservar o pacote $n + 1$ no buffer (salvá-lo) e entregar esse pacote à camada

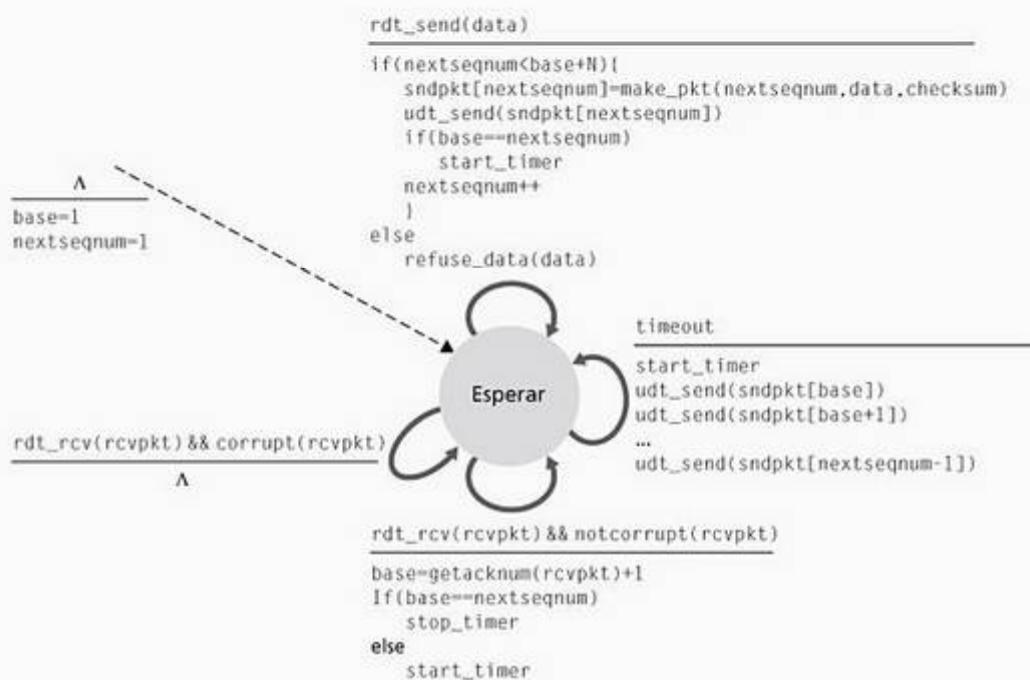


Figura 3.20 Descrição da FSM estendida do remetente GBN

superior mais tarde, após ter recebido o pacote n . Contudo, se o pacote n for perdido, os pacotes n e $n + 1$ serão ambos finalmente retransmitidos como resultado da regra de retransmissão do GBN no remetente. Assim, o destinatário pode simplesmente descartar o pacote $n + 1$. A vantagem dessa abordagem é a simplicidade da manipulação de buffers no destinatário — o destinatário não precisa colocar no buffer nenhum pacote que esteja fora de ordem. Desse modo, enquanto o remetente deve manter os limites superior e inferior de sua janela e a posição de `nextseqnum` dentro dessa janela, a única informação que o destinatário precisa manter é o número de sequência do próximo pacote esperado conforme a ordem. Esse valor é retido na variável `expectedseqnum` mostrada na FSM destinatária da Figura 3.21. Evidentemente, a desvantagem de jogar fora um pacote recebido corretamente é que a retransmissão subsequente desse pacote pode ser perdida ou ficar truncada, caso em que ainda mais retransmissões seriam necessárias.

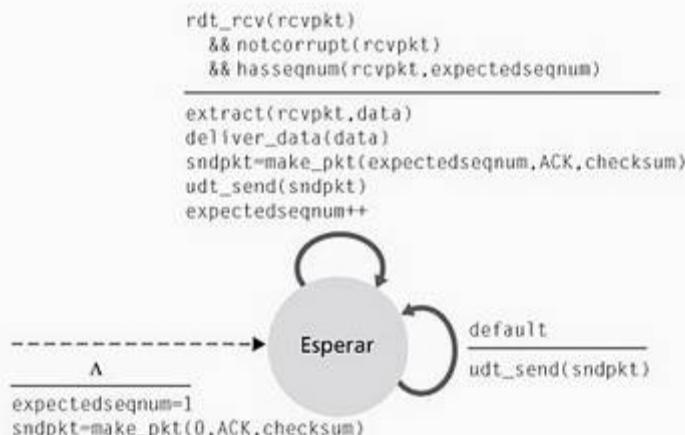


Figura 3.21 Descrição da FSM estendida do destinatário GBN

A Figura 3.22 mostra a operação do protocolo GBN para o caso de um tamanho de janela de quatro pacotes. Por causa da limitação do tamanho dessa janela, o remetente envia os pacotes de 0 a 3, mas, em seguida, tem de esperar que um ou mais desses pacotes sejam reconhecidos antes de prosseguir. E, à medida que cada ACK sucesivo (por exemplo, ACK0 e ACK1) é recebido, a janela se desloca para a frente e o remetente pode transmitir um novo pacote (pkt4 e pkt5, respectivamente). Do lado destinatário, o pacote 2 é perdido. Desse modo, verifica-se que os pacotes 3, 4 e 5 estão fora de ordem e, portanto, são descartados.

Antes de encerrarmos nossa discussão sobre o GBN, devemos ressaltar que uma implementação desse protocolo em uma pilha de protocolo provavelmente seria estruturada de modo semelhante à da FSM estendida da Figura 3.20. A implementação provavelmente também seria estruturada sob a forma de vários procedimentos que implementam as ações a serem executadas em resposta aos vários eventos que podem ocorrer. Nessa **programação baseada em eventos**, os vários procedimentos são chamados (invocados) por outros procedimentos presentes na pilha de protocolo ou como resultado de uma interrupção. No remetente, esses eventos seriam: (1) uma chamada pela entidade da camada superior invocando `rdt_send()`, (2) uma interrupção pelo temporizador e (3) uma chamada pela camada inferior invocando `rdt_rcv()` quando chega um pacote. Os exercícios de programação ao final deste capítulo lhe darão a chance de implementar de verdade essas rotinas em um ambiente de rede simulado, mas realista.

Salientamos que o protocolo GBN incorpora quase todas as técnicas que encontraremos quando estudarmos, na Seção 3.5, os componentes de transferência confiável de dados do TCP. Essas técnicas incluem a utilização de números de sequência, reconhecimentos cumulativos, somas de verificação e uma operação de esgotamento de temporização/retransmissão.

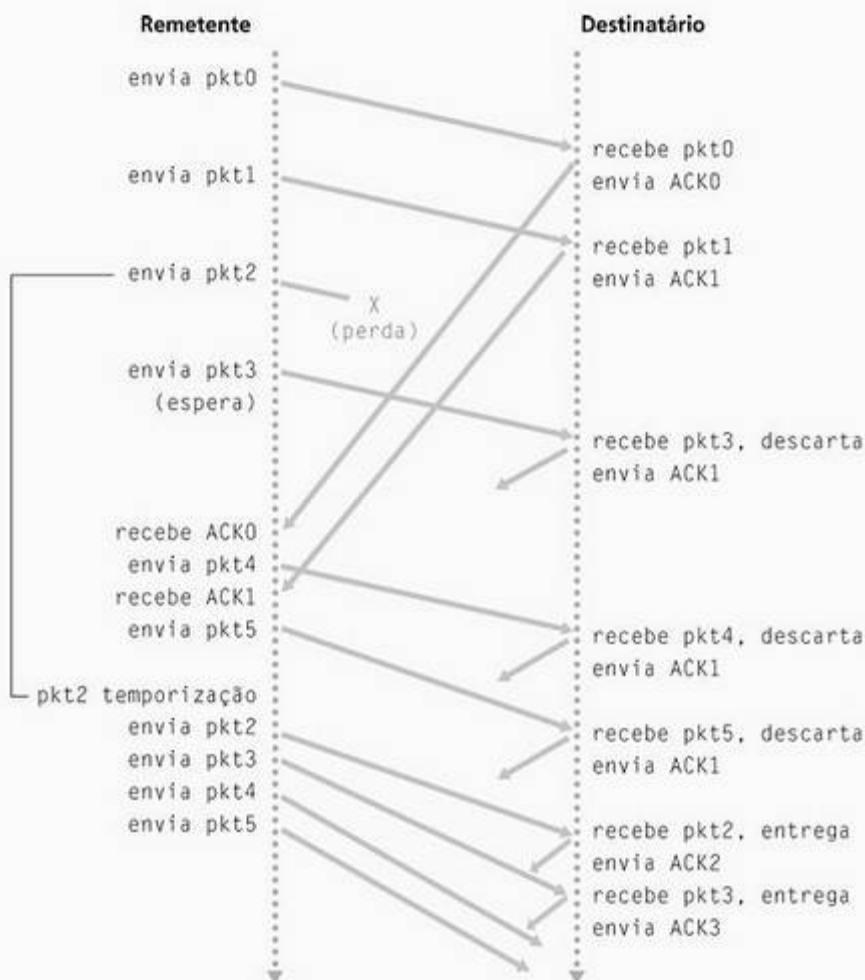


Figura 3.22 GBN em operação

3.4.4 Repetição seletiva (SR)

O protocolo GBN permite que o remetente potencialmente “encha a rede” com pacotes na Figura 3.17, evitando, assim, os problemas de utilização de canal observados em protocolos do tipo pare e espere. Há, contudo, casos em que o próprio GBN sofre com problemas de desempenho. Em especial, quando o tamanho da janela e o produto entre o atraso e a largura de banda são grandes, pode haver muitos pacotes pendentes na rede. Assim, um único erro de pacote pode fazer com que o GBN retransmita um grande número de pacotes — muitos deles desnecessariamente. À medida que aumenta a probabilidade de erros no canal, a rede pode ficar lotada com essas retransmissões desnecessárias. Imagine se, em uma conversa, toda vez que uma palavra fosse pronunciada de maneira truncada as outras mil que a circundam (por exemplo, um tamanho de janela de mil palavras) tivessem de ser repetidas. A conversa sofreria atrasos devido a todas essas palavras reiteradas.

Como o próprio nome sugere, protocolos de repetição seletiva (*selective repeat* — SR) evitam retransmissões desnecessárias porque fazem o remetente retransmitir somente os pacotes suspeitos de terem sido recebidos com erro (isto é, que foram perdidos ou corrompidos) no destinatário. Essa retransmissão individual, somente quando necessária, exige que o destinatário reconheça *individualmente* os pacotes recebidos de modo correto. Uma janela de tamanho N será usada novamente para limitar o número de pacotes pendentes não reconhecidos dentro da rede. Contudo, ao contrário do GBN, o remetente já terá recebido ACKs para alguns dos pacotes na janela. A Figura 3.23 mostra a visão que o protocolo de SR remetente tem do espaço do número de sequência. A Figura 3.24 detalha as várias ações executadas pelo protocolo SR remetente.

O protocolo SR destinatário reconhecerá um pacote corretamente recebido esteja ele ou não na ordem certa. Pacotes fora de ordem ficam no buffer até que todos os pacotes faltantes (isto é, os que têm números de sequência menores) sejam recebidos, quando então um conjunto de pacotes poderá ser entregue à camada superior na ordem correta. A Figura 3.25 apresenta as várias ações realizadas pelo protocolo SR destinatário.

A Figura 3.26 apresenta um exemplo de operação do protocolo SR quando ocorre perda de pacotes. Note que, nessa figura, o destinatário inicialmente armazena os pacotes 3, 4 e 5 e os entrega juntamente com o pacote 2 à camada superior, quando o pacote 2 é finalmente recebido.

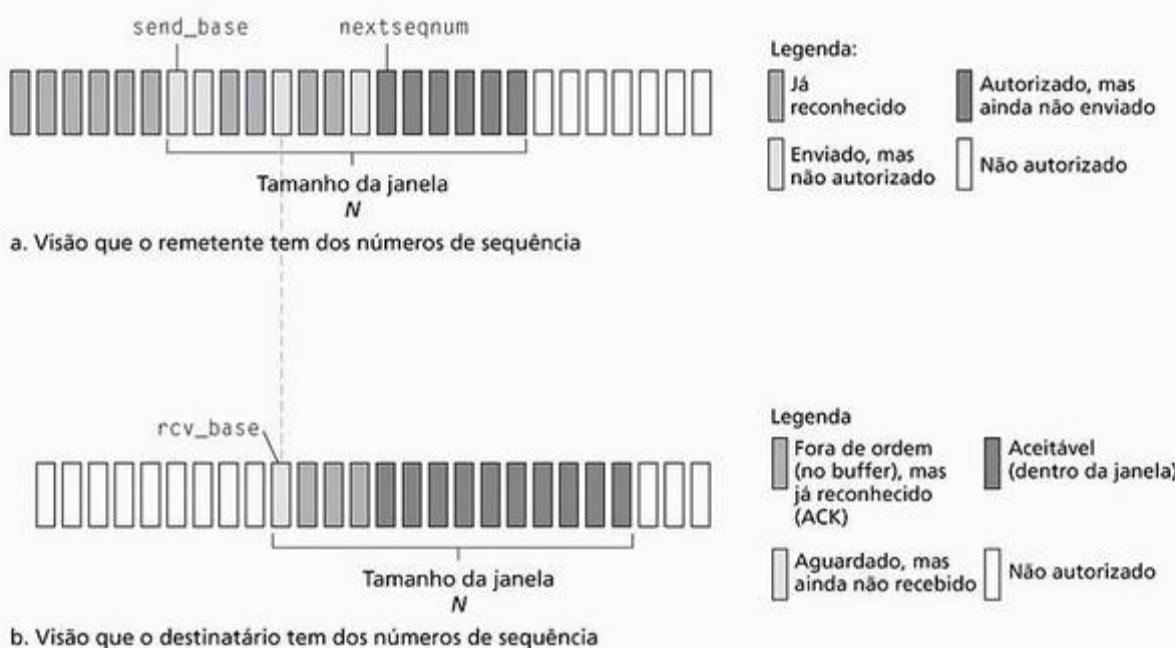


Figura 3.23 Visões que os protocolos SR remetente e destinatário têm do espaço de número de sequência

1. Dados recebidos de cima. Quando são recebidos dados de cima, o protocolo SR remetente verifica o próximo número de sequência disponível para o pacote. Se o número de sequência está dentro da janela do remetente, os dados são empacotados e enviados; do contrário, eles são armazenados ou devolvidos à camada superior para transmissão posterior, como acontece no GBN.
2. Esgotamento de temporização. Novamente são usados temporizadores para proteção contra perda de pacotes. Contudo, cada pacote agora deve ter seu próprio temporizador lógico, já que apenas um pacote será transmitido quando a temporização se esgotar. Um único hardware de temporizador pode ser usado para emular a operação de múltiplos temporizadores lógicos [Varghese, 1997].
3. ACK recebido. Se for recebido um ACK, o SR remetente marcará aquele pacote como recebido, contanto que esteja na janela. Se o número de sequência do pacote for igual a `send_base`, a base da janela se deslocará para a frente até o pacote não reconhecido que tiver o menor número de sequência. Se a janela se deslocar e houver pacotes não transmitidos com números de sequência que agora caem dentro da janela, esses pacotes serão transmitidos.

Figura 3.24 Eventos e ações do protocolo SR remetente

É importante notar que na etapa 2 da Figura 3.25 o destinatário reconhece novamente (em vez de ignorar) pacotes já recebidos com certos números de sequência que estão abaixo da atual base da janela. É bom que você se convença de que esse reconhecimento duplo é de fato necessário. Dados os espaços dos números de sequência do remetente e do destinatário na Figura 3.23, por exemplo, se não houver ACK para pacote com número `send_base` propagando-se do destinatário ao remetente, este acabará retransmitindo o pacote `send_base`, embora esteja claro (para nós, e não para o remetente!) que o destinatário já recebeu esse pacote. Caso o destinatário não reconhecesse esse pacote, a janela do remetente jamais se deslocaria para a frente! Esse exemplo ilustra um importante aspecto dos protocolos SR (e também de muitos outros). O remetente e o destinatário nem sempre têm uma visão idêntica do que foi recebido corretamente e do que não foi. Para protocolos SR, isso significa que as janelas do remetente e do destinatário nem sempre coincidirão.

A falta de sincronização entre as janelas do remetente e do destinatário tem importantes consequências quando nos defrontamos com a realidade de uma faixa finita de números de sequência. Considere o que poderia acontecer, por exemplo, com uma faixa finita de quatro números de sequência de pacotes (0, 1, 2, 3) e um tamanho de janela de três. Suponha que os pacotes de 0 a 2 sejam transmitidos, recebidos e reconhecidos corretamente no destinatário. Nesse ponto, a janela do destinatário está sobre o quarto, o quinto e o sexto pacotes, que têm os números de sequência 3, 0 e 1, respectivamente. Agora, considere dois cenários. No primeiro, mostrado na Figura

1. Pacote com número de sequência no intervalo $[rcv_base, rcv_base+N-1]$ foi corretamente recebido. Nesse caso, o pacote recebido cai dentro da janela do destinatário e um pacote ACK seletivo é devolvido ao remetente. Se o pacote não tiver sido recebido anteriormente, irá para o buffer. Se esse pacote tiver um número de sequência igual à base da janela destinatária (`rcv_base` na Figura 3.22), então ele, e quaisquer outros pacotes anteriormente armazenados no buffer e numerados consecutivamente (começando com `rcv_base`), serão entregues à camada superior. A janela destinatária é então deslocada para a frente de acordo com o número de pacotes entregues à camada superior. Como exemplo, considere a Figura 3.26. Quando um pacote com número de sequência `rcv_base=2` é recebido, ele e os pacotes 3, 4 e 5 podem ser entregues à camada superior.
2. Pacote com número de sequência em $[rcv_base-N, rcv_base-1]$ é recebido. Nesse caso, um ACK deve ser gerado mesmo que esse pacote já tenha sido reconhecido anteriormente pelo destinatário.
3. Qualquer outro. Ignore o pacote

Figura 3.25 Eventos e ações do protocolo SR destinatário

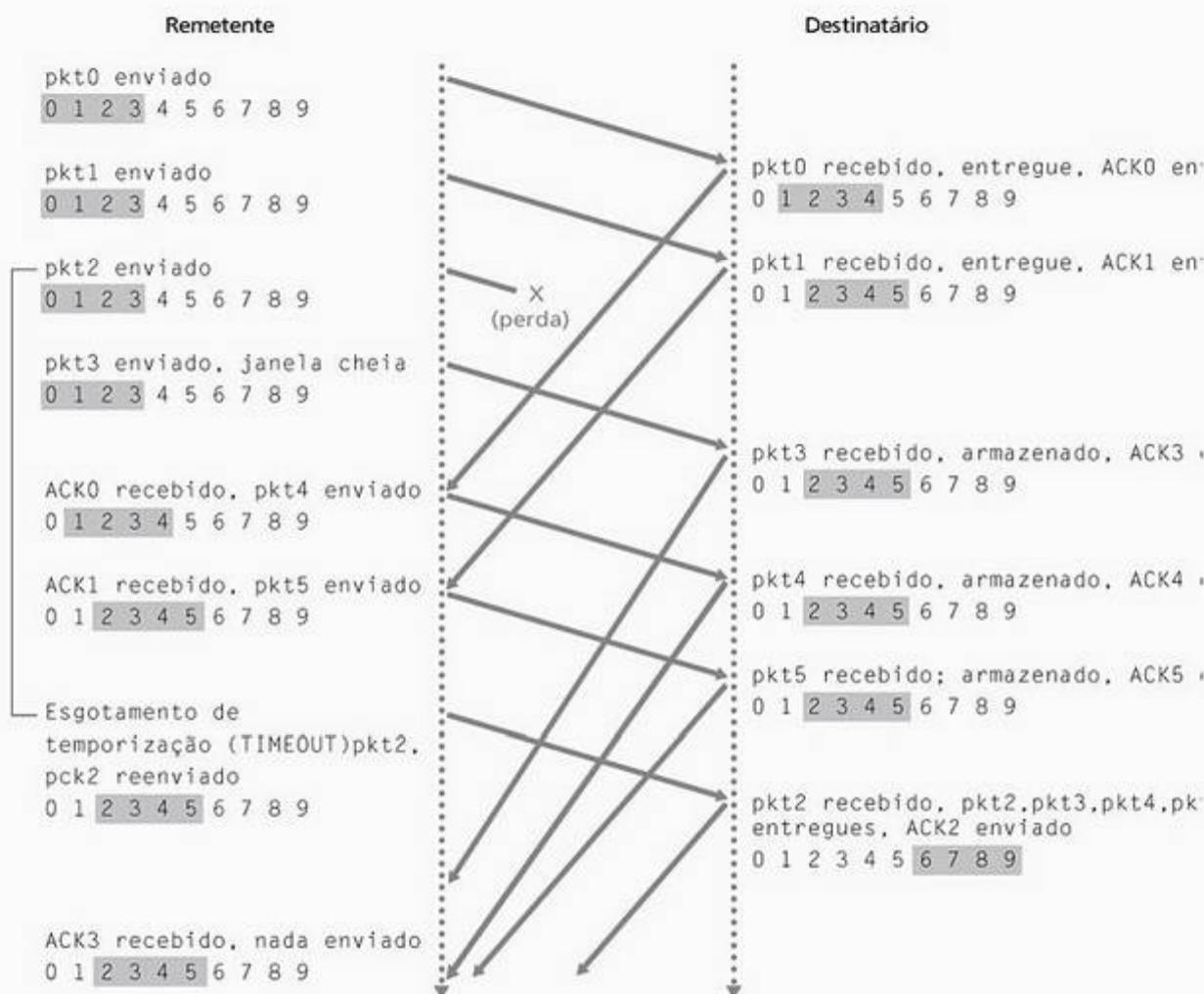


Figura 3.26 Operação SR

3.27(a), os ACKs para os três primeiros pacotes foram perdidos e o remetente retransmite esses pacotes. Assim, o que o destinatário recebe em seguida é um pacote com o número de sequência 0 — uma cópia do primeiro pacote enviado.

No segundo cenário, mostrado na Figura 3.27(b), os ACKs para os três primeiros pacotes foram entregues corretamente. Assim, o remetente desloca sua janela para a frente e envia o quarto, o quinto e o sexto pacotes com os números de sequência 3, 0 e 1, respectivamente. O pacote com o número de sequência 3 é perdido, mas o pacote com o número de sequência 0 chega — um pacote que contém dados novos.

Agora, na Figura 3.27, considere o ponto de vista do destinatário, que tem uma cortina imaginária entre o remetente e ele, já que o destinatário não pode ‘ver’ as ações executadas pelo remetente. Tudo o que o destinatário observa é a sequência de mensagens que ele recebe do canal e envia para o canal. No que concerne a ele, os dois cenários da Figura 3.27 são *idênticos*. Não há nenhum modo de distinguir a retransmissão do primeiro pacote da transmissão original do quinto pacote. Fica claro que um tamanho de janela que seja igual ao tamanho do espaço de numeração sequencial menos 1 não vai funcionar. Mas qual deve ser o tamanho da janela? Um problema ao final deste capítulo pede que você demonstre que o tamanho da janela pode ser menor ou igual à metade do tamanho do espaço de numeração sequencial para os protocolos SR.

No Companion Website, você encontrará um applet que anima a operação do protocolo SR. Tente realizar os mesmos experimentos feitos com o applet GBN. Os resultados combinam com o que você espera?

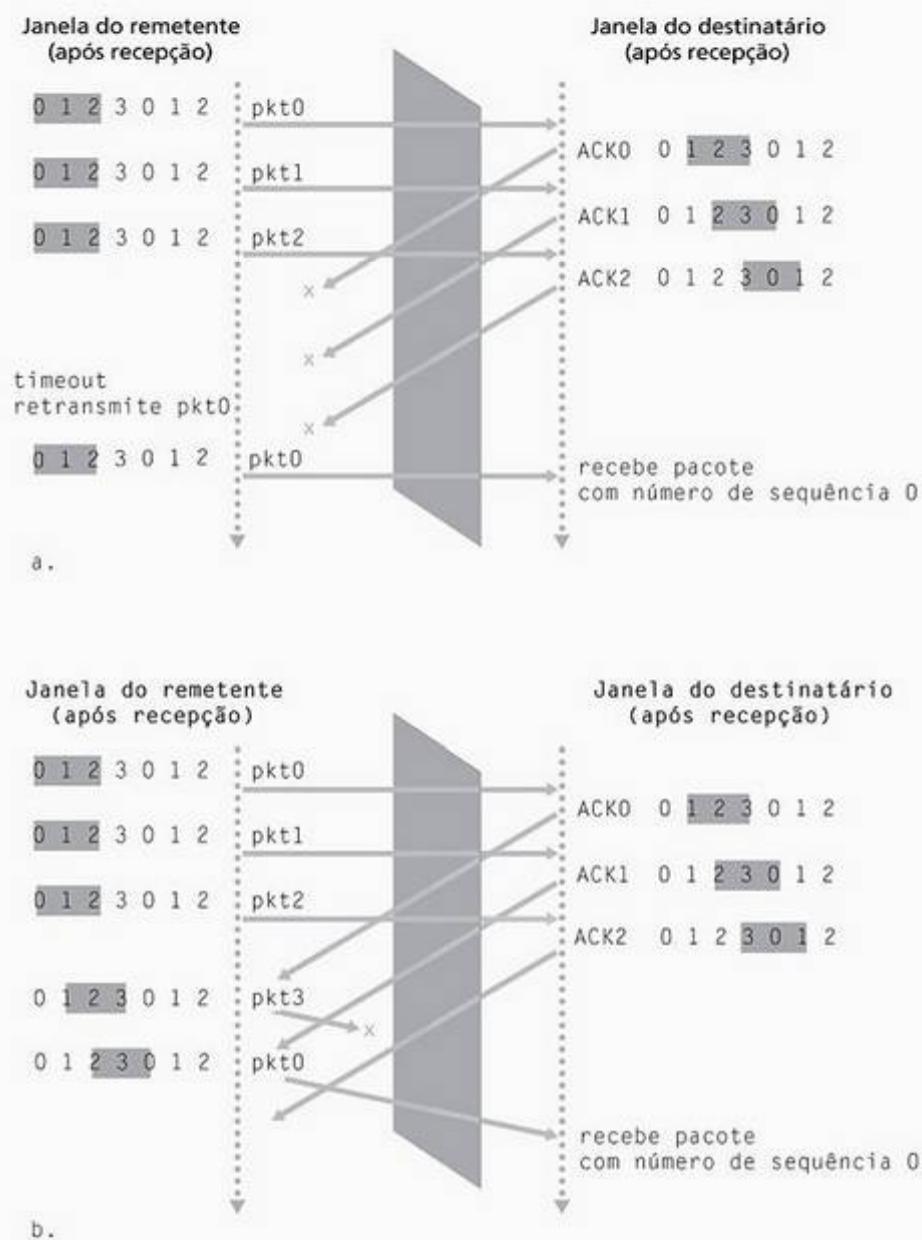


Figura 3.27 Dilema do remetente SR com janelas muito grandes: um novo pacote ou uma retransmissão?

Com isso, concluímos nossa discussão sobre protocolos de transferência confiável de dados. Percorremos um longo caminho e apresentamos numerosos mecanismos que, juntos, proveem transferência confiável de dados. A Tabela 3.1 resume esses mecanismos. Agora que já vimos todos eles em operação e podemos enxergar ‘o quadro geral’, aconselhamos que você leia novamente esta seção para perceber como esses mecanismos foram adicionados pouco a pouco, de modo a abordar modelos (realistas) de complexidade crescente do canal que conecta o remetente ao destinatário ou para melhorar o desempenho dos protocolos.

Encerraremos nossa explanação considerando uma premissa remanescente em nosso modelo de canal subjacente. Lembre-se de que admitimos que pacotes não podem ser reordenados dentro do canal entre o remetente e o destinatário. Esta é uma premissa em geral razoável quando o remetente e o destinatário estão conectados por um único fio físico. Contudo, quando o ‘canal’ que está conectando os dois é uma rede, pode ocorrer reordenação de pacotes. Uma manifestação da reordenação de pacotes é que podem aparecer cópias antigas de um pacote com

Mecanismo	Utilização, Comentários
Soma de verificação	Usado para detectar erros de bits em um pacote transmitido.
Temporizador	Usado para controlar a temporização/retransmitir um pacote, possivelmente porque o pacote (ou seu ACK) foi perdido dentro do canal. Como pode ocorrer esgotamento de temporização quando um pacote está atrasado, mas não perdido (exaurimento de temporização prematuro), ou quando um pacote foi recebido pelo destinatário mas o ACK remetente-destinatário foi perdido, um destinatário pode receber cópias duplicadas de um pacote.
Número de sequência	Usado para numeração sequencial de pacotes de dados que transitam do remetente ao destinatário. Lacunas nos números de sequência de pacotes recebidos permitem que o destinatário detecte um pacote perdido. Pacotes com números de sequência duplicados permitem que o destinatário detecte cópias duplicadas de um pacote.
Reconhecimento	Usado pelo destinatário para avisar o remetente que um pacote ou conjunto de pacotes foi recebido corretamente. Reconhecimentos normalmente portam o número de sequência do pacote, ou pacotes, que estão sendo reconhecidos. Reconhecimentos podem ser individuais ou cumulativos, dependendo do protocolo.
Reconhecimento negativo	Usado pelo destinatário para avisar o remetente que um pacote não foi recebido corretamente. Reconhecimentos negativos normalmente portam o número de sequência do pacote que não foi recebido corretamente.
Janela, paralelismo	O remetente pode ficar restrito a enviar somente pacotes com números de sequência que caem dentro de uma determinada faixa. Permitindo que vários pacotes sejam transmitidos, ainda que não reconhecidos, a utilização do remetente pode ser aumentada em relação ao modo de operação pare e espere. Em breve veremos que o tamanho da janela pode ser estabelecido com base na capacidade do destinatário receber e fazer buffer de mensagens ou no nível de congestionamento na rede, ou em ambos.

Tabela 3.1 Resumo de mecanismos de transferência confiável de dados e sua utilização

número de sequência ou de reconhecimento x , mesmo que nem a janela do remetente nem a do destinatário contenham x . Com a reordenação de pacotes, podemos considerar que o canal essencialmente usa armazenamento de pacotes e emite-os espontaneamente em algum momento qualquer do futuro. Como números de sequência podem ser reutilizados, devemos tomar um certo cuidado para nos prevenir contra esses pacotes duplicados. A abordagem adotada na prática é garantir que um número de sequência não seja reutilizado até que o remetente esteja ‘certo’ de que nenhum pacote enviado anteriormente com número de sequência x está na rede. Isso é feito admitindo que um pacote não pode ‘viver’ na rede mais do que um certo tempo máximo fixado. As extensões do TCP para redes de alta velocidade [RFC 1323] usam um tempo de vida máximo de pacote de aproximadamente três minutos. [Sunshine, 1978] descreve um método para usar números de sequência tais que os problemas de reordenação podem ser completamente evitados.

3.5 Transporte orientado para conexão: TCP

Agora que já vimos os princípios subjacentes à transferência confiável de dados, vamos voltar para o TCP — o protocolo de transporte confiável da camada de transporte, orientado para conexão, da Internet. Nesta seção, veremos que, para poder fornecer transferência confiável de dados, o TCP conta com muitos dos princípios subjacentes discutidos na seção anterior, incluindo detecção de erro, retransmissões, reconhecimentos cumulativos, temporizadores e campos de cabeçalho para números de sequência e de reconhecimento. O TCP está definido nos RFCs 793, 1122, 1323, 2018 e 2581.

3.5.1 A conexão TCP

Dizemos que o TCP é **orientado para conexão** porque, antes que um processo de aplicação possa começar a enviar dados a outro, os dois processos precisam primeiramente se ‘apresentar’ — isto é, devem enviar alguns segmentos preliminares um ao outro para estabelecer os parâmetros da transferência de dados em questão. Como parte do estabelecimento da conexão TCP, ambos os lados da conexão iniciarão muitas “variáveis de estado” (muitas das quais serão discutidas nesta seção e na Seção 3.7) associadas com a conexão TCP.

A ‘conexão’ TCP não é um circuito TDM ou FDM fim a fim, como acontece em uma rede de comutação de circuitos. Tampouco é um circuito virtual (veja o Capítulo 1), pois o estado de conexão reside inteiramente nos dois sistemas finais. Como o protocolo TCP roda somente nos sistemas finais, e não nos elementos intermediários da rede (roteadores e comutadores de camada de enlace), os elementos intermediários não mantêm estado de conexão TCP. Na verdade, os roteadores intermediários são completamente alheios às conexões TCP; eles enxergam datagramas, e não conexões.

Uma conexão TCP provê um **serviço full-duplex**: se houver uma conexão TCP entre o processo A em um hospedeiro e o processo B em outro hospedeiro, então os dados da camada de aplicação poderão fluir de A para B ao mesmo tempo em que os dados da camada de aplicação fluem de B para A. A conexão TCP é sempre **ponto a ponto**, isto é, entre um único remetente e um único destinatário. O chamado ‘multicast’ (veja a Seção 4.7) — a transferência de dados de um remetente para vários destinatários em uma única operação de envio — não é possível com o TCP. Com o TCP, dois hospedeiros é bom; três é demais!

Vamos agora examinar como uma conexão TCP é estabelecida. Suponha que um processo que roda em um hospedeiro queira iniciar a conexão com outro processo em outro hospedeiro. Lembre-se de que o processo que está iniciando a conexão é denominado *processo cliente*, enquanto o outro processo é denominado *processo servidor*. O processo de aplicação cliente primeiramente informa à camada de transporte no cliente que ele quer estabelecer uma conexão com um processo no servidor. Lembre-se (Seção 2.7) de que um programa cliente em Java faz isso emitindo o comando

```
Socket clientSocket = new Socket ("hostname", portNumber);
```

em que *hostname* é o nome do servidor e *portNumber* identifica o processo no servidor. A camada de transporte no cliente então passa a estabelecer uma conexão TCP-servidor. Discutiremos com algum detalhe o procedimento de estabelecimento de conexão ao final desta seção. Por enquanto, basta saber que o cliente primeiramente envia um segmento TCP especial; o servidor responde com um segundo segmento TCP especial e, por fim, o cliente responde novamente com um terceiro segmento especial. Os primeiros dois segmentos não contêm nenhuma “carga útil”, isto é, nenhum dado da camada de aplicação; o terceiro desses segmentos pode carregar uma carga útil. Como três segmentos são enviados entre dois hospedeiros, esse procedimento de estabelecimento de conexão é frequentemente denominado **apresentação de três vias** (3-way handshake).

Uma vez estabelecida uma conexão TCP, os dois processos de aplicação podem enviar dados um para o outro. Vamos considerar o envio de dados do processo cliente para o processo servidor. O processo cliente passa uma cadeia de dados através do socket (a porta do processo), como descrito na Seção 2.7. Tão logo passem pelo socket, os dados estão nas mãos do TCP que está rodando no cliente. Como mostra a Figura 3.28, o TCP direciona seus dados para o **buffer de envio** da conexão, que é um dos buffers reservados durante a apresentação de três vias inicial. De quando em quando, o TCP arranca grandes pedaços de dados do buffer de envio. O interessante é que a especificação do TCP [RFC 793] é muito lacônica ao indicar quando o TCP deve realmente enviar dados que estão nos buffers, determinando apenas que o TCP “deve enviar aqueles dados em segmentos segundo sua própria conveniência”. A quantidade máxima de dados que pode ser retirada e colocada em um segmento é

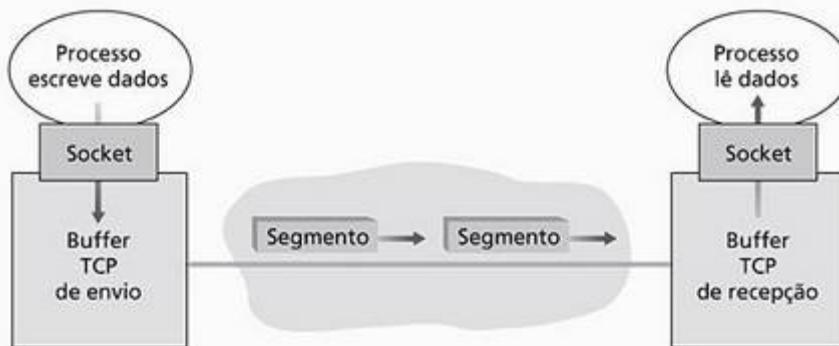


Figura 3.28 Buffers TCP de envio e de recepção



História

Vinton Cerf, Robert Kahn e TCP/IP

No início da década de 1970, as redes de comutação de pacotes começaram a proliferar. A ARPAnet — precursora da Internet — era apenas mais uma dentre tantas redes, que tinham, cada uma, seu próprio protocolo. Dois pesquisadores, Vinton Cerf e Robert Kahn, reconheceram a importância de interconectar essas redes e inventaram um protocolo inter-redes denominado TCP/IP, que quer dizer Transmission Control Protocol/Internet Protocol (protocolo de controle de transmissão/protocolo da Internet). Embora no começo Cerf e Kahn considerassem o protocolo como uma entidade única, mais tarde ele foi dividido em duas partes, TCP e IP, que operavam separadamente. Cerf e Kahn publicaram um artigo sobre o TCP/IP em maio de 1974 em *IEEE Transactions on Communication Technology* [Cerf, 1974].

O protocolo TCP/IP, que é o ‘feijão com arroz’ da Internet de hoje, foi elaborado antes dos PCs e das estações de trabalho, antes da proliferação das Ethernets e de outras tecnologias de redes locais, antes da Web, da recepção de vídeo e do bate-papo virtual. Cerf e Kahn perceberam a necessidade de um protocolo de rede que, de um lado, fornecesse amplo suporte para aplicações ainda a serem definidas e que, de outro, permitisse a interoperabilidade entre hospedeiros arbitrários e protocolos de camada de enlace.

Em 2004, Cerf e Kahn receberam o prêmio ACM Turing Award, considerado o Prêmio Nobel da Computação pelo “trabalho pioneiro sobre interligação em rede, incluindo o projeto e a implementação dos protocolos de comunicação da Internet, TCP/IP e por inspirarem liderança na área de redes.”

limitada pelo **tamanho máximo do segmento** (*maximum segment size* — MSS). O MSS normalmente é estabelecido primeiramente determinando o tamanho do maior quadro de camada de enlace que pode ser enviado pelo hospedeiro remetente local (denominado **unidade máxima de transmissão** — *maximum transmission unit* — MTU) e, em seguida, estabelecendo um MSS que garanta que um segmento TCP (quando encapsulado em um datagrama IP) caberá em um único quadro de camada de enlace. Valores comuns da MTU são 1.460 bytes, 536 bytes e 512 bytes. Também foram propostas abordagens para descobrir a MTU de Caminho (Path MTU) — o maior quadro de camada de enlace que pode ser enviado por todos os enlaces desde a fonte até o destino [RFC 1191]. Note que o MSS é a máxima quantidade de dados de camada de aplicação no segmento, e não o tamanho máximo do segmento TCP incluindo cabeçalhos. (Essa terminologia é confusa, mas temos de conviver com ela, pois já está arraigada.)

O TCP combina cada porção de dados do cliente com um cabeçalho TCP, formando, assim, **segmentos TCP**. Os segmentos são passados para baixo, para a camada de rede, onde são encapsulados separadamente dentro dos datagramas IP de camada de rede. Os datagramas IP são então enviados para dentro da rede. Quando o TCP recebe um segmento na outra extremidade, os dados do segmento são colocados no buffer de recepção da conexão, como mostra a Figura 3.28. A aplicação lê a cadeia de dados desse buffer. Cada lado da conexão tem seus próprios buffers de envio e seu próprio buffer de recepção. (Você pode ver o applet de controle de fluxo on-line em http://www.aw.com/kurose_br, que oferece uma animação dos buffers de envio e de recepção.)

Entendemos, dessa discussão, que uma conexão TCP consiste em buffers, variáveis e um socket de conexão de um processo em um hospedeiro e outro conjunto de buffers, variáveis e um socket de conexão de um processo em outro hospedeiro. Como mencionamos anteriormente, nenhum buffer nem variáveis são alocados à conexão nos elementos da rede (roteadores, comutadores e repetidores) existentes entre os hospedeiros.

3.5.2 Estrutura do segmento TCP

Agora que examinamos brevemente a conexão TCP, vamos verificar a estrutura do segmento TCP, que consiste em campos de cabeçalho e um campo de dados. O campo de dados contém uma quantidade de dados de

aplicação. Como citado antes, o MSS limita o tamanho máximo do campo de dados de um segmento. Quando o TCP envia um arquivo grande, tal como uma imagem de uma página Web, ele comumente fragmenta o segmento em pedaços de tamanho MSS (exceto o último pedaço, que muitas vezes é menor do que o MSS). Aplicações interativas, contudo, muitas vezes transmitem quantidades de dados que são menores do que o MSS. Por exemplo, com aplicações de login remoto como Telnet, o campo de dados do segmento TCP é, muitas vezes, de apenas 1 byte. Como o cabeçalho TCP tem tipicamente 20 bytes (12 bytes mais do que o cabeçalho UDP), o comprimento dos segmentos enviados por Telnet pode ser de apenas 21 bytes.

A Figura 3.29 mostra a estrutura do segmento TCP. Como acontece com o UDP, o cabeçalho inclui **números de porta de fonte e de destino**, que são usados para multiplexação e demultiplexação de dados de/para aplicações de camadas superiores e, assim como no UDP, inclui um **campo de soma de verificação**. Um cabeçalho de segmento TCP também contém os seguintes campos:

- O campo de **número de seqüência** de 32 bits e o campo de **número de reconhecimento** de 32 bits são usados pelos TCPs remetente e destinatário na implementação de um serviço confiável de transferência de dados, como discutido a seguir.
- O campo de **janela de recepção** de 16 bits é usado para controle de fluxo. Veremos em breve que esse campo é usado para indicar o número de bytes que um destinatário está disposto a aceitar.
- O campo de **comprimento de cabeçalho** de 4 bits especifica o comprimento do cabeçalho TCP em palavras de 32 bits. O cabeçalho TCP pode ter comprimento variável devido ao campo de opções TCP. (O campo de opções TCP normalmente está vazio, de modo que o comprimento do cabeçalho TCP típico é 20 bytes.)
- O campo de **opções**, opcional e de comprimento variável, é usado quando um remetente e um destinatário negociam o MSS, ou como um fator de aumento de escala da janela para utilização em redes de alta velocidade. Uma opção de marca de tempo é também definida. Consulte o RFC 854 e o RFC 1323 para detalhes adicionais.
- O campo de **flag** contém 6 bits. O bit **ACK** é usado para indicar se o valor carregado no campo de reconhecimento é válido, isto é, se o segmento contém um reconhecimento para um segmento que foi recebido com sucesso. Os bits **RST**, **SYN** e **FIN** são usados para estabelecer e encerrar a conexão, como

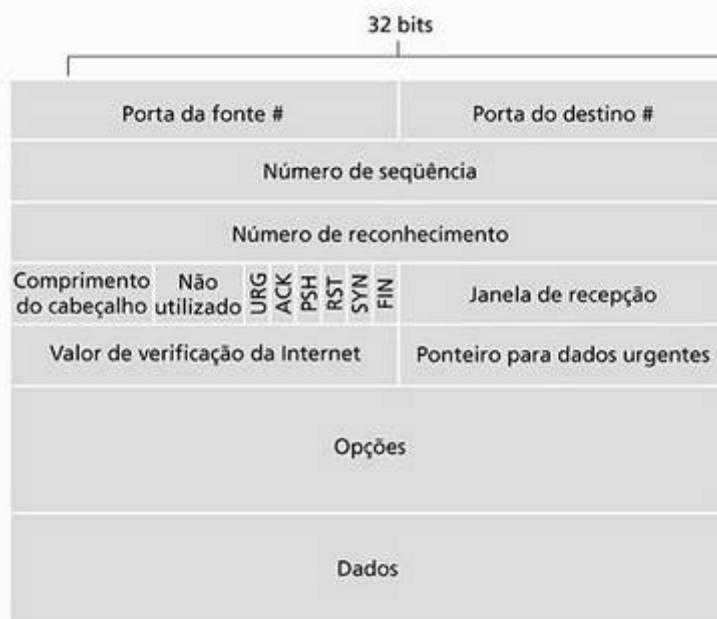


Figura 3.29 Estrutura do segmento TCP

discutiremos ao final desta seção. Marcar o bit **PSH** indica que o destinatário deve passar os dados para a camada superior imediatamente. Por fim, o bit **URG** é usado para mostrar que há dados nesse segmento que a entidade da camada superior do lado remetente marcou como 'urgentes'. A localização do último byte desses dados urgentes é indicada pelo **campo de ponteiro de urgência** de 16 bits. O TCP deve informar à entidade da camada superior do lado destinatário quando existem dados urgentes e passar a ela um ponteiro para o final desses dados. (Na prática, o PSH, o URG e o ponteiro de dados urgentes não são usados. Contudo, mencionamos esses campos para descrever todos.)

Números de sequência e números de reconhecimento

Dois dos mais importantes campos do cabeçalho do segmento TCP são o campo de número de sequência e o campo de número de reconhecimento. Esses campos são parte fundamental do serviço de transferência confiável de dados do TCP. Mas, antes de discutirmos como esses campos são utilizados, vamos explicar exatamente o que o TCP coloca nesses campos.

O TCP vê os dados como uma cadeia de bytes não estruturada, mas ordenada. O uso que o TCP faz dos números de sequência reflete essa visão, pois esses números são aplicados sobre a cadeia de bytes transmitidos, e *não* sobre a série de segmentos transmitidos. O **número de sequência para um segmento** é o número do primeiro byte do segmento. Vamos ver um exemplo. Suponha que um processo no hospedeiro A queira enviar uma cadeia de dados para um processo no hospedeiro B por uma conexão TCP. O TCP do hospedeiro A vai implicitamente numerar cada byte da cadeia de dados. Suponha que a cadeia de dados consista em um arquivo composto de 500.000 bytes, que o MSS seja de 1.000 bytes e que seja atribuído o número 0 ao primeiro byte da cadeia de dados. Como mostra a Figura 3.30, o TCP constrói 500 segmentos a partir da cadeia de dados. O primeiro segmento recebe o número de sequência 0; o segundo, o número de sequência 1.000; o terceiro, o número de sequência 2.000, e assim por diante. Cada número de sequência é inserido no campo de número de sequência no cabeçalho do segmento TCP apropriado.

Vamos agora considerar os números de reconhecimento. Esses números são um pouco mais complicados do que os números de sequência. Lembre-se de que o TCP é *full-duplex*, portanto o hospedeiro A pode estar recebendo dados do hospedeiro B enquanto envia dados ao hospedeiro B (como parte da mesma conexão TCP). Cada um dos segmentos que chegam do hospedeiro B tem um número de sequência para os dados que estão fluindo de B para A. O **número de reconhecimento que o hospedeiro A atribui a seu segmento** é o número de sequência do próximo byte que ele estiver aguardando do hospedeiro B. É bom examinarmos alguns exemplos para entendermos o que está acontecendo aqui. Suponha que o hospedeiro A tenha recebido do hospedeiro B todos os bytes numerados de 0 a 535 e também que esteja prestes a enviar um segmento ao hospedeiro B. O hospedeiro A está esperando pelo byte 536 e por todos os bytes subsequentes da corrente de dados do hospedeiro B. Assim, ele coloca o número 536 no campo de número de reconhecimento do segmento que envia para o hospedeiro B.

Como outro exemplo, suponha que o hospedeiro A tenha recebido um segmento do hospedeiro B contendo os bytes de 0 a 535 e outro segmento contendo os bytes de 900 a 1.000. Por alguma razão, o hospedeiro A ainda não recebeu os bytes de 536 a 899. Nesse exemplo, ele ainda está esperando pelo byte 536 (e os superiores) para poder recriar a cadeia de dados de B. Assim, o segmento seguinte que A envia a B conterá 536 no campo de

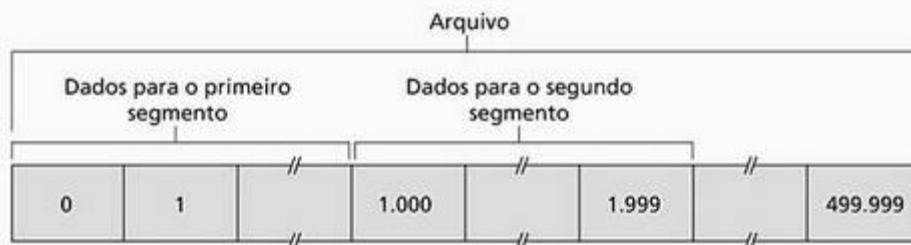


Figura 3.30 Dividindo os dados do arquivo em segmentos TCP

número de reconhecimento. Como o TCP somente reconhece bytes até o primeiro byte que estiver faltando na cadeia, dizemos que o TCP provê **reconhecimentos cumulativos**.

Esse último exemplo também revela uma questão importante, mas sutil. O hospedeiro A recebeu o terceiro segmento (bytes de 900 a 1.000) antes de receber o segundo (bytes de 536 a 899). Portanto, o terceiro segmento chegou fora de ordem. E o que um hospedeiro faz quando recebe segmentos fora de ordem em uma conexão TCP? Esta é a questão. O interessante é que os RFCs do TCP não impõem nenhuma regra para isso e deixam a decisão para quem estiver programando a implementação TCP. Há basicamente duas opções: (1) o destinatário descarta imediatamente os segmentos fora de ordem (o que, como discutimos anteriormente, pode simplificar o projeto do destinatário) ou (2) o destinatário conserva os bytes fora de ordem e espera pelos bytes faltantes para preencher as lacunas. Claro que a segunda alternativa é mais eficiente em termos de largura de banda de rede e é a abordagem adotada na prática.

Na Figura 3.30, admitimos que o número de sequência inicial era 0. Na verdade, ambos os lados de uma conexão TCP escolhem aleatoriamente um número de sequência inicial. Isso é feito para minimizar a possibilidade de um segmento de uma conexão já encerrada entre dois hospedeiros e ainda presente na rede ser tomado por um segmento válido em uma conexão posterior entre esses dois mesmos hospedeiros (que também podem estar usando os mesmos números de porta da conexão antiga) [Sunshine, 1978].

Telnet: um estudo de caso para números de sequência e números de reconhecimento

O Telnet, definido no RFC 854, é um protocolo popular de camada de aplicação utilizado para fazer login remoto. Ele roda sobre TCP e é projetado para trabalhar entre qualquer par de hospedeiros. Diferentemente das aplicações de transferência de dados em grandes blocos, que foram discutidas no Capítulo 2, o Telnet é uma aplicação interativa. Discutiremos, agora, um exemplo de Telnet, pois ele ilustra muito bem números de sequência e de reconhecimento do TCP. Observamos que muitos usuários agora preferem usar o protocolo 'ssh' em vez do Telnet, visto que dados enviados por uma conexão Telnet (incluindo senhas!) não são criptografados, o que torna essa aplicação vulnerável a ataques de bisbilhoteiros (como discutiremos na Seção 8.7).

Suponha que o hospedeiro A inicie uma sessão Telnet com o hospedeiro B. Como o hospedeiro A inicia a sessão, ele é rotulado de cliente, enquanto o hospedeiro B é rotulado de servidor. Cada caractere digitado pelo usuário (no cliente) será enviado ao hospedeiro remoto; este devolverá uma cópia ('eco') de cada caractere, que será apresentada na tela Telnet do usuário. Esse eco é usado para garantir que os caracteres vistos pelo usuário do Telnet já foram recebidos e processados no local remoto. Assim, cada caractere atravessa a rede duas vezes entre o momento em que o usuário aperta o teclado e o momento em que o caractere é apresentado em seu monitor.

Suponha agora que o usuário digite a letra 'C' e saia para tomar um café. Vamos examinar os segmentos TCP que são enviados entre o cliente e o servidor. Como mostra a Figura 3.31, admitamos que os números de sequência iniciais sejam 42 e 79 para cliente e servidor, respectivamente. Lembre-se de que o número de sequência de um segmento será o número de sequência do primeiro byte do seu campo de dados. Assim, o primeiro segmento enviado do cliente terá número de sequência 42; o primeiro segmento enviado do servidor terá número de sequência 79. Note que o número de reconhecimento será o número de sequência do próximo byte de dados que o hospedeiro estará aguardando. Após o estabelecimento da conexão TCP, mas antes de quaisquer dados serem enviados, o cliente ficará esperando pelo byte 79 e o servidor, pelo byte 42.

Como mostra a Figura 3.31, são enviados três segmentos. O primeiro é enviado do cliente ao servidor, contendo, em seu campo de dados, um byte com a representação ASCII para a letra 'C'. O primeiro segmento também tem 42 em seu campo de número de sequência, como acabamos de descrever. E mais, como o cliente ainda não recebeu nenhum dado do servidor, esse segmento terá o número 79 em seu campo de número de reconhecimento.

O segundo segmento é enviado do servidor ao cliente. Esse segmento tem dupla finalidade. A primeira finalidade é fornecer um reconhecimento para os dados que o servidor recebeu. Ao colocar 43 no campo de reconhecimento, o servidor está dizendo ao cliente que recebeu com sucesso tudo até o byte 42 e agora está aguardando os bytes de 43 em diante. A segunda finalidade desse segmento é ecoar a letra 'C'. Assim, o segundo segmento tem a representação ASCII de 'C' em seu campo de dados. Ele tem o número de sequência 79, que é o

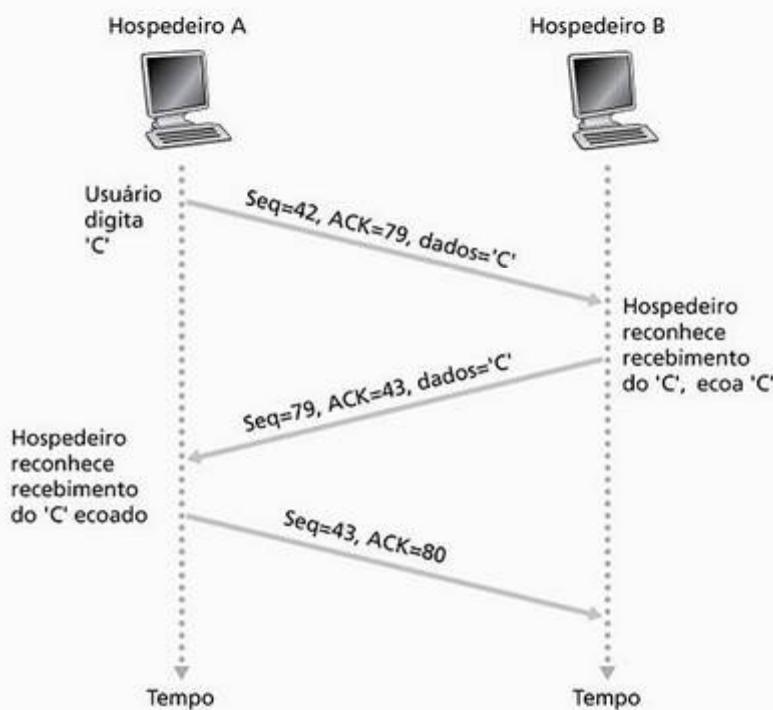


Figura 3.31 Números de sequência e de reconhecimento para uma aplicação Telnet simples sobre TCP

número de sequência inicial do fluxo de dados de servidor para cliente dessa conexão TCP, pois este é o primeiríssimo byte de dados que o servidor está enviando. Note que o reconhecimento para dados do cliente para o servidor é levado em um segmento que carrega dados do servidor para o cliente. Dizemos que este reconhecimento **pegou uma carona** no segmento de dados do servidor para o cliente. Tecnicamente, essa ‘carona’ recebe o nome de **piggyback**.

O terceiro segmento é enviado do cliente ao servidor. Seu único propósito é reconhecer os dados que recebeu do servidor. (Lembre-se de que o segundo segmento continha dados — a letra ‘C’ — do servidor para o cliente.) Esse segmento tem um campo de dados vazio (isto é, o reconhecimento não está pegando carona com nenhum dado do cliente para o servidor), tem o número 80 no campo do número de reconhecimento porque o cliente recebeu a cadeia de dados até o byte com número de sequência 79 e agora está aguardando os bytes de 80 em diante. É possível que você esteja pensando que é estranho que esse segmento também tenha um número de sequência, já que não contém dados. Mas, como o TCP tem um campo de número de sequência, o segmento precisa apresentar algum número para preenchê-lo.

3.5.3 Estimativa do tempo de viagem de ida e volta e de esgotamento de temporização

O TCP, assim como o nosso protocolo rdt da Seção 3.4, utiliza um mecanismo de controle de temporização/retransmissão para recuperar segmentos perdidos. Embora conceitualmente simples, surgem muitas questões sutis quando implementamos um mecanismo de controle de temporização/retransmissão em um protocolo real como o TCP. Talvez a pergunta mais óbvia seja a duração dos intervalos de controle. Evidentemente, esse intervalo deve ser maior do que o tempo de viagem de ida e volta da conexão (RTT), isto é, o tempo decorrido entre o envio de um segmento e seu reconhecimento. Se não fosse assim, seriam enviadas retransmissões desnecessárias. Mas quão maior deve ser o intervalo e, antes de mais nada, como o RTT deve ser estimado? Deve-se associar um temporizador a cada segmento não reconhecido? São tantas perguntas! Nesta seção, nossa discussão se baseia no trabalho de [Jacobson, 1988] sobre TCP e nas recomendações da IETF vigentes para o gerenciamento de temporizadores TCP [RFC 2988].

Estimativa do tempo de viagem de ida e volta

Vamos iniciar nosso estudo do gerenciamento do temporizador TCP considerando como esse protocolo estima o tempo de viagem de ida e volta entre remetente e destinatário, o que apresentaremos a seguir. O RTT para um segmento, denominado SampleRTT no exemplo, é a quantidade de tempo transcorrido entre o momento em que o segmento é enviado (isto é, passado ao IP) e o momento em que é recebido um reconhecimento para o segmento. Em vez de medir um SampleRTT para cada segmento transmitido, a maioria das implementações de TCP executa apenas uma medição de SampleRTT por vez. Isto é, em qualquer instante, o SampleRTT estará sendo estimado para apenas um dos segmentos transmitidos mas ainda não reconhecidos, o que resulta em um novo valor de SampleRTT para aproximadamente cada RTT. E mais, o TCP nunca computa um SampleRTT para um segmento que foi retransmitido; apenas mede-o para segmentos que foram transmitidos uma vez. (Um dos problemas ao final do capítulo perguntará por quê.)

Obviamente, os valores de SampleRTT sofrerão variação de segmento para segmento devido a congestionamento nos roteadores e a variações de carga nos sistemas finais. Por causa dessa variação, qualquer dado valor de SampleRTT pode ser atípico. Portanto, para estimar um RTT típico, é natural tomar alguma espécie de média dos valores de SampleRTT. O TCP mantém uma média, denominada EstimatedRTT, dos valores de SampleRTT. Ao obter um novo SampleRTT, o TCP atualiza EstimatedRTT de acordo com a seguinte fórmula:

$$\text{EstimatedRTT} = (1 - \alpha) \cdot \text{EstimatedRTT} + \alpha \cdot \text{SampleRTT}.$$

Essa fórmula está escrita sob a forma de um comando de linguagem de programação — o novo valor de EstimatedRTT é uma combinação ponderada entre o valor anterior de EstimatedRTT e o novo valor para SampleRTT. O valor recomendado de α é $\alpha = 0,125$ (isto é, $1/8$) [RFC 2988], caso em que essa fórmula se torna:

$$\text{EstimatedRTT} = 0,875 \cdot \text{EstimatedRTT} + 0,125 \cdot \text{SampleRTT}.$$

Note que EstimatedRTT é uma média ponderada dos valores de SampleRTT. Como veremos em um exercício ao final deste capítulo, essa média ponderada atribui um peso maior às amostras recentes do que às amostras antigas. Isso é natural, pois as amostras mais recentes refletem melhor o estado atual de congestionamento da rede. Em estatística, esse tipo de média é denominada **média móvel exponencial ponderada**. A palavra 'exponencial' aparece na MMEP porque o peso atribuído a um dado SampleRTT diminui exponencialmente à medida que as atualizações são realizadas. Os exercícios pedirão que você derive o termo exponencial em EstimatedRTT.

A Figura 3.32 mostra os valores de SampleRTT e EstimatedRTT para um valor de $\alpha = 1/8$, para uma conexão TCP entre gaia.cs.umass.edu (em Amherst, Massachusetts) e fantasia.eurecom.fr (no sul da França). Fica claro que as variações em SampleRTT são atenuadas no cálculo de EstimatedRTT.

Além de ter uma estimativa do RTT, também é valioso ter uma medida de sua variabilidade. O [RFC 2988] define a variação do RTT, DevRTT, como uma estimativa do desvio típico entre SampleRTT e EstimatedRTT:

$$\text{DevRTT} = (1 - \beta) \cdot \text{DevRTT} + \beta \cdot |\text{SampleRTT} - \text{EstimatedRTT}|$$

Note que DevRTT é uma MMEP da diferença entre SampleRTT e EstimatedRTT. Se os valores de SampleRTT apresentarem pouca variação, então DevRTT será pequeno; por outro lado, se houver muita variação, DevRTT será grande. O valor recomendado para β é 0,25.

Estabelecimento e gerenciamento da temporização de retransmissão

Dados valores de EstimatedRTT e DevRTT, que valor deve ser utilizado para a temporização de retransmissão do TCP? É óbvio que o intervalo deve ser maior ou igual a EstimatedRTT, caso contrário seriam enviadas retransmissões desnecessárias. Mas a temporização de retransmissão não deve ser muito maior do que EstimatedRTT, senão, quando um segmento fosse perdido, o TCP não o retransmitiria rapidamente, o que resultaria em grandes atrasos de transferência de dados. Portanto, é desejável que o valor estabelecido para a temporização seja igual a EstimatedRTT mais uma certa margem, que deverá ser grande quando houver muita

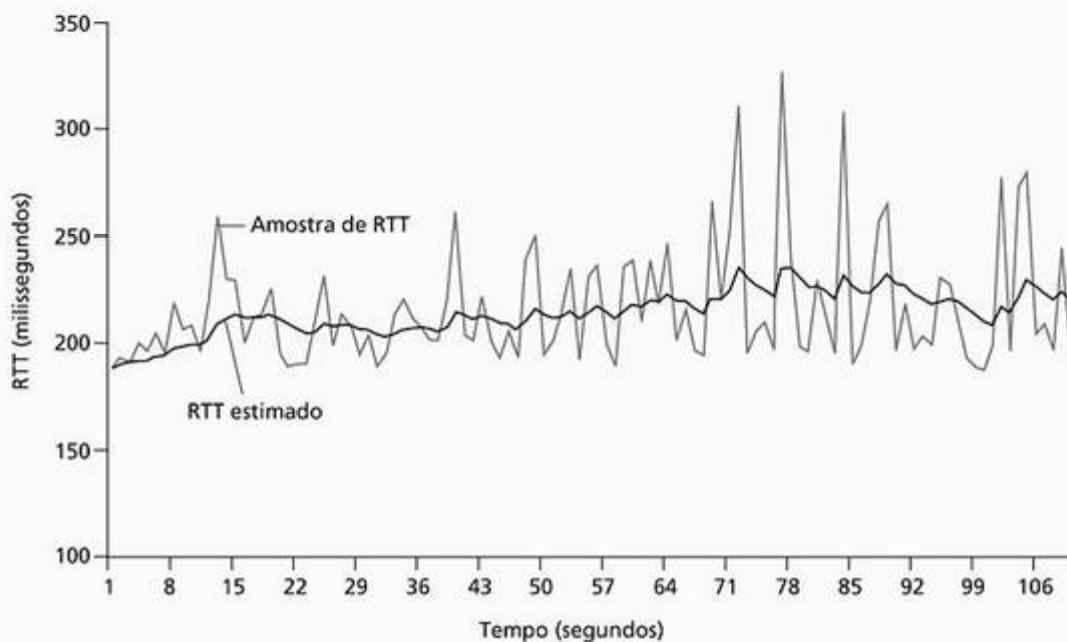


Figura 3.32 Amostras de RTTs e RTTs estimados

variação nos valores de SampleRTT e pequena quando houver pouca variação. Assim, o valor de DevRTT deve entrar em jogo. Todas essas considerações são levadas em conta no método do TCP para determinar a temporização de retransmissão:

$$\text{TimeoutInterval} = \text{EstimatedRTT} + 4 \cdot \text{DevRTT}$$

Princípios na prática

O TCP fornece transferência confiável de dados usando reconhecimentos positivos e temporizadores, de modo muito parecido com o que estudamos na Seção 3.4. O protocolo reconhece dados que foram recebidos corretamente e retransmite segmentos quando entende que eles ou seus reconhecimentos correspondentes foram perdidos ou corrompidos. Certas versões do TCP também têm um mecanismo NAK implícito — com o mecanismo de retransmissão rápida do TCP. O recebimento de três ACKs duplicados para um dado segmento serve como um NAK implícito para o segmento seguinte, acionando a retransmissão daquele segmento antes que o tempo se esgote. O TCP usa números de sequência para permitir que o destinatário identifique segmentos perdidos ou duplicados. Exatamente como no caso de nosso protocolo de transferência confiável de dados rdt3.0, o TCP em si não pode determinar com certeza se um segmento, ou seu ACK, está perdido, corrompido ou excessivamente atrasado. No remetente, a resposta do TCP será a mesma: retransmitir o segmento em questão. O TCP também utiliza paralelismo, permitindo que o remetente tenha, a qualquer tempo, múltiplos segmentos transmitidos mas ainda não reconhecidos. Vimos anteriormente que o paralelismo pode melhorar muito a vazão de uma sessão quando a razão entre o tempo de transmissão do segmento e o atraso de viagem de ida e volta é pequena. O número específico de segmentos não reconhecidos que um remetente pode ter é determinado pelos mecanismos de controle de fluxo e controle de congestionamento do TCP. O controle de fluxo do TCP é discutido no final desta seção; o controle de congestionamento do TCP é discutido na Seção 3.7. Por enquanto, devemos apenas ficar cientes de que o TCP remetente usa paralelismo.

3.5.4 Transferência confiável de dados

Lembre-se de que o serviço da camada de rede da Internet (serviço IP) não é confiável. O IP não garante a entrega de datagramas na ordem correta nem a integridade de seus dados nos datagramas. Com o serviço IP, os datagramas podem transbordar dos buffers dos roteadores e jamais alcançar seu destino; podem também chegar fora de ordem. Além disso, os bits dos datagramas podem ser corrompidos (passar de 0 para 1 e vice-versa). Como os segmentos da camada de transporte são carregados pela rede por datagramas IPs, também eles podem sofrer esses mesmos problemas.

O TCP cria um **serviço de transferência confiável de dados** sobre o serviço de melhor esforço do IP. Esse serviço de transferência garante que a cadeia de dados que um processo lê a partir de seu buffer de recebimento TCP não está corrompida, não tem lacunas, não tem duplicações e está em sequência, isto é, a cadeia de bytes é exatamente a mesma cadeia de bytes enviada pelo sistema final que está do outro lado da conexão. O modo como o TCP provê transferência confiável de dados envolve muitos dos princípios estudados na Seção 3.4.

Quando desenvolvemos anteriormente técnicas de transferência confiável de dados, era conceitualmente mais fácil admitir que existia um temporizador individual associado com cada segmento transmitido mas ainda não reconhecido. Embora, em teoria, isso seja ótimo, o gerenciamento de temporizadores pode exigir considerável sobrecarga. Assim, os procedimentos recomendados no [RFC 2988] para gerenciamento de temporizadores TCP utilizam apenas um único temporizador de retransmissão, mesmo que haja vários segmentos transmitidos ainda não reconhecidos. O protocolo TCP apresentado nesta seção segue essa recomendação.

Discutiremos como o TCP provê transferência confiável de dados em duas etapas incrementais. Em primeiro lugar, apresentamos uma descrição muito simplificada de um remetente TCP que utiliza somente controle de temporizadores para se recuperar da perda de segmentos; em seguida, apresentaremos uma descrição mais complexa que utiliza reconhecimentos duplicados além de temporizadores de retransmissão. Na discussão que se segue, admitimos que os dados estão sendo enviados em uma direção somente, do hospedeiro A ao hospedeiro B, e que o hospedeiro A está enviando um arquivo grande.

A Figura 3.33 apresenta uma descrição muito simplificada de um remetente TCP.

Vemos que há três eventos importantes relacionados com a transmissão e a retransmissão de dados no TCP remetente: dados recebidos da aplicação acima; esgotamento do temporizador e recebimento de ACK. Quando ocorre o primeiro evento importante, o TCP recebe dados da camada de aplicação, encapsula-os em um segmento e passa-o ao IP. Note que cada segmento inclui um número de sequência que é o número da corrente de bytes do primeiro byte de dados no segmento, como descrito na Seção 3.5.2. Note também que, se o temporizador não estiver funcionando naquele instante para algum outro segmento, o TCP aciona o temporizador quando o segmento é passado para o IP. (Fica mais fácil se você imaginar que o temporizador está associado com o mais antigo segmento não reconhecido.) O intervalo de expiração para esse temporizador é o `TimeoutInterval`, calculado a partir de `EstimatedRTT` e `DevRTT`, como descrito na Seção 3.5.3.

O segundo evento importante é o esgotamento do temporizador. O TCP responde a esse evento retransmitindo o segmento que causou o esgotamento da temporização e então reinicia o temporizador.

O terceiro evento importante que deve ser manipulado pelo TCP remetente é a chegada de um segmento de reconhecimento (ACK) do destinatário (mais especificamente, um segmento contendo um valor de campo de ACK válido). Quando da ocorrência desse evento, o TCP compara o valor do ACK, y , com sua variável `SendBase`. A variável de estado `SendBase` do TCP é o número de sequência do mais antigo byte não reconhecido. (Assim, $SendBase - 1$ é o número de sequência do último byte que se sabe ter sido recebido pelo destinatário corretamente e na ordem certa.) Como comentamos anteriormente, o TCP usa reconhecimentos cumulativos, de modo que y reconhece o recebimento de todos os bytes antes do byte número y . Se $y > SendBase$, então o ACK está reconhecendo um ou mais bytes não reconhecidos anteriormente. Desse modo, o remetente atualiza sua variável `SendBase` e também reinicia o temporizador se houver quaisquer segmentos ainda não reconhecidos.

Alguns cenários interessantes

Acabamos de descrever uma versão muito simplificada do modo como o TCP provê transferência confiável de dados, mas mesmo essa descrição tão simplificada tem muitas sutilezas. Para ter uma boa percepção de como

/ Suponha que o remetente não seja compelido pelo fluxo de TCP ou controle de congestionamento, que o tamanho dos dados vindos de cima seja menor do que o MSS e que a transferência de dados ocorra apenas em uma direção.*/*

```

NextSeqNum=InitialSeqNumber
SendBase=InitialSeqNumber

loop (forever) {
    switch (event)

        event: data received from the application above
            create TCP segment with sequence number NextSeqNum
            if (timer currently not running)
                start timer
            pass segment to IP
            NextSeqNum=NextSeqNum+length (data)
            break;

        event: timer timeout
            retransmit not-yet-acknowledged segment with
                smallest sequence number
            start timer
            break;

        event: ACK received, with ACK field value of y
            if (y > SendBase) {
                SendBase=y
                If (there are currently any not-yet-acknowledged segments)
                    start timer
            }
}

```

Figura 3.33 Remetente TCP simplificado

esse protocolo funciona, vamos agora examinar alguns cenários simples. A Figura 3.34 ilustra o primeiro cenário em que um hospedeiro A envia um segmento ao hospedeiro B. Suponha que esse segmento tenha número de sequência 92 e contenha 8 bytes de dados. Após enviá-lo, o hospedeiro A espera por um segmento de B com número de reconhecimento 100. Embora o segmento de A seja recebido em B, o reconhecimento de B para A se perde. Nesse caso, ocorre o evento de expiração do temporizador e o hospedeiro A retransmite o mesmo segmento. É claro que, quando recebe a retransmissão, o hospedeiro B observa, pelo número de sequência, que o segmento contém dados que já foram recebidos. Assim, o TCP no hospedeiro B descarta os bytes do segmento retransmitido.

Em um segundo cenário, mostrado na Figura 3.35, o hospedeiro A envia dois segmentos seguidos. O primeiro segmento tem número de sequência 92 e 8 bytes de dados. O segundo segmento tem número de sequência 100 e 20 bytes de dados. Suponha que ambos cheguem intactos em B e que B envie dois reconhecimentos separados para cada um desses segmentos. O primeiro desses reconhecimentos tem número de reconhecimento 100; o segundo, número de reconhecimento 120. Suponha agora que nenhum dos reconhecimentos chegue ao hospedeiro A antes do esgotamento do temporizador. Quando ocorre o evento de expiração do temporizador,

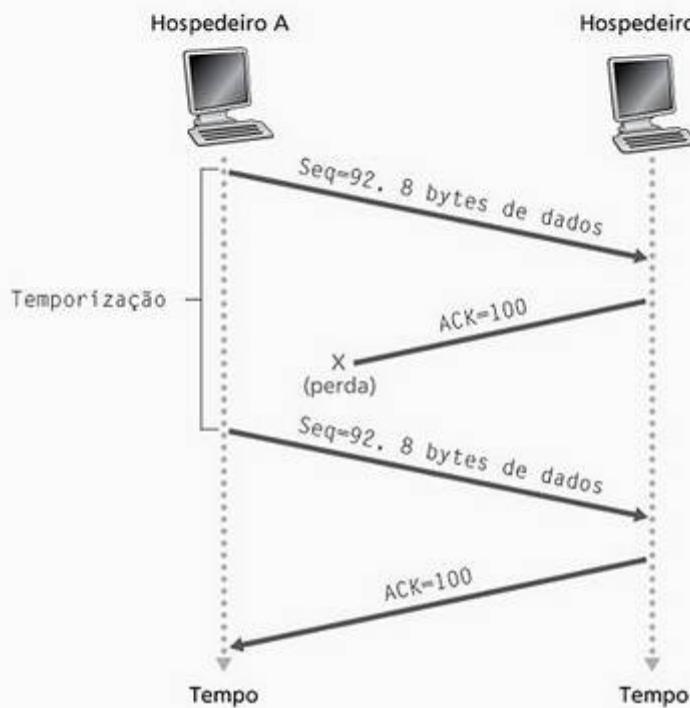


Figura 3.34 Retransmissão devido a um reconhecimento perdido

o hospedeiro A reenvia o primeiro segmento com número de sequência 92 e reinicia o temporizador. Contanto que o ACK do segundo segmento chegue antes que o temporizador expire novamente, o segundo segmento não será retransmitido.

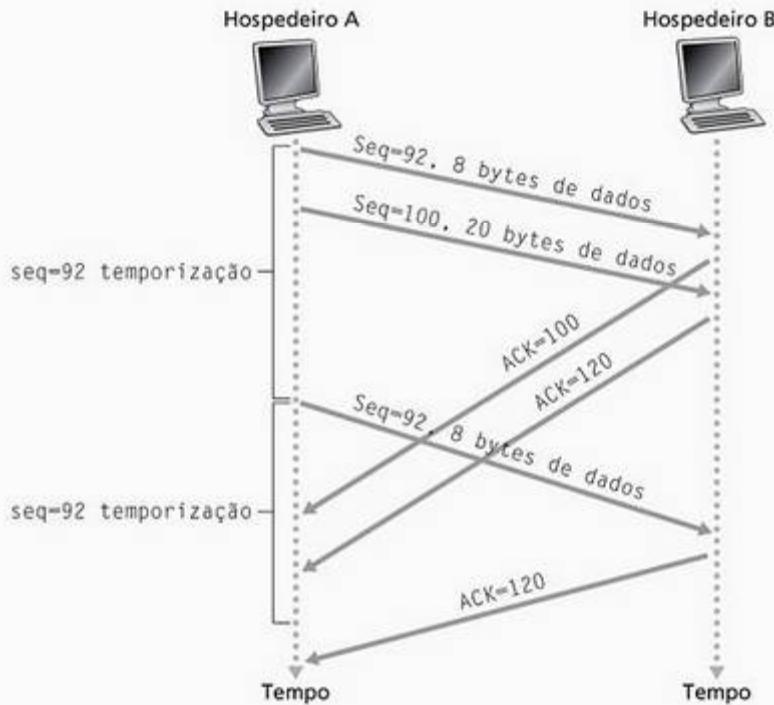


Figura 3.35 Segmento 100 não retransmitido

Em um terceiro e último cenário, suponha que o hospedeiro A envie dois segmentos, exatamente como no segundo exemplo. O reconhecimento do primeiro segmento é perdido na rede, mas, um pouco antes do evento de expiração, o hospedeiro A recebe um reconhecimento com número 120. O hospedeiro A, portanto, sabe que B recebeu tudo até o byte 119; portanto, ele não reenvia nenhum dos dois segmentos. Esse cenário está ilustrado na Figura 3.36.

Duplicação do tempo de expiração

Discutiremos agora algumas modificações empregadas por grande parte das implementações do TCP. A primeira refere-se à duração do tempo de expiração após a expiração de um temporizador. Nessa modificação, sempre que ocorre o evento de expiração do temporizador, o TCP retransmite o segmento ainda não reconhecido que tenha o menor número de sequência, como descrevemos anteriormente. Mas, a cada retransmissão, o TCP ajusta o próximo tempo de expiração para o dobro do valor anterior em vez de derivá-lo dos últimos EstimatedRTT e DevRTT (como descrito na Seção 3.5.3). Por exemplo, suponha que o TimeoutInterval associado com o mais antigo segmento ainda não reconhecido seja 0,75 segundo quando o temporizador expirar pela primeira vez. O TCP então retransmite esse segmento e ajusta o novo tempo de expiração para 1,5 segundo. Se o temporizador expirar novamente 1,5 segundo mais tarde, o TCP retransmitirá novamente esse segmento, agora ajustando o tempo de expiração para 3,0 segundos. Assim, o tempo aumenta exponencialmente após cada retransmissão. Todavia, sempre que o temporizador é iniciado após qualquer um dos outros dois eventos (isto é, dados recebidos da aplicação acima e ACK recebido), o TimeoutInterval será derivado dos valores mais recentes de EstimatedRTT e DevRTT.

Essa modificação provê uma forma limitada de controle de congestionamento. (Maneiras mais abrangentes de controle de congestionamento no TCP serão estudadas na Seção 3.7). A causa mais provável da expiração do temporizador é o congestionamento na rede, isto é, um número muito grande de pacotes chegando a uma (ou mais) fila de roteadores no caminho entre a fonte e o destino, o que provoca descarte de pacotes e/ou longos atrasos de fila. Se as fontes continuarem a retransmitir pacotes persistentemente durante um congestionamento, ele pode piorar. Para que isso não aconteça, o TCP age mais educadamente: cada remetente retransmite após intervalos cada vez mais longos. Veremos que uma ideia semelhante a essa é utilizada pela Ethernet, quando estudarmos CSMA/CD no Capítulo 5.

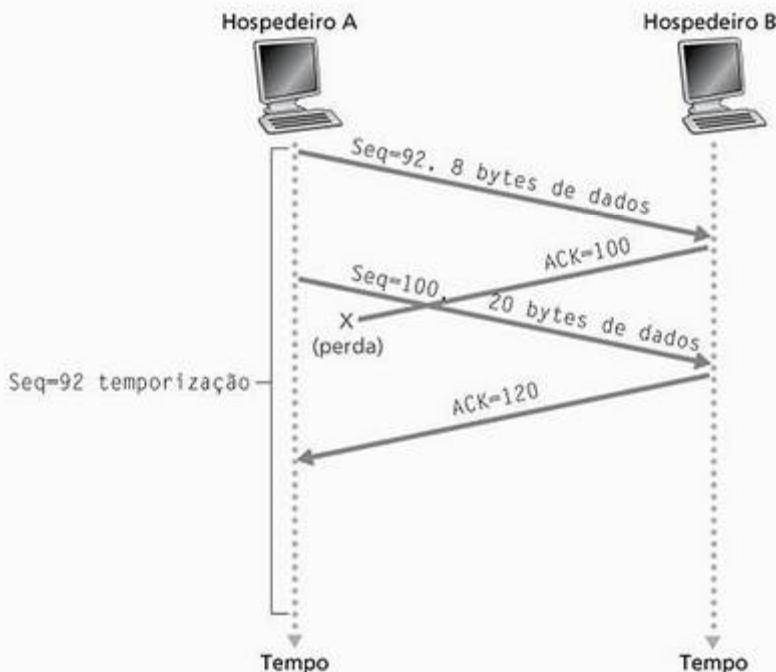


Figura 3.36 Um reconhecimento cumulativo evita retransmissão do primeiro segmento

Retransmissão rápida

Um dos problemas de retransmissões acionadas por expiração de temporizador é que o período de expiração pode ser relativamente longo. Quando um segmento é perdido, esse longo período força o remetente a atrasar o reenvio do pacote perdido, consequentemente aumentando o atraso fim a fim. Felizmente, o remetente pode com frequência detectar perda de pacote bem antes de ocorrer o evento de expiração, observando os denominados ACKs duplicados. Um **ACK duplicado** é um ACK que reconhece novamente um segmento para o qual o remetente já recebeu um reconhecimento anterior. Para entender a resposta do remetente a um ACK duplicado, devemos examinar por que o destinatário envia um ACK duplicado em primeiro lugar. A Tabela 3.2 resume a política de geração de ACKs do TCP destinatário [RFC 1122, RFC 2581].

Quando um TCP destinatário recebe um segmento com um número de sequência que é maior do que o número de sequência subsequente, esperado e na ordem, ele detecta uma lacuna na corrente de dados — isto é, a falta de um segmento. Essa lacuna poderia ser o resultado de segmentos perdidos ou reordenados dentro da rede. Uma vez que o TCP não usa reconhecimentos negativos, o destinatário não pode enviar um reconhecimento negativo explícito de volta ao remetente. Em vez disso, ele simplesmente reconhece novamente (isto é, gera um ACK duplicado) o último byte de dados que recebeu na ordem. (Note que a Tabela 3.2 admite o caso em que o destinatário não descarta segmentos fora de ordem.)

Como um remetente quase sempre envia um grande número de segmentos, um atrás do outro, se um segmento for perdido, provavelmente existirão muitos ACKs duplicados, também um após o outro. Se o TCP remetente receber três ACKs duplicados para os mesmos dados, ele tomará isso como indicação de que o segmento que se seguiu ao segmento reconhecido três vezes foi perdido. (Nos exercícios de fixação consideraremos por que o remetente espera três ACKs duplicados e não apenas um.) No caso de receber três ACKs duplicados, o TCP remetente realiza uma **retransmissão rápida** [RFC 2581], retransmitindo o segmento que falta *antes* da expiração do temporizador do segmento. Isso é mostrado na Figura 3.37, em que o segundo segmento é perdido, e então retransmitido antes da expiração do temporizador. Para o TCP com retransmissão rápida, o seguinte trecho de codificação substitui o evento ACK recebido na Figura 3.33:

```

event: ACK received, with ACK field value of y
    if (y > SendBase) {
        SendBase=y
        if (there are currently any not yet
            acknowledged segments)
            start timer
        }
    else /* a duplicate ACK for already ACKed
           segment */
        increment number of duplicate ACKs
        received for y
        if (number of duplicate ACKS received
            for y==3) {
                /* TCP fast retransmit */
                resend segment with sequence number y
            }
    break;

```

Observamos anteriormente que muitas questões sutis vêm à tona quando um mecanismo de controle de temporização/retransmissão é implementado em um protocolo real como o TCP. Os procedimentos acima, cuja evolução é resultado de mais de 15 anos de experiência com temporizadores TCP, devem convencê-lo de que, indubitavelmente, é isso que acontece!

Evento	Ação do TCP Destinatário
Chegada de segmento na ordem com número de sequência esperado. Todos os dados até o número de sequência esperado já reconhecidos.	ACK retardado. Espera de até 500 milissegundos pela chegada de um outro segmento na ordem. Se o segmento seguinte na ordem não chegar nesse intervalo, envia um ACK.
Chegada de segmento na ordem com número de sequência esperado. Um outro segmento na ordem esperando por transmissão de ACK.	Envio imediato de um único ACK cumulativo, reconhecendo ambos os segmentos na ordem.
Chegada de um segmento fora da ordem com número de sequência mais alto do que o esperado. Lacuna detectada.	Envio imediato de um ACK duplicado, indicando número de sequência do byte seguinte esperado (que é a extremidade mais baixa da lacuna).
Chegada de um segmento que preenche, parcial ou completamente, a lacuna nos dados recebidos.	Envio imediato de um ACK, contanto que o segmento comece na extremidade mais baixa da lacuna.

Tabela 3.2 Recomendação para geração de ACKs TCP [RFC 1122, RFC 2581]

Go-Back-N ou repetição seletiva?

Vamos encerrar nosso estudo do mecanismo de recuperação de erros do TCP considerando a seguinte pergunta: o TCP é um protocolo GBN ou SR? Lembre-se de que, no TCP, os reconhecimentos são cumulativos e segmentos corretamente recebidos, mas fora da ordem, não são reconhecidos (ACK) individualmente pelo destinatário. Consequentemente, como mostra a Figura 3.33 (veja também a Figura 3.19), o TCP remetente precisa tão somente lembrar o menor número de sequência de um byte transmitido mas não reconhecido (SendBase) e o número de sequência do byte seguinte a ser enviado (NextSeqNum). Nesse sentido, o TCP se parece muito com um protocolo ao estilo do GBN. Porém, há algumas diferenças surpreendentes entre o TCP e o GBN. Muitas implementações do TCP armazenarão segmentos recebidos corretamente, mas fora da ordem [Stevens, 1994].

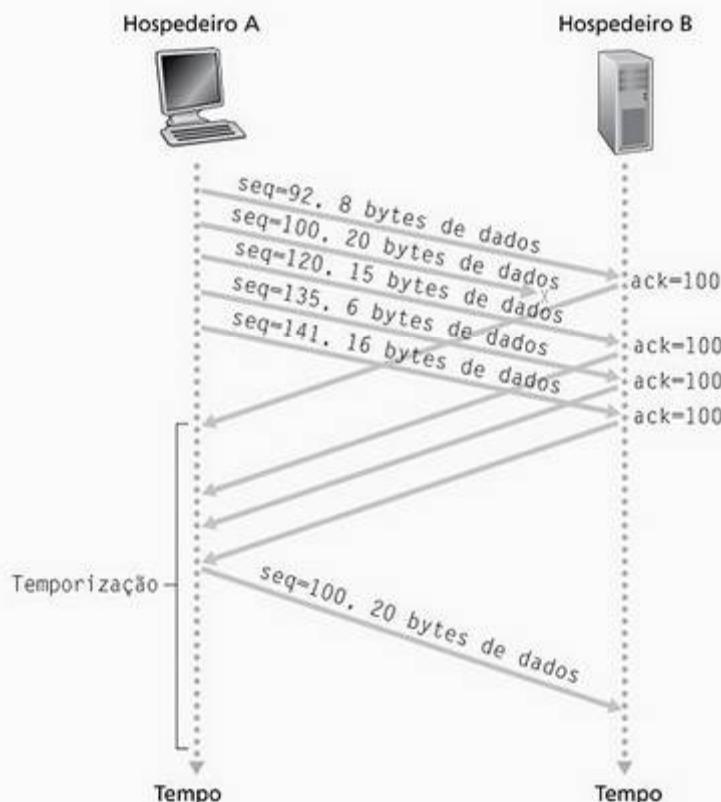


Figura 3.37 Retransmissão rápida: retransmitir o segmento que falta antes da expiração do temporizador do segmento

Considere também o que acontece quando o remetente envia uma sequência de segmentos $1, 2, \dots, N$ e todos os segmentos chegam ao destinatário na ordem e sem erro. Além disso, suponha que o reconhecimento para o pacote $n < N$ se perca, mas que os $N - 1$ reconhecimentos restantes cheguem ao remetente antes do esgotamento de suas respectivas temporizações. Nesse exemplo, o GBN retransmitiria não somente o pacote n , mas também todos os pacotes subsequentes $n + 1, n + 2, \dots, N$. O TCP, por outro lado, retransmitiria no máximo um segmento, a saber, o segmento n . E mais, o TCP nem ao menos retransmitiria o segmento n se o reconhecimento para o segmento $n + 1$ chegasse antes do final da temporização para o segmento n .

Uma modificação proposta para o TCP, denominada **reconhecimento seletivo** [RFC 2018], permite que um destinatário TCP reconheça seletivamente segmentos fora de ordem, em vez de apenas reconhecer cumulativamente o último segmento recebido corretamente e na ordem. Quando combinado com retransmissão seletiva — isto é, saltar a retransmissão de segmentos que já foram reconhecidos seletivamente pelo destinatário —, o TCP se parece muito com nosso protocolo SR genérico. Assim, o mecanismo de recuperação de erros do TCP provavelmente é mais bem caracterizado como um híbrido dos protocolos GBN e SR.

3.5.5 Controle de fluxo

Lembre-se de que os hospedeiros de cada lado de uma conexão TCP reservam um buffer de recepção para a conexão. Quando a conexão TCP recebe bytes que estão corretos e em sequência, ele coloca os dados no buffer de recepção. O processo de aplicação associado lerá os dados a partir desse buffer, mas não necessariamente no momento em que são recebidos. Na verdade, a aplicação receptora pode estar ocupada com alguma outra tarefa e nem ao menos tentar ler os dados até muito depois da chegada deles. Se a aplicação for relativamente lenta na leitura dos dados, o remetente pode muito facilmente saturar o buffer de recepção da conexão por enviar demasiados dados muito rapidamente.

O TCP provê um **serviço de controle de fluxo** às suas aplicações, para eliminar a possibilidade de o remetente saturar o buffer do destinatário. Assim, controle de fluxo é um serviço de compatibilização de velocidades — compatibiliza a taxa à qual o remetente está enviando com a aquela à qual a aplicação receptora está lendo. Como notamos anteriormente, um TCP remetente também pode ser estrangulado devido ao congestionamento dentro da rede IP. Esse modo de controle do remetente é denominado **controle de congestionamento**, um tópico que será examinado detalhadamente nas seções 3.6 e 3.7. Mesmo que as ações executadas pelo controle de fluxo e pelo controle de congestionamento sejam similares (a regulagem do remetente), fica evidente que elas são executadas por razões muito diferentes. Infelizmente, muitos autores usam os termos de modo intercambiável, e o leitor esperto tem de tomar muito cuidado para distinguir os dois casos. Vamos agora discutir como o TCP provê seu serviço de controle de fluxo. Para podermos enxergar o quadro geral, sem nos fixarmos nos detalhes, nesta seção admitiremos que essa implementação do TCP descarta segmentos fora da ordem.

O TCP provê serviço de controle de fluxo fazendo com que o remetente mantenha uma variável denominada **janela de recepção**. Informalmente, a janela de recepção é usada para dar ao remetente uma ideia do espaço de buffer livre disponível no destinatário. Como o TCP é *full-duplex*, o remetente de cada lado da conexão mantém uma janela de recepção distinta. Vamos examinar a janela de recepção no contexto de uma transferência de arquivo. Suponha que o hospedeiro A esteja enviando um arquivo grande ao hospedeiro B por uma conexão TCP. O hospedeiro B aloca um buffer de recepção a essa conexão; denominemos seu tamanho *RcvBuffer*. De tempos em tempos, o processo de aplicação no hospedeiro B faz a leitura do buffer. São definidas as seguintes variáveis:

- LastByteRead = o número do último byte na cadeia de dados lido do buffer pelo processo de aplicação em B.
- LastByteRcvd = o número do último byte na cadeia de dados que chegou da rede e foi colocado no buffer de recepção de B.

Como o TCP não tem permissão para saturar o buffer alocado, devemos ter:

$$\text{LastByteRcvd} - \text{LastByteRead} \leq \text{RcvBuffer}$$

A janela de recepção, denominada *RcvWindow*, é ajustada para a quantidade de espaço disponível no buffer:

$$\text{rwnd} = \text{RcvBuffer} - [\text{LastByteRcvd} - \text{LastByteRead}]$$

Como o espaço disponível muda com o tempo, Rwnd é dinâmica. A variável Rwnd está ilustrada na Figura 3.38.

Como a conexão usa a variável Rwnd para prover o serviço de controle de fluxo? O hospedeiro A diz ao hospedeiro B quanto espaço disponível ele tem no buffer da conexão colocando o valor corrente de Rwnd no campo de janela de recepção de cada segmento que envia a B. Inicialmente, o hospedeiro B estabelece $rwnd = RcvBuffer$. Note que, para conseguir isso, o hospedeiro B deve monitorar diversas variáveis específicas da conexão.

O hospedeiro A, por sua vez, monitora duas variáveis, LastByteSent e LastByteAcked, cujos significados são óbvios. Note que a diferença entre essas duas variáveis, $LastByteSent - LastByteAcked$, é a quantidade de dados não reconhecidos que A enviou para a conexão. Mantendo a quantidade de dados não reconhecidos menor do que o valor de Rwnd, o hospedeiro A tem certeza de que não está fazendo transbordar o buffer de recepção no hospedeiro B. Assim, o hospedeiro A tem de certificar-se, durante toda a duração da conexão, de que:

$$LastByteSent - LastByteAcked \leq rwnd$$

Há um pequeno problema técnico com esse esquema. Para percebê-lo, suponha que o buffer de recepção do hospedeiro B fique tão cheio que $Rwnd = 0$. Após anunciar ao hospedeiro A que $Rwnd = 0$, imagine que B não tenha *nada* para enviar ao hospedeiro A. Agora considere o que acontece. Enquanto o processo de aplicação em B esvazia o buffer, o TCP não envia novos segmentos com novos valores Rwnd para o hospedeiro A. Na verdade, o TCP enviará um segmento ao hospedeiro A somente se tiver dados ou um reconhecimento para enviar. Por conseguinte, o hospedeiro A nunca será informado de que foi aberto algum espaço no buffer de recepção do hospedeiro B: ele ficará bloqueado e não poderá transmitir mais dados! Para resolver esse problema, a especificação do TCP requer que o hospedeiro A continue a enviar segmentos com um byte de dados quando a janela de recepção de B for zero. Esses segmentos serão reconhecidos pelo receptor. Finalmente o buffer começará a envasiar, e os reconhecimentos conterão um valor diferente de zero em rwnd.

O Companion Website do livro (http://www.aw.com/kurose_br) fornece um applet interativo em Java que ilustra a operação da janela de recepção do TCP.

Agora que descrevemos o serviço de controle de fluxo do TCP, mencionaremos brevemente que o UDP não provê controle de fluxo. Para entender a questão, considere o envio de uma série de segmentos UDP de um processo no hospedeiro A para um processo no hospedeiro B. Para uma implementação UDP típica, o UDP anexará os segmentos a um buffer de tamanho finito que ‘precede’ o socket correspondente (isto é, o socket para o processo). O processo lê um segmento inteiro do buffer por vez. Se o processo não ler os segmentos com rapidez suficiente, o buffer transbordará e os segmentos serão descartados.

3.5.6 Gerenciamento da conexão TCP

Nesta subseção, examinamos mais de perto como uma conexão TCP é estabelecida e encerrada. Embora esse tópico talvez não pareça particularmente interessante, é importante, porque o estabelecimento da conexão TCP tem um peso significativo nos atrasos percebidos (por exemplo, ao navegar pela Web). Além disso, muitos dos ataques mais comuns a redes — entre eles o incrivelmente popular ataque de inundação SYN — exploram



Figura 3.38 A janela de recepção (Rwnd) e o buffer de recepção (RcvBuffer)

vulnerabilidades no gerenciamento da conexão TCP. Em primeiro lugar, vamos ver como essa conexão é estabelecida. Suponha que um processo que roda em um hospedeiro (cliente) queira iniciar uma conexão com outro processo em outro hospedeiro (servidor). O processo de aplicação cliente primeiramente informa ao TCP cliente que quer estabelecer uma conexão com um processo no servidor. O TCP no cliente então estabelece uma conexão TCP com o TCP no servidor da seguinte maneira:

Etapa 1. O lado cliente do TCP primeiramente envia um segmento TCP especial ao lado servidor do TCP. Esse segmento especial não contém nenhum dado de camada de aplicação, mas um dos bits de flag no seu cabeçalho (veja a Figura 3.29), o bit SYN, é ajustado para 1. Por essa razão, esse segmento é denominado um segmento SYN. Além disso, o cliente escolhe aleatoriamente um número de sequência inicial (`client_isn`) e coloca esse número no campo de número de sequência do segmento TCP SYN inicial. Esse segmento é encapsulado em um datagrama IP e enviado ao servidor. A aleatoriedade adequada da escolha de `client_isn` de modo a evitar certos ataques à segurança tem despertado considerável interesse [CERT 2001-09].

Etapa 2. Assim que o datagrama IP contendo o segmento TCP SYN chega ao hospedeiro servidor (admitindo-se que ele realmente chegue!), o servidor extrai o segmento TCP SYN do datagrama, aloca buffers e variáveis TCP à conexão e envia um segmento de aceitação de conexão ao TCP cliente. (Veremos, no Capítulo 8, que a alocação desses buffers e variáveis, antes da conclusão da terceira etapa da apresentação de três vias, torna o TCP vulnerável a um ataque de recusa de serviço conhecido como inundação SYN.) Esse segmento de aceitação de conexão também não contém nenhum dado de camada de aplicação. Contudo, contém três informações importantes no cabeçalho do segmento: o bit SYN está com valor 1; o campo de reconhecimento do cabeçalho do segmento TCP está ajustado para `client_isn+1`; e, por fim, o servidor escolhe seu próprio número de sequência inicial (`server_isn`) e coloca esse valor no campo de número de sequência do cabeçalho do segmento TCP. Esse segmento de aceitação de conexão está dizendo, com efeito, “Recebi seu pacote SYN para começar uma conexão com seu número de sequência inicial `client_isn`. Concordo em estabelecer essa conexão. Meu número de sequência inicial é `server_isn`”. O segmento de concessão da conexão às vezes é denominado **segmento SYNACK**.

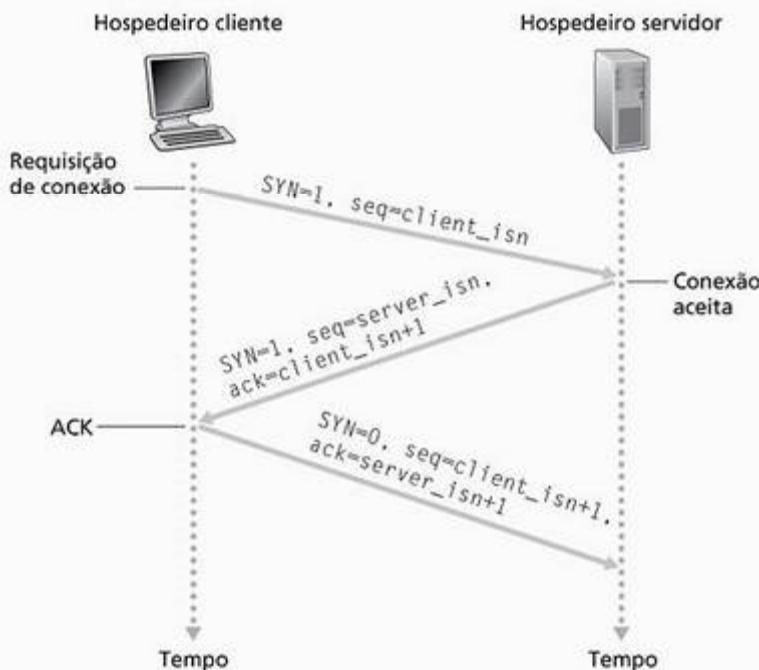


Figura 3.39 Apresentação de três vias do TCP: troca de segmentos

Etapa 3. Ao receber o segmento SYNACK, o cliente também reserva buffers e variáveis para a conexão. O hospedeiro cliente então envia ao servidor mais um segmento. Esse último segmento reconhece o segmento de confirmação da conexão do servidor (o cliente o faz colocando o valor `server_isn+1` no campo de reconhecimento do cabeçalho do segmento TCP). O bit SYN é ajustado para 0, já que a conexão está estabelecida. A terceira etapa da apresentação de três vias pode conduzir os dados cliente/servidor na carga útil do segmento.

Completadas as três etapas, os hospedeiros cliente e servidor podem enviar segmentos contendo dados um ao outro. Em cada um desses futuros segmentos, o bit SYN estará ajustado para 0. Note que, para estabelecer a conexão, três pacotes são enviados entre dois hospedeiros, como ilustra a Figura 3.39. Por essa razão, esse procedimento de estabelecimento de conexão é frequentemente denominado **apresentação de três vias**. Vários aspectos da apresentação de três vias do TCP são tratados nos exercícios ao final deste capítulo (Por que são necessários os números de sequência iniciais? Por que é preciso uma apresentação de três vias, e não apenas de duas vias?) É interessante notar que um alpinista e seu amarrador (que fica mais abaixo e cuja tarefa é passar a corda de segurança ao alpinista) usa um protocolo de comunicação de apresentação de três vias idêntico ao do TCP para garantir que ambos os lados estejam prontos antes de o alpinista iniciar a escalada.

Tudo o que é bom dura pouco, e o mesmo é válido para uma conexão TCP. Qualquer um dos dois processos que participam de uma conexão TCP pode encerrar a conexão. Quando a conexão termina, os 'recursos' (isto é, os buffers e as variáveis) nos hospedeiros são liberados. Como exemplo, suponha que o cliente decida encerrar a conexão, como mostra a Figura 3.40. O processo de aplicação cliente emite um comando para fechar. Isso faz com que o TCP cliente envie um segmento TCP especial ao processo servidor, cujo bit de flag no cabeçalho do segmento, denominado bit FIN (veja a Figura 3.39), tem valor ajustado em 1. Quando o servidor recebe esse segmento, ele envia de volta ao cliente um segmento de reconhecimento. O servidor então envia seu próprio segmento de encerramento, que tem o bit FIN ajustado em 1. Por fim, o cliente reconhece o segmento de encerramento do servidor. Nesse ponto, todos os recursos dos dois hospedeiros estão liberados.

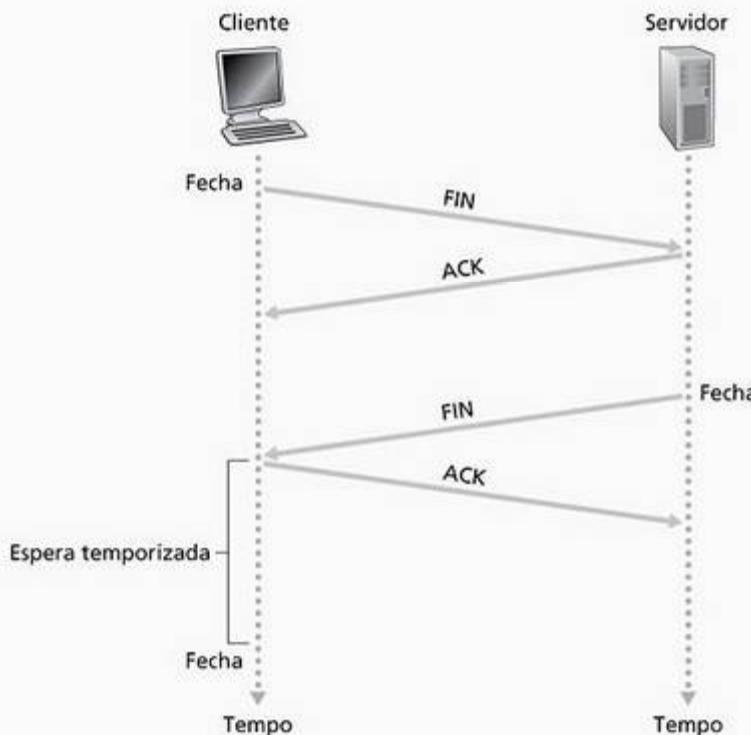


Figura 3.40 Encerramento de uma conexão TCP

Durante a vida de uma conexão TCP, o protocolo TCP que roda em cada hospedeiro faz transições pelos vários **estados do TCP**. A Figura 3.41 ilustra uma sequência típica de estados do TCP visitados pelo TCP cliente. O TCP cliente começa no estado CLOSED. A aplicação no lado cliente inicia uma nova conexão TCP (criando um objeto socket como nos exemplos em Java do Capítulo 2). Isso faz com que o TCP no cliente envie um segmento SYN ao TCP no servidor. Após ter enviado o segmento SYN, o TCP cliente entra no estado SYN_SENT e, enquanto isso, o TCP cliente espera por um segmento do TCP servidor que inclui um reconhecimento para o segmento anterior do cliente, e tem o bit SYN ajustado para o valor 1. Assim que recebe esse segmento, o TCP cliente entra no estado ESTABLISHED, quando pode enviar e receber segmentos TCP que contêm carga útil de dados (isto é, gerados pela aplicação).

Suponha que a aplicação cliente decida que quer fechar a conexão. (Note que o servidor também tem a alternativa de fechá-la.) Isso faz com que o TCP cliente envie um segmento TCP com o bit FIN ajustado em 1 e entre no estado FIN_WAIT_1. No estado FIN_WAIT_1, o TCP cliente espera por um segmento TCP do servidor com um reconhecimento. Quando recebe esse segmento, o TCP cliente entra no estado FIN_WAIT_2. No estado FIN_WAIT_2, ele espera por outro segmento do servidor com o bit FIN ajustado para 1. Após receber esse segmento, o TCP cliente reconhece o segmento do servidor e entra no estado TIME_WAIT. O estado TIME_WAIT permite que o TCP cliente reenvie o reconhecimento final, caso o ACK seja perdido. O tempo passado no estado TIME_WAIT depende da implementação, mas os valores típicos são 30 segundos, 1 minuto e 2 minutos. Após a espera, a conexão se encerra formalmente e todos os recursos do lado cliente (inclusive os números de porta) são liberados.

A Figura 3.42 ilustra a série de estados normalmente visitados pelo TCP do lado servidor, admitindo-se que é o cliente quem inicia o encerramento da conexão. As transições são autoexplicativas. Nesses dois diagramas de transição de estados, mostramos apenas como uma conexão TCP é normalmente estabelecida e fechada. Não descrevemos o que acontece em certos cenários patológicos, por exemplo, quando ambos os lados de uma conexão querem fechar ao mesmo tempo. Se estiver interessado em aprender mais sobre esse assunto e sobre outros mais avançados referentes ao TCP, consulte o abrangente livro de [Stevens, 1994].

Nossa discussão acima concluiu que o cliente e o servidor estão preparados para se comunicar, isto é, que o servidor está ouvindo na porta pela qual o cliente envia seu segmento SYN.

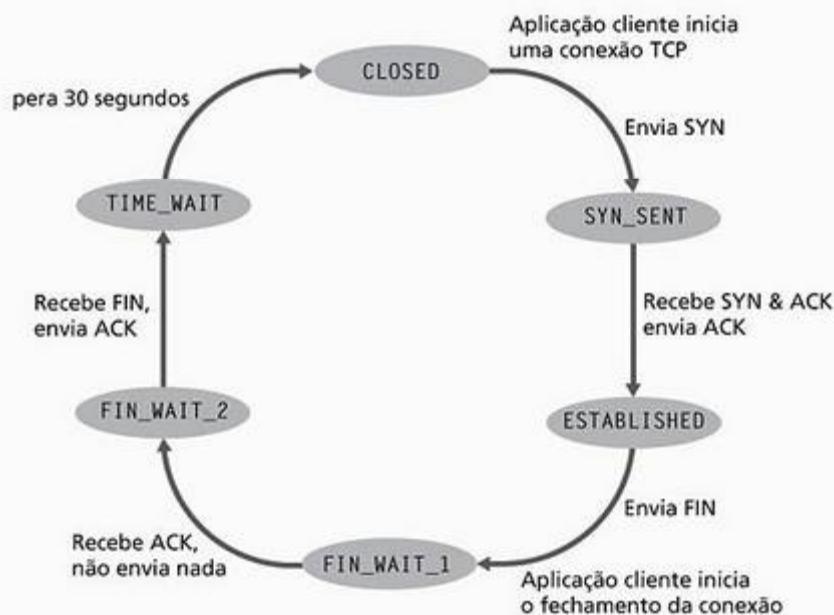


Figura 3.41 Uma sequência típica de estados do TCP visitados por um TCP cliente

 Segurança em foco

O ataque Syn Flood

Vimos em nossa discussão sobre a apresentação de três vias do TCP que um servidor aloca e inicializa as variáveis da conexão e os buffers em resposta ao SYN recebido. O servidor, então, envia um SYNACK em resposta e aguarda um segmento ACK do cliente, o terceiro e último passo na apresentação antes de uma conexão ser completamente estabelecida. Se o cliente não enviar um ACK para completar o terceiro passo da apresentação de três vias, com o tempo (geralmente após um minuto ou mais) o servidor finalizará a conexão semiaberta e recuperará os recursos alocadas.

Esse protocolo de gerenciamento da conexão TCP abre caminho para um ataque DoS clássico, ou seja, o ataque SYN flood. Neste ataque, o vilão envia um grande número de segmentos SYN TCP, sem concluir a terceira etapa de apresentação. O ataque pode ser ampliado enviando os SYNs de diversas fontes, criando um ataque DDoS (recusa de serviço distribuído) SYN flood. Com esse acúmulo de segmentos SYN, os recursos de conexão do servidor podem rapidamente se esgotar já que são alocadas (mas nunca usadas) para conexões semiabertas. Esgotando-se os recursos do servidor, o serviço é negado ao verdadeiro cliente. Esses ataques SYN flood [CERT SYN, 1996] estavam entre os primeiros ataques DoS documentados pelo CERT [CERT, 2009].

O SYN flood é um ataque potencialmente destruidor. Felizmente, há uma proteção eficaz, denominada SYN Cookies [Skoudis, 2006; Cisco SYN, 2009; Bernstein, 2009], agora implementada na maioria dos sistemas operacionais. O SYN Cookie age da seguinte maneira:

- Quando o servidor recebe um segmento SYN, não se sabe se ele vem de um usuário verdadeiro ou se é parte desse ataque. Então, o servidor não cria uma conexão TCP semiaberta para esse SYN. Em vez disso, o servidor cria um número de sequência inicial TCP, uma função hash de endereços de fonte e endereços de destino IP e números de porta do segmento SYN, assim como de um número secreto somente conhecido pelo usuário. (O usuário utiliza o mesmo número secreto para um grande número de conexões.) Esse número de sequência inicial criado cuidadosamente é o assim chamado "cookie". O servidor, então, envia um pacote SYNACK com esse número de sequência especial. É importante mencionar que o servidor não se lembra do cookie ou de qualquer outra informação correspondente ao SYN.
- Se o cliente for verdadeiro, então um segmento ACK retornará. O servidor, ao receber esse ACK, precisa verificar se ele corresponde a algum SYN enviado anteriormente. Como isto é feito se o servidor não guarda nenhuma memória sobre os segmentos SYN? Como você deve ter imaginado, esse processo é realizado com o cookie. Para um ACK legítimo, especificamente, o valor no campo de reconhecimento é igual ao número de sequência no SYNACK mais um (veja Figura 3.39). O servidor, então, executará a mesma função utilizando os mesmos campos no segmento ACK e número secreto. Se o resultado da função mais um for o mesmo que o número de reconhecimento, o servidor conclui que o ACK corresponde a um segmento SYN anterior e, portanto, é válido. O servidor, então, cria uma conexão totalmente aberta com um socket.
- Por outro lado, se o cliente não retorna um segmento ACK, então o SYN original não causou nenhum dano ao servidor, uma vez que este não alocou nenhum recurso para ele!

Os SYN cookies eliminam, efetivamente, a ameaça de um ataque SYN flood. Uma variação desse ataque é o cliente malicioso retornar um segmento ACK válido para cada segmento SYNACK que o servidor gera. Isto fará com que o servidor estabeleça conexões TCP totalmente abertas, mesmo se seu sistema operacional utilizar SYN cookies. Se centenas de clientes estiverem sendo usados (ataque DDoS), cada um de uma fonte de endereço IP diferente, então é difícil o servidor reconhecer as fontes verdadeiras e as maliciosas. Assim, pode ser mais difícil de se proteger desse "ataque de apresentação completado" do que o clássico ataque SYN flood.

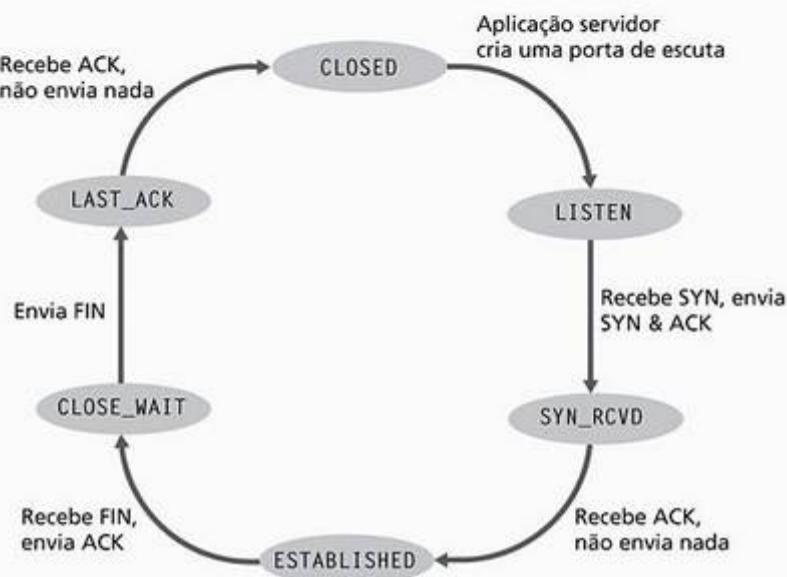


Figura 3.42 Uma sequência típica de estados do TCP visitados por um TCP do lado do servidor

Vamos considerar o que acontece quando um hospedeiro recebe um segmento TCP cujos números de porta ou endereço IP não são compatíveis com nenhuma das portas existentes no hospedeiro. Por exemplo, suponha que um hospedeiro receba um pacote TCP SYN com porta de destino 80, mas não está aceitando conexões nessa porta (isto é, não está rodando um servidor Web na porta 80). Então, ele enviará à fonte um segmento especial de reinicialização. Esse segmento TCP tem o bit de flag RST ajustado para 1 (veja Seção 3.5.2). Assim, quando um hospedeiro envia um segmento de reinicialização ele está dizendo à fonte: “Eu não tenho um socket para esse segmento. Favor não enviá-lo novamente”. Quando um hospedeiro recebe um pacote UDP cujo número de porta de destino não é compatível com as portas de um UDP em curso, ele envia um datagrama ICMP especial, como será discutido no Capítulo 4.

Agora que obtivemos uma boa compreensão sobre gerenciamento da conexão TCP, vamos voltar à ferramenta de varredura de porta nmap e analisar mais precisamente como ela funciona. Para explorar uma porta TCP, digamos que a porta 6789, o nmap enviará ao computador-alvo um segmento TCP SYN com a porta de destino. Os três possíveis resultados são:

- O computador-fonte recebe um segmento TCP SYNACK de um computador-alvo. Como isso significa que uma aplicação está sendo executada com a porta TCP 6789 no computador-alvo, o nmap retorna “aberto”.
- O computador-fonte recebe um segmento TCP RST de um computador-alvo. Isto significa que o segmento SYN atingiu o computador-alvo, mas este não está executando uma aplicação com a porta TCP 6789. Mas o atacante, pelo menos, sabe que os segmentos destinados ao computador na porta 6789 não estão bloqueados pelo firewall no percurso entre o computador-fonte e o alvo. (Firewalls são abordados no Capítulo 8).
- A fonte não recebe nada. Isto, provavelmente, significa que o segmento SYN foi bloqueado pelo firewall e nunca atingiu o computador-alvo.

O nmap é uma ferramenta potente, que pode “examinar o local” não somente para abrir portas TCP, mas também para abrir portas UDP, para firewalls e suas configurações, e até mesmo para as versões de aplicações e sistemas operacionais. A maior parte disto é feito através da manipulação dos segmentos de gerenciamento da conexão TCP [Skoudis, 2006]. Caso você esteja sentado próximo a uma máquina Linux, talvez você queira fazer uma experiência com o nmap apenas digitando o nome da ferramenta na linha de comando. É possível fazer download do nmap para outros sistemas operacionais do site <http://insecure.org/nmap>.

Com isso, concluímos nossa introdução ao controle de erro e controle de fluxo em TCP. Voltaremos ao TCP na Seção 3.7 e então examinaremos mais detalhadamente o controle de congestionamento. Antes, contudo, vamos analisar a questão do controle de congestionamento em um contexto mais amplo.

3.6 Princípios de controle de congestionamento

Nas seções anteriores, examinamos os princípios gerais e também os mecanismos específicos do TCP usados para prover um serviço de transferência confiável de dados em relação à perda de pacotes. Mencionamos anteriormente que, na prática, essa perda resulta, caracteristicamente, de uma saturação de buffers de roteadores à medida que a rede fica congestionada. Assim, a retransmissão de pacotes trata de um sintoma de congestionamento de rede (a perda de um segmento específico de camada de transporte), mas não trata da causa do congestionamento da rede: demasiadas fontes tentando enviar dados a uma taxa muito alta. Para tratar da causa do congestionamento de rede, são necessários mecanismos para regular os remetentes quando esse congestionamento ocorre.

Nesta seção, consideramos o problema do controle de congestionamento em um contexto geral, buscando entender por que o congestionamento é algo ruim, como o congestionamento de rede se manifesta no desempenho recebido por aplicações da camada superior e várias medidas que podem ser adotadas para evitar o congestionamento de rede ou reagir a ele. Esse estudo mais geral do controle de congestionamento é apropriado, já que, como acontece com a transferência confiável de dados, o congestionamento é um dos “dez maiores” da lista de problemas fundamentalmente importantes no trabalho em rede. Concluímos esta seção com uma discussão sobre o controle de congestionamento no serviço de **tакса de bits disponível** (available bit-rate — ABR) em redes com **modo de transferência assíncrono** (ATM). A seção seguinte contém um estudo detalhado do algoritmo de controle de congestionamento do TCP.

3.6.1 As causas e os custos do congestionamento

Vamos começar nosso estudo geral do controle de congestionamento examinando três cenários de complexidade crescente nos quais ocorre o congestionamento. Em cada caso, examinaremos, primeiramente, por que ele ocorre e, depois, seu custo (no que se refere aos recursos não utilizados integralmente e ao baixo desempenho recebido pelos sistemas finais). Não focalizaremos (ainda) como reagir ao congestionamento, ou evitá-lo; preferimos estudar uma questão mais simples, que é entender o que acontece quando hospedeiros aumentam sua taxa de transmissão e a rede fica congestionada.

Cenário 1: dois remetentes, um roteador com buffers infinitos

Começamos considerando o que talvez seja o cenário de congestionamento mais simples possível: dois hospedeiros (A e B), cada um com uma conexão que compartilha um único trecho de rede entre a fonte e o destino, como mostra a Figura 3.43.

Vamos admitir que a aplicação no hospedeiro A esteja enviando dados para a conexão (por exemplo, passando dados para o protocolo de camada de transporte por um socket) a uma taxa média de λ_{in} bytes/segundo. Esses dados são originais no sentido de que cada unidade de dados é enviada para dentro do socket apenas uma vez. O protocolo de camada de transporte subjacente é simples. Os dados são encapsulados e enviados; não há recuperação de erros (por exemplo, retransmissão), controle de fluxo, nem controle de congestionamento. Desprezando a sobrecarga adicional causada pela adição de informações de cabeçalhos de camada de transporte e de camadas mais baixas, a taxa à qual o hospedeiro A oferece tráfego ao roteador nesse primeiro cenário é λ_{in} bytes/segundo. O hospedeiro B funciona de maneira semelhante, e admitimos, por simplicidade, que ele também esteja enviando dados a uma taxa de λ_{in} bytes/segundo. Os pacotes dos hospedeiros A e B passam por um roteador e por um enlace de saída compartilhado de capacidade R. O roteador tem buffers que lhe permitem armazenar os pacotes que chegam quando a taxa de chegada de pacotes excede a capacidade do enlace de saída. No primeiro cenário, admitimos que o roteador tenha capacidade de armazenamento infinita.

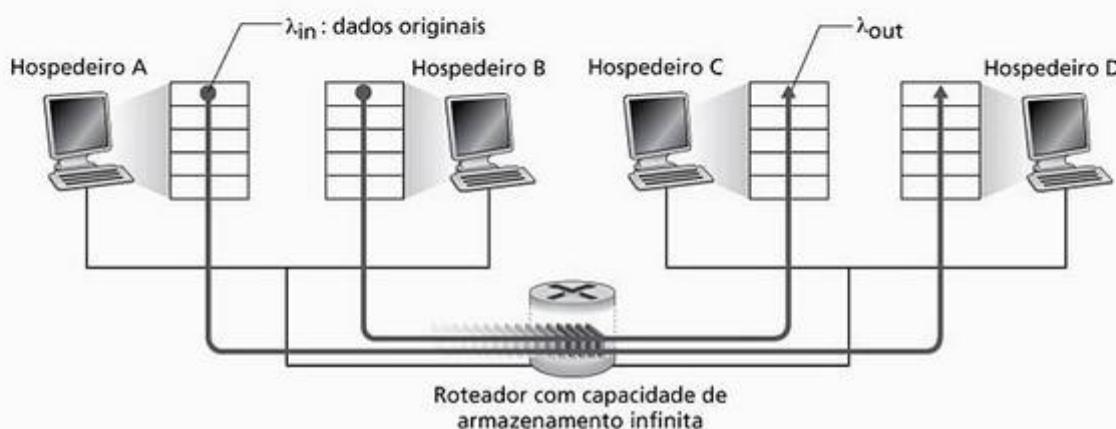


Figura 3.43 Cenário de congestionamento 1: duas conexões compartilhando um único roteador com número infinito de buffers

A Figura 3.44 apresenta o desempenho da conexão do hospedeiro A nesse primeiro cenário. O gráfico da esquerda dessa figura apresenta a **vazão por conexão** (número de bytes por segundo no destinatário) como uma função da taxa de envio da conexão. Para uma taxa de transmissão entre 0 e $R/2$, a vazão no destinatário é igual à velocidade de envio do remetente — tudo o que o remetente envia é recebido no destinatário com um atraso finito. Quando a velocidade de envio estiver acima de $R/2$, contudo, a vazão será somente $R/2$. Esse limite superior da vazão é consequência do compartilhamento da capacidade do enlace entre duas conexões. O enlace simplesmente não consegue entregar os pacotes a um destinatário com uma taxa em regime que excede $R/2$. Não importa quão altas sejam as taxas de envio ajustadas nos hospedeiros A e B, eles jamais alcançarão uma vazão maior do que $R/2$.

Alcançar uma vazão de $R/2$ por conexão pode até parecer uma coisa boa, pois o enlace está sendo integralmente utilizado para entregar pacotes no destinatário. No entanto, o gráfico do lado direito da Figura 3.44 mostra as consequências de operar próximo à capacidade máxima do enlace. À medida que a taxa de envio se aproxima de $R/2$ (partindo da esquerda), o atraso médio fica cada vez maior. Quando a taxa de envio ultrapassa $R/2$, o número médio de pacotes na fila no roteador é ilimitado e o atraso médio entre a fonte e o destino se torna infinito (admitindo que as conexões operem a essas velocidades de transmissão durante um período de tempo infinito e que a capacidade de armazenamento também seja infinita). Assim, embora operar a uma vazão agregada próxima a R possa ser ideal do ponto de vista da vazão, está bem longe de ser ideal do ponto de vista do atraso. Mesmo

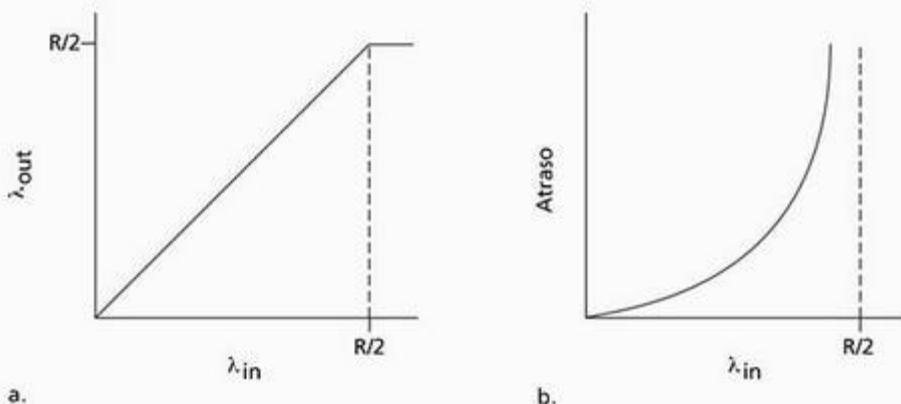


Figura 3.44 Cenário de congestionamento 1: vazão e atraso em função da taxa de envio do hospedeiro

nesse cenário (extremamente) idealizado, já descobrimos um custo da rede congestionada — há grandes atrasos de fila quando a taxa de chegada de pacotes se aproxima da capacidade do enlace.

Cenário 2: dois remetentes, um roteador com buffers finitos

Vamos agora modificar ligeiramente o cenário 1 dos dois modos seguintes (veja a Figura 3.45). Em primeiro lugar, admitamos que a capacidade de armazenamento do roteador seja finita. Em uma situação real, essa suposição teria como consequência o descarte de pacotes que chegam a um buffer que já está cheio. Em segundo lugar, admitamos que cada conexão seja confiável. Se um pacote contendo um segmento de camada de transporte for descartado no roteador, o remetente por fim o retransmitirá. Como os pacotes podem ser retransmitidos, agora temos de ser mais cuidadosos com o uso da expressão “taxa de envio”. Especificamente, vamos novamente designar a taxa com que a aplicação envia dados originais para dentro do socket como λ_{in} bytes/segundo. A taxa com que a camada de transporte envia segmentos (contendo dados originais e dados retransmitidos) para dentro da rede será denominada λ'_{in} bytes/segundo. Essa taxa (λ'_{in}) às vezes é denominada **carga oferecida** à rede.

O desempenho obtido no cenário 2 agora dependerá muito de como a retransmissão é realizada. Primeiramente, considere o caso não realista em que o hospedeiro A consiga, de algum modo (fazendo mágica!), determinar se um buffer do roteador está livre ou não. Assim, o hospedeiro A envia um pacote somente quando o buffer estiver livre. Nesse caso, não ocorreria nenhuma perda, λ_{in} seria igual a λ'_{in} e a vazão da conexão seria igual a λ_{in} . Esse caso é mostrado pela curva superior da Figura 3.46(a).

Do ponto de vista da vazão, o desempenho é ideal — tudo o que é enviado é recebido. Note que, nesse cenário, a taxa média de envio do hospedeiro não pode ultrapassar $R/2$, já que admitimos que nunca ocorre perda de pacote.

Considere, em seguida, o caso um pouco mais realista em que o remetente retransmite somente quando sabe, com certeza, que o pacote foi perdido. (Novamente, essa suposição é um pouco forçada. Contudo, é possível ao hospedeiro remetente ajustar seu temporizador de retransmissão para uma duração longa o suficiente para ter razoável certeza de que um pacote que não foi reconhecido foi perdido.) Nesse caso, o desempenho pode ser parecido com o que é mostrado na Figura 3.46(b). Para avaliar o que está acontecendo aqui, considere o caso em que a carga oferecida, λ_{in} (a taxa de transmissão dos dados originais mais as retransmissões) é igual a $R/2$. De acordo com a Figura 3.46(b), nesse valor de carga oferecida, a velocidade com a qual os dados são entregues à aplicação do destinatário é $R/3$. Assim, de 0,5R unidade de dados transmitida, 0,333R byte/segundo (em média) são dados originais e 0,166R byte/segundo (em média) são dados retransmitidos. *Observamos aqui outro custo de*

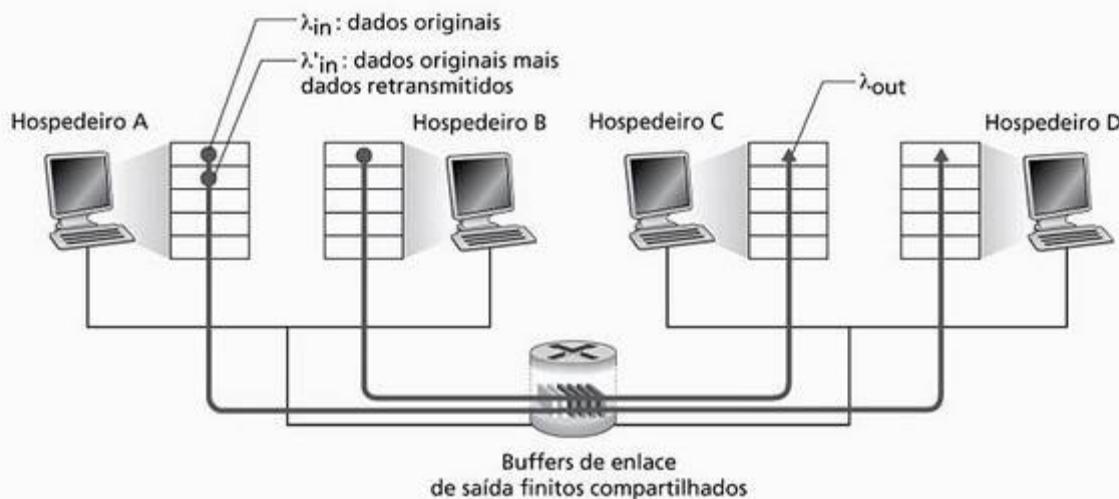


Figura 3.45 Cenário 2: dois hospedeiros (com retransmissões) e um roteador com buffers finitos

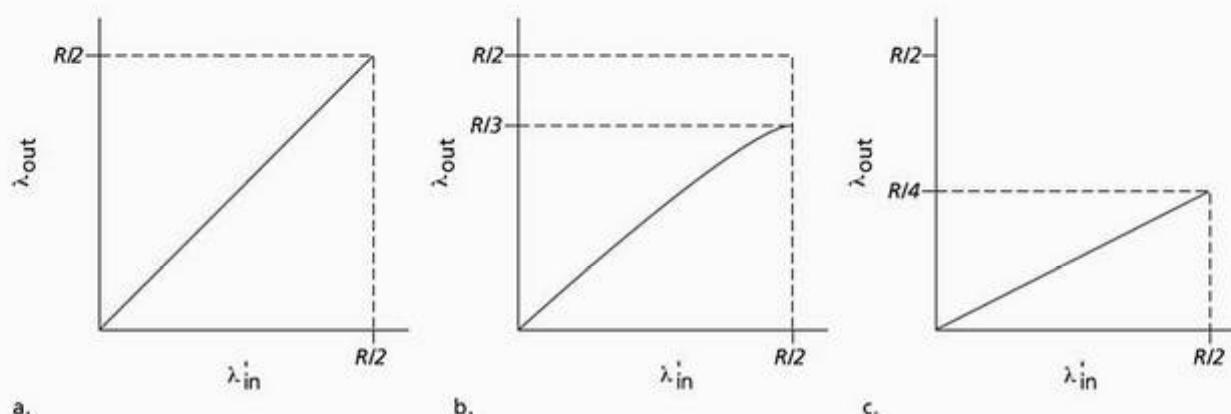


Figura 3.46 Desempenho no cenário 2 com buffers finitos

uma rede congestionada — o remetente deve realizar retransmissões para compensar os pacotes descartados (perdidos) devido ao esgotamento do buffer.

Finalmente, vamos considerar o caso em que a temporização do remetente se esgota prematuramente e ele retransmita um pacote que ficou atrasado na fila, mas que ainda não está perdido. Aqui, tanto o pacote de dados original quanto a retransmissão podem alcançar o destinatário. É claro que o destinatário precisa apenas de uma cópia desse pacote e descartará a retransmissão. Nesse caso, o trabalho realizado pelo roteador ao repassar a cópia retransmitida do pacote original é desperdiçado, pois o destinatário já terá recebido a cópia original desse pacote. Em vez disso, seria melhor o roteador usar a capacidade de transmissão do enlace para enviar um pacote diferente. *Eis aqui mais um custo da rede congestionada — retransmissões desnecessárias feitas pelo remetente em face de grandes atrasos podem fazer com que um roteador use sua largura de banda de enlace para repassar cópias desnecessárias de um pacote.* A Figura 3.4.6(c) mostra a vazão versus a carga oferecida admitindo-se que cada pacote seja enviado (em média) duas vezes pelo roteador. Visto que cada pacote é enviado duas vezes, a vazão terá um valor assintótico de $R/4$ à medida que a carga oferecida se aproximar de $R/2$.

Cenário 3: quatro remetentes, roteadores com buffers finitos e trajetos com múltiplos roteadores

Em nosso cenário final de congestionamento, quatro hospedeiros transmitem pacotes sobre trajetos sobrepostos que apresentam dois saltos, como ilustrado na Figura 3.47. Novamente admitamos que cada hospedeiro use um mecanismo de temporização/retransmissão para implementar um serviço de transferência confiável de dados, que todos os hospedeiros tenham o mesmo valor de λ_{in} e que todos os enlaces dos roteadores tenham capacidade de R bytes/segundo.

Vamos considerar a conexão do hospedeiro A ao hospedeiro C que passa pelos roteadores R1 e R2. A conexão A-C compartilha o roteador R1 com a conexão D-B e o roteador 2 com a conexão B-D. Para valores extremamente pequenos de λ_{in} , esgotamentos de buffers são raros (como acontecia nos cenários de congestionamento 1 e 2) e a vazão é quase igual à carga oferecida. Para valores de λ_{in} ligeiramente maiores, a vazão correspondente é também maior, pois mais dados originais estão sendo transmitidos para a rede e entregues no destino, e os esgotamentos ainda são raros. Assim, para valores pequenos de λ_{in} , um aumento em λ_{in} resulta em um aumento em λ_{out} .

Como já analisamos o caso de tráfego extremamente baixo, vamos examinar aquele em que λ_{in} (e, portanto, λ'_{in}) é extremamente alto. Considere o roteador R2. O tráfego A-C que chega ao roteador R2 (após ter sido repassado de R1) pode ter uma taxa de chegada em R2 de, no máximo, R , que é a capacidade do enlace de R1 a R2, não importando qual seja o valor de λ_{in} . Se λ'_{in} for extremamente alto para todas as conexões (incluindo a conexão B-D), então a taxa de chegada do tráfego B-D em R2 poderá ser muito maior do que a taxa do tráfego A-C. Como os tráfegos A-C e B-D têm de competir no roteador R2 pelo espaço limitado de buffer, a quantidade de tráfego A-C que consegue passar por R2 (isto é, que não se perde devido ao congestionamento de buffer) diminui cada vez mais à medida que

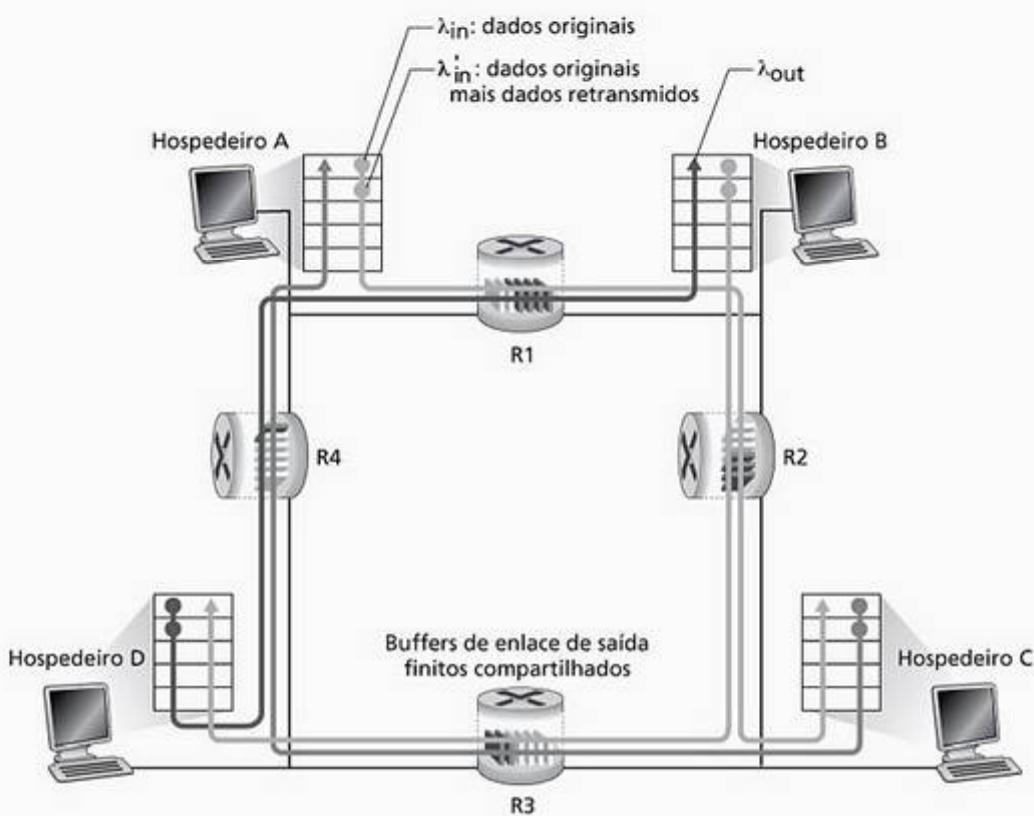


Figura 3.47 Quatro remetentes, roteadores com buffers finitos e trajetos com vários saltos

a carga oferecida de B-D vai ficando maior. No limite, quando a carga oferecida se aproxima do infinito, um buffer vazio em R2 é imediatamente preenchido por um pacote B-D e a vazão da conexão A-C em R2 cai a zero. Isso, por sua vez, implica que a vazão sim a fim de A-C vai a zero no limite de tráfego pesado. Essas considerações dão origem ao comportamento da carga oferecida versus a vazão mostrada na Figura 3.48.

A razão para o decréscimo final da vazão com o crescimento da carga oferecida é evidente quando consideramos a quantidade de trabalho desperdiçado realizado pela rede. No cenário de alto tráfego que acabamos de descrever, sempre que um pacote é descartado em um segundo roteador, o trabalho realizado pelo primeiro para enviar o pacote ao segundo acaba sendo 'desperdiçado'. A rede teria se saído igualmente bem (melhor dizendo,

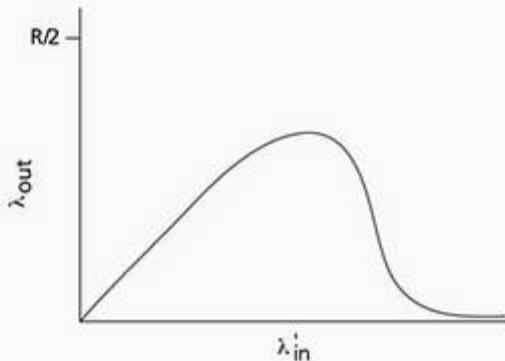


Figura 3.48 Desempenho obtido no cenário 3, com buffers finitos e trajetos com múltiplos roteadores

igualmente mal) se o primeiro roteador tivesse simplesmente descartado aquele pacote e tivesse ficado inativo. Mais objetivamente, a capacidade de transmissão utilizada no primeiro roteador para enviar o pacote ao segundo teria sido maximizada para transmitir um pacote diferente. (Por exemplo, ao selecionar um pacote para transmissão, seria melhor que um roteador desse prioridade a pacotes que já atravessaram alguns roteadores anteriores.) Portanto, vemos aqui mais um custo, o do descarte de pacotes devido ao congestionamento — quando um pacote é descartado ao longo de um caminho, a capacidade de transmissão que foi usada em cada um dos enlaces anteriores para repassar o pacote até o ponto em que foi descartado acaba sendo desperdiçada.

3.6.2 Mecanismos de controle de congestionamento

Na Seção 3.7, examinaremos detalhadamente os mecanismos específicos do TCP para o controle de congestionamento. Nesta subseção, identificaremos os dois procedimentos mais comuns adotados, na prática, para o controle de congestionamento. Além disso, examinaremos arquiteturas específicas de rede e protocolos de controle de congestionamento que incorporam esses procedimentos.

No nível mais amplo, podemos distinguir mecanismos de controle de congestionamento conforme a camada de rede ofereça ou não assistência explícita à camada de transporte com a finalidade de controle de congestionamento:

Controle de congestionamento fim a fim. Nesse método para o controle de congestionamento, a camada de rede não fornece nenhum suporte explícito à camada de transporte com a finalidade de controle de congestionamento. Até mesmo a presença de congestionamento na rede deve ser intuída pelos sistemas finais com base apenas na observação do comportamento da rede (por exemplo, perda de pacotes e atraso). Veremos na Seção 3.7 que o TCP deve necessariamente adotar esse método fim a fim para o controle de congestionamento, uma vez que a camada IP não fornece realimentação de informações aos sistemas finais quanto ao congestionamento da rede. A perda de segmentos TCP (apontada por uma temporização ou por três reconhecimentos duplicados) é tomada como indicação de congestionamento, e o TCP reduz o tamanho da janela de acordo com isso. Veremos também que as novas propostas para o TCP usam valores de atraso de viagem de ida e volta crescentes como indicadores de aumento do congestionamento da rede.

Controle de congestionamento assistido pela rede. Com esse método, os componentes da camada de rede (isto é, roteadores) fornecem realimentação específica de informações ao remetente a respeito do estado de congestionamento na rede. Essa realimentação pode ser tão simples como um único bit indicando o congestionamento em um enlace. Adotada nas primeiras arquiteturas de rede IBM SNA [Schwartz, 1982] e DEC DECnet [Jain, 1989; Ramakrishnan, 1990], essa abordagem foi proposta recentemente para redes TCP/IP [Floyd TCP, 1994; RFC 3168]; ela é usada também no controle de congestionamento em ATM com serviço de transmissão ABR, como discutido a seguir. A realimentação mais sofisticada de rede também é possível. Por exemplo, uma forma de controle de congestionamento ATM ABR que estudaremos mais adiante permite que um roteador informe explicitamente ao remetente a velocidade de transmissão que ele (o roteador) pode suportar em um enlace de saída. O protocolo XCP [Katabi, 2002] provê um retorno (feedback) calculado pelo roteador para cada fonte, transmitido no cabeçalho do pacote, referente ao modo que essa fonte deve aumentar ou diminuir sua taxa de transmissão.

Para controle de congestionamento assistido pela rede, a informação sobre congestionamento é normalmente realimentada da rede para o remetente por um de dois modos, como mostra a Figura 3.49. Realimentação direta pode ser enviada de um roteador de rede ao remetente. Esse modo de notificação normalmente toma a forma de um **pacote de congestionamento** (*choke packet*) (que, em essência, diz: “Estou congestionado!”). O segundo modo de notificação ocorre quando um roteador marca/atualiza um campo em um pacote que está fluindo do remetente ao destinatário para indicar congestionamento. Ao receber um pacote marcado, o destinatário informa ao remetente a indicação de congestionamento. Note que esse último modo de notificação leva, no mínimo, o tempo total de uma viagem de ida e volta.

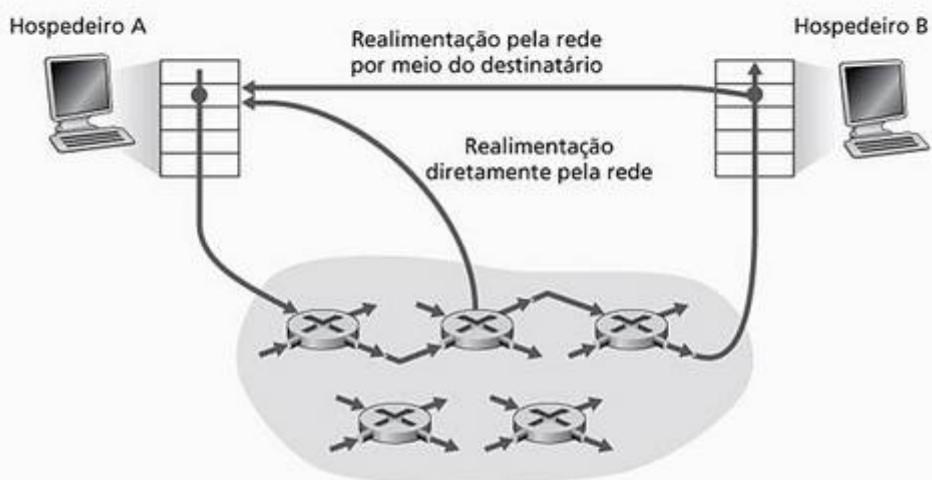


Figura 3.49 Dois caminhos de realimentação para informação sobre congestionamento indicado pela rede

3.6.3 Exemplo de controle de congestionamento assistido pela rede: controle de congestionamento ATM ABR

Concluímos esta seção com um breve estudo de caso do algoritmo de controle de congestionamento ATM ABR. Salientamos que nossa meta aqui *não* é, de modo algum, descrever detalhadamente aspectos da arquitetura ATM, mas mostrar um protocolo que adota uma abordagem de controle de congestionamento notavelmente diferente da adotada pelo protocolo TCP da Internet. Na verdade, apresentamos a seguir apenas os poucos aspectos da arquitetura ATM necessários para entender o controle de congestionamento ABR.

Fundamentalmente, o ATM adota uma abordagem orientada para circuito virtual (CV) da comutação de pacotes. Lembre, da nossa discussão no Capítulo 1, que isso significa que cada comutador no caminho entre a fonte e o destino manterá estado sobre o CV entre a fonte e o destino. Esse estado por CV permite que um comutador monitore o comportamento de remetentes individuais (por exemplo, monitorando sua taxa média de transmissão) e realize ações específicas de controle de congestionamento (tais como sinalizar explicitamente ao remetente para que ele reduza sua taxa quando o comutador fica congestionado). Esse estado por CV em comutadores de rede torna o ATM idealmente adequado para realizar controle de congestionamento assistido pela rede.

O ABR foi projetado como um serviço de transferência de dados elástico que faz lembrar, de certo modo, o TCP. Quando a rede está vazia, o serviço ABR deve ser capaz de aproveitar a vantagem da largura de banda disponível. Quando a rede está congestionada, deve limitar sua taxa de transmissão a algum valor mínimo predefinido de taxa de transmissão. Um tutorial detalhado sobre controle de congestionamento e gerenciamento de tráfego ATM ABR é fornecido por [Jain, 1996].

A Figura 3.50 mostra a estrutura do controle de congestionamento para ATM ABR. Nessa discussão, adotaremos a terminologia ATM (por exemplo, usaremos o termo ‘comutador’ em vez de ‘roteador’ e o termo ‘célula’ em vez de ‘pacote’). Com o serviço ATM ABR, as células de dados são transmitidas de uma fonte a um destino por meio de uma série de comutadores intermediários. Intercaladas às células de dados estão as **células de gerenciamento de recursos**, ou **células RM** (resource-management cells); essas células podem ser usadas para portar informações relacionadas ao congestionamento entre hospedeiros e comutadores. Quando uma célula RM chega a um destinatário, ela será ‘virada’ e enviada de volta ao remetente (possivelmente após o destinatário ter modificado o conteúdo dela). Também é possível que um comutador gere, ele mesmo, uma célula RM e a envie diretamente a uma fonte. Assim, células RM podem ser usadas para fornecer realimentação diretamente da rede e também realimentação da rede por intermédio do destinatário, como mostra a Figura 3.50.

O controle de congestionamento ATM ABR é um método baseado em taxa. O remetente estima explicitamente uma taxa máxima na qual pode enviar e se autorregula de acordo com ela. O serviço ABR provê três mecanismos para sinalizar informações relacionadas a congestionamento dos comutadores ao destinatário:

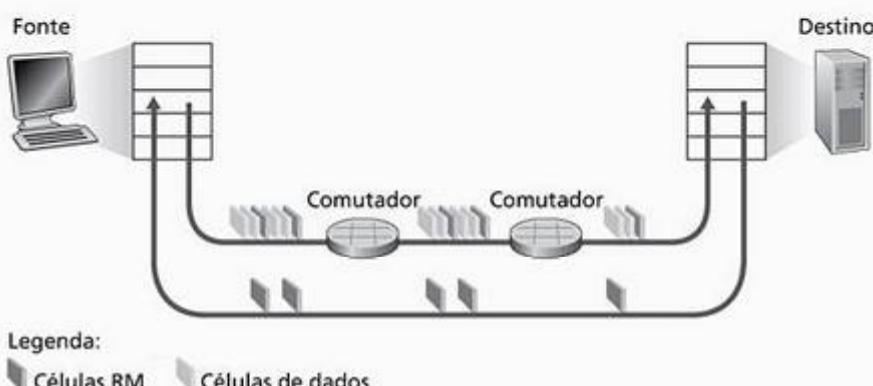


Figura 3.50 Estrutura de controle de congestionamento ATM ABR

Bit EFCI. Cada célula de dados contém um **bit EFCI de indicação explícita de congestionamento** à frente (*explicit forward congestion indication*). Um comutador de rede congestionado pode modificar o bit EFCI dentro de uma célula para 1, a fim de sinalizar congestionamento ao hospedeiro destinatário, que deve verificar o bit EFCI em todas as células de dados recebidas. Quando uma célula RM chega ao destinatário, se a célula de dados recebida mais recentemente tiver o bit EFCI com valor 1, o destinatário modifica o bit de indicação de congestionamento (o bit CI) da célula RM para 1 e envia a célula RM de volta ao remetente. Usando o bit EFCI em células de dados e o bit CI em células RM, um remetente pode ser notificado sobre congestionamento em um comutador da rede.

Bits CI e NI. Como observado anteriormente, células RM remetente/destinatário estão intercaladas com células de dados. A taxa de intercalação de células RM é um parâmetro ajustável, sendo uma célula RM a cada 32 células de dados o valor default. Essas células RM têm um bit de indicação de congestionamento (CI) e um bit NI (*no increase* — não aumentar) que podem ser ajustados por um comutador de rede congestionado. Especificamente, um comutador pode modificar para 1 o bit NI de uma célula RM que está passando quando a condição de congestionamento é leve e pode modificar o bit CI para 1 sob severas condições de congestionamento. Quando um hospedeiro destinatário recebe uma célula RM, ele a enviará de volta ao remetente com seus bits CI e NI inalterados (exceto que o CI pode ser ajustado para 1 pelo destinatário como resultado do mecanismo EFCI descrito antes).

Ajuste de ER. Cada célula RM contém também um **campo ER** (*explicit rate* — taxa explícita) de dois bytes. Um comutador congestionado pode reduzir o valor contido no campo ER de uma célula RM que está passando. Desse modo, o campo ER será ajustado para a taxa mínima suportável por todos os comutadores no trajeto fonte-destino.

Uma fonte ATM ABR ajusta a taxa na qual pode enviar células como uma função dos valores de CI, NI e ER de uma célula RM devolvida. As regras para fazer esse ajuste de taxa são bastante complicadas e um tanto tediosas. O leitor poderá consultar [Jain, 1996] se quiser saber mais detalhes.

3.7 Controle de congestionamento no TCP

Nesta seção, voltamos ao estudo do TCP. Como aprendemos na Seção 3.5, o TCP provê um serviço de transferência confiável entre dois processos que rodam em hospedeiros diferentes. Outro componente extremamente importante do TCP é seu mecanismo de controle de congestionamento. Como indicamos na seção anterior, o TCP deve usar controle de congestionamento fim a fim em vez de controle de congestionamento assistido pela rede, já que a camada IP não fornece aos sistemas finais realimentação explícita relativa ao congestionamento da rede.

A abordagem adotada pelo TCP é obrigar cada remetente a limitar a taxa à qual enviam tráfego para sua conexão como uma função do congestionamento de rede percebido. Se um remetente TCP perceber que há pouco congestionamento no caminho entre ele e o destinatário, aumentará sua taxa de envio; se perceber que há congestionamento ao longo do caminho, reduzirá sua taxa de envio. Mas essa abordagem levanta três questões. A primeira é como um remetente TCP limita a taxa à qual envia tráfego para sua conexão? A segunda é como um remetente TCP percebe que há congestionamento entre ele e o destinatário? E a terceira, que algoritmo o remetente deve utilizar para modificar sua taxa de envio como uma função do congestionamento fim a fim?

Em primeiro lugar, vamos examinar como um remetente TCP limita a taxa de envio à qual envia tráfego para sua conexão. Na Seção 3.5, vimos que cada lado de uma conexão TCP consiste em um buffer de recepção, um buffer de envio e diversas variáveis (*LastByteRead*, *rwnd* e assim por diante). O mecanismo de controle de congestionamento que opera no remetente monitora uma variável adicional, a **janela de congestionamento**. A janela de congestionamento, denominada *cwnd*, impõe uma limitação à taxa à qual um remetente TCP pode enviar tráfego para dentro da rede. Especificamente, a quantidade de dados não reconhecidos em um hospedeiro não pode exceder o mínimo de *cwnd* e *rwnd*, ou seja:

$$\text{LastByteSent} - \text{LastByteAcked} \leq \min(\text{cwnd}, \text{rwnd})$$

Para focalizar a discussão no controle de congestionamento (ao contrário do controle de fluxo), daqui em diante vamos admitir que o buffer de recepção TCP seja tão grande que a limitação da janela de recepção pode ser desprezada; assim, a quantidade de dados não reconhecidos no remetente estará limitada exclusivamente por *cwnd*. Vamos admitir também que o remetente sempre tenha dados para enviar, isto é, que todos os segmentos dentro da janela de congestionamento sejam enviados.

A restrição acima limita a quantidade de dados não reconhecidos no remetente e, por conseguinte, limita indiretamente a taxa de envio do remetente. Para entender melhor, considere uma conexão na qual perdas e atrasos de transmissão de pacotes sejam desprezíveis. Então, em linhas gerais, no início de cada RTT a limitação permite que o remetente envie *cwnd* bytes de dados para a conexão; ao final do RTT, o remetente recebe reconhecimentos para os dados. Assim, a taxa de envio do remetente é aproximadamente *cwnd/RTT bytes por segundo*. Portanto, ajustando o valor de *cwnd*, o remetente pode ajustar a taxa à qual envia dados para sua conexão.

Em seguida, vamos considerar como um remetente TCP percebe que há congestionamento no caminho entre ele e o destino. Definimos “evento de perda” em um remetente TCP como a ocorrência de um esgotamento de temporização ou do recebimento de três ACKs duplicados do destinatário (lembre-se da nossa discussão, na Seção 3.5.4, do evento de temporização apresentado na Figura 3.33 e da subsequente modificação para incluir transmissão rápida quando do recebimento de três ACKs duplicados). Quando há congestionamento excessivo, então um (ou mais) buffers de roteadores ao longo do caminho transborda, fazendo com que um datagrama (contendo um segmento TCP) seja descartado. Esse datagrama descartado, por sua vez, resulta em um evento de perda no remetente — ou um esgotamento de temporização ou o recebimento de três ACKs duplicados — que é tomado por ele como uma indicação de congestionamento no caminho remetente-destinatário.

Já consideramos como é detectado o congestionamento; agora vamos considerar o caso mais otimista de uma rede livre de congestionamento, isto é, quando não ocorre um evento de perda. Nesse caso, o TCP remetente receberá reconhecimentos para segmentos anteriormente não reconhecidos. Como veremos, o TCP considerará a chegada desses reconhecimentos como uma indicação de que tudo está bem — os segmentos que estão sendo transmitidos para a rede estão sendo entregues com sucesso no destinatário — e usará reconhecimentos para aumentar o tamanho de sua janela de congestionamento (e, por conseguinte, sua taxa de transmissão). Note que, se os reconhecimentos chegarem ao remetente a uma taxa relativamente baixa (por exemplo, se o atraso no caminho fime a fime for alto ou se nele houver um enlace de baixa largura de banda), então a janela de congestionamento será aumentada a uma taxa relativamente baixa. Por outro lado, se os reconhecimentos chegarem a uma taxa alta, então a janela de congestionamento será aumentada mais rapidamente. Como o TCP utiliza reconhecimentos para acionar (ou regular) o aumento de tamanho de sua janela de congestionamento, diz-se que o TCP é **autorregulado**.

Dado o mecanismo de ajustar o valor de cwnd para controlar a taxa de envio, permanece a importante pergunta: Como um remetente TCP deve determinar a taxa a que deve enviar? Se os remetentes TCP enviam coletivamente muito rápido, eles podem congestionar a rede, levando ao tipo de congestionamento que vimos na Figura 3.8. Realmente, a versão de TCP que vamos estudar brevemente foi desenvolvida em resposta aos congestionamentos da Internet observados [Jacobson, 1988] sob versões anteriores do TCP. Entretanto, se os remetentes forem muito cautelosos e enviarem lentamente, eles podem subutilizar a largura de banda na rede; ou seja, os remetentes TCP podem enviar a uma taxa mais alta sem congestionar a rede. Então, como os remetentes TCP determinam suas taxas de envio de um modo que não congestionem a rede mas, ao mesmo tempo, façam uso de toda a largura de banda? Os remetentes TCP são claramente coordenados, ou existe uma abordagem distribuída na qual eles podem ajustar suas taxas de envio baseando-se somente nas informações locais? O TCP responde a essas perguntas utilizando os seguintes princípios:

Um segmento perdido implica em congestionamento, portanto, a taxa do remetente TCP deve diminuir quando um segmento é perdido. Lembre-se da nossa discussão na Seção 3.5.4, que um evento de esgotamento do temporizador ou o recebimento de quatro reconhecimentos para um certo segmento (um ACK original e, depois, três ACKs duplicados) é interpretado como uma indicação de “evento de perda” absoluto do segmento subsequente ao segmento ACK quadruplicado, acionando uma retransmissão do segmento perdido. De um ponto de vista do controle de congestionamento, a pergunta é como o remetente TCP deve diminuir sua janela de congestionamento e, portanto, sua taxa de envio, em resposta ao suposto evento de perda.

Um segmento reconhecido indica que a rede está enviando os segmentos do remetente ao destinatário e, por isso, a taxa do remetente pode aumentar quando um ACK chegar para um segmento anteriormente não reconhecido. A chegada de reconhecimentos é tida como uma indicação absoluta de que tudo está bem — os segmentos estão sendo enviados com sucesso do remetente ao destinatário, fazendo, assim, com que a rede não fique congestionada. Dessa forma, o tamanho da janela de congestionamento pode ser elevado.

Busca por largura de banda. Dado os ACKs que indicam um percurso de fonte a destino sem congestionamento, e eventos de perda que indicam um percurso congestionado, a estratégia do TCP de ajustar sua taxa de transmissão é aumentar sua taxa em resposta aos ACKs que chegam até que ocorra um evento de perda, momento em que a taxa de transmissão diminui. Desse modo, o remetente TCP aumenta sua taxa de transmissão para buscar a taxa a qual o congestionamento se inicia, recua dessa taxa e novamente faz a busca para ver se a taxa de início do congestionamento foi alterada. O comportamento do remetente TCP é análogo a uma criança que pede (e ganha) cada vez mais doces até finalmente receber um “Não!”, recuar, mas começar a pedir novamente pouco tempo depois. Observe que não há nenhuma sinalização explícita de congestionamento pela rede — os ACKs e eventos de perda servem como sinais implícitos — e que cada remetente TCP atua em informações locais em momentos diferentes de outros remetentes TCP.

Após essa visão geral sobre controle de congestionamento no TCP, agora podemos considerar os detalhes renomado algoritmo de controle de congestionamento no TCP, sendo primeiramente descrito em [Jacobson, 1988] e padronizado em [RFC 2581]. O algoritmo possui três componentes principais: (1) partida lenta, (2) contenção de congestionamento e (3) recuperação rápida. A partida lenta e a contenção de congestionamento são componentes obrigatórios do TCP, diferenciando em como eles aumentam o tamanho do cwnd em resposta a ACKs recebidos. Abordaremos em poucas palavras que a partida lenta aumenta o tamanho do cwnd de forma mais rápida (apesar do nome!) do que a contenção de congestionamento. A recuperação rápida é recomendada, mas não exigida, para remetentes TCP.

Partida lenta

Quando uma conexão TCP começa, o valor de cwnd normalmente é inicializado em 1 MSS [RFC 3390], resultando em uma taxa inicial de envio de aproximadamente MSS/RTT. Como exemplo, se MSS = 500 bytes e RTT = 200 milissegundos, então a taxa de envio inicial resultante é aproximadamente 20 kbps apenas. Como a largura de banda disponível para a conexão pode ser muito maior do que MSS/RTT, o remetente TCP ia gostar

de aumentar a quantia de largura de banda rapidamente. Dessa forma, no estado de partida lenta, o valor de cwnd começa em 1 MSS e aumenta 1 MSS toda vez que um segmento transmitido é reconhecido. No exemplo da Figura 3.51, o TCP envia o primeiro segmento para a rede e aguarda um reconhecimento. Quando o reconhecimento chega, o remetente TCP aumenta a janela de congestionamento para 1 MSS e envia dois segmentos de tamanho máximo. Esses segmentos são reconhecidos, e o remetente aumenta a janela de congestionamento para 1 MSS para cada reconhecimento de segmento, fornecendo uma janela de congestionamento de 4 MSS e assim por diante. Esse processo resulta em uma multiplicação da taxa de envio a cada RTT. Assim, a taxa de envio TCP se inicia lenta, mas cresce exponencialmente durante a fase de partida lenta.

Mas em que momento esse crescimento exponencial termina? A partida lenta apresenta diversas respostas para essa pergunta. Primeiro, se houver um evento de perda (ou seja, um congestionamento) indicado por um esgotamento de temporização, o remetente TCP estabelece o valor de cwnd para 1 e inicia o processo de partida lenta novamente. Ele também estabelece o valor de uma segunda variável de estado, ssthresh (abreviação de "slow start threshold [limiar de partida lenta]") para $cwnd / 2$ — metade do valor da janela de congestionamento quando este foi detectado. O segundo modo pelo qual a partida lenta pode terminar é ligado diretamente ao valor de ssthresh. Visto que ssthresh é metade do valor de cwnd quando o congestionamento foi detectado pela última vez, pode ser uma atitude precipitada continuar duplicando cwnd ao atingir ou ultrapassar o valor de ssthresh. Assim, quando o valor de cwnd se igualar ao de ssthresh, a partida lenta termina e o TCP é alterado para o modo de prevenção de congestionamento. Como veremos, o TCP aumenta cwnd com mais cautela quando está no modo de prevenção de congestionamento. O último modo pelo qual a partida lenta pode terminar é se três ACKs duplicados forem detectados, caso no qual o TPC apresenta uma retransmissão rápida (veja Seção 3.5.4) e entra no estado de recuperação rápida, como discutido abaixo. O comportamento do TCP na partida lenta está resumido na descrição FSM do controle de congestionamento no TCP na Figura 3.52. O algoritmo de partida lenta foi inicialmente proposto por [Jacobson, 1998]; uma abordagem semelhante à partida lenta também foi proposta de maneira independente em [Jain, 1986].

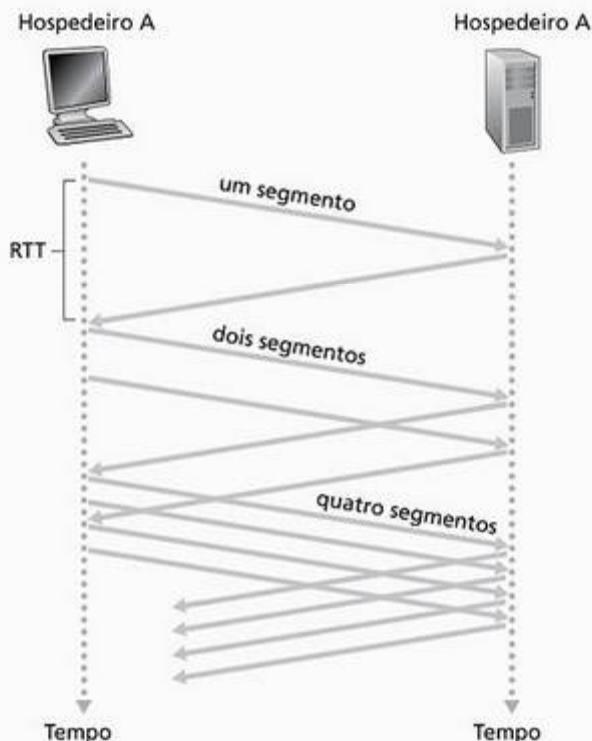


Figura 3.51 Partida lenta TCP

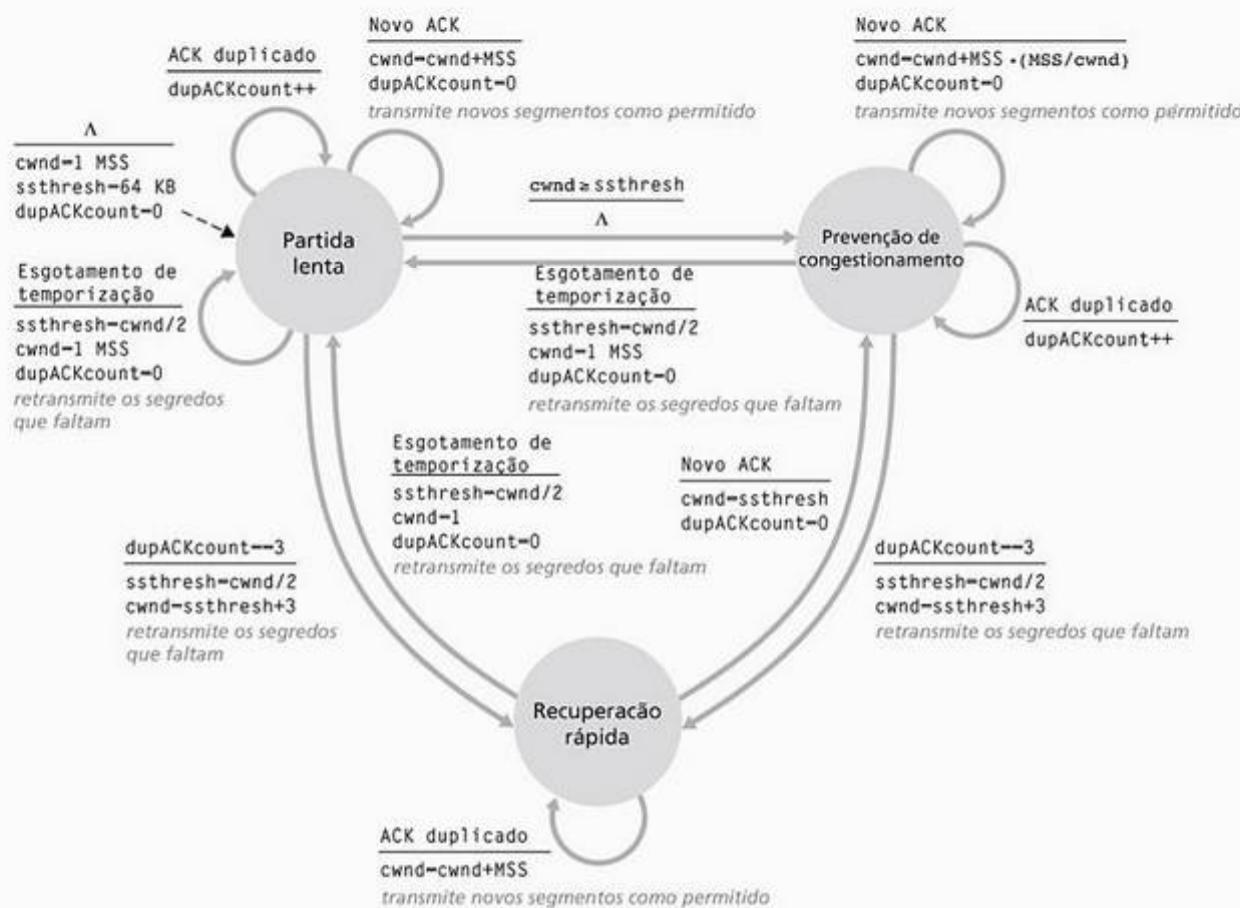


Figura 3.52 Descrição FSM do controle de congestionamento no TCP

Prevenção de congestionamento

Ao entrar no estado de prevenção de congestionamento, o valor de $cwnd$ é aproximadamente metade de seu valor quando o congestionamento foi encontrado pela última vez — o congestionamento poderia estar por perto! Desta forma, em vez de duplicar o valor de $cwnd$ a cada RTT, o TCP adota uma abordagem mais conservadora e aumenta o valor de $cwnd$ por meio de um único MSS a cada RTT [RFC 2581]. Isso pode ser realizado de diversas formas. Uma abordagem comum é o remetente aumentar $cwnd$ por MSS bytes ($MSS/cwnd$) no momento em que um novo reconhecimento chegar. Por exemplo, se o MSS possui 1.460 bytes e $cwnd$, 14.600 bytes, então 10 segmentos estão sendo enviados dentro de um RTT. Cada ACK que chega (considerando um ACK por segmento) aumenta o tamanho da janela de congestionamento em 1/10 MSS e, assim, o valor da janela de congestionamento terá aumentado para um MSS após os ACKs quando todos os segmentos tiverem sido recebidos.

Mas em que momento o aumento linear da prevenção de congestionamento (de 1 MSS por RTT) deve terminar? O algoritmo de prevenção de congestionamento TCP se comporta da mesma forma quando ocorre um esgotamento de temporização. Como no caso da partida lenta: o valor de $cwnd$ é ajustado para 1 MSS, e o valor de $ssthresh$ é atualizado para metade do valor de $cwnd$ quando ocorreu o evento de perda. Lembre-se, entretanto, de que um evento de perda também pode ser acionado por um evento ACK duplicado triplo. Neste caso, a rede continua a enviar segmentos do remetente ao destinatário (como indicado pelo recebimento de ACKs duplicados). Portanto, o comportamento do TCP para esse tipo de evento de perda deve ser menos drástico do que com uma perda de esgotamento de temporização: O TCP reduz o valor de $cwnd$ para metade (adicionando em 3 MSS a mais para contabilizar os ACKs duplicados triplos recebidos) e registra o valor de

$ssthresh$ como metade do de $cwnd$ quando os ACKs duplicados triplos foram recebidos. Então, entra-se no estado de recuperação rápida.

Recuperação rápida

Na recuperação rápida, o valor de $cwnd$ é aumentado por 1 MSS para cada ACK duplicado recebido no segmento perdido que fez com que o TCP entrasse no modo de recuperação rápida. Consequentemente, quando um ACK chega ao segmento perdido, o TCP entra no modo de prevenção de congestionamento após reduzir $cwnd$. Se um evento de esgotamento de temporização ocorrer, a recuperação rápida é alterada para o modo partida lenta após desempenhar as mesmas ações que a partida lenta e a prevenção de congestionamento: o valor de $cwnd$ é ajustado para 1 MSS, e o valor de $ssthresh$, para metade do valor de $cwnd$ no momento em que o evento de perda ocorreu.

A recuperação rápida é recomendada, mas não exigida, para o protocolo TCP [RFC 2581]. É interessante o fato de que uma antiga versão do TCP, conhecida como TCP Tahoe, reduz incondicionalmente sua janela de congestionamento para 1 MSS e entrou na fase de partida lenta após um evento de perda de esgotamento do temporizador ou de ACK duplicado triplo. A versão atual do TCP, a TCP Reno, incluiu a recuperação rápida.

A Figura 3.53 ilustra a evolução da janela de congestionamento do TCP para as versões Reno e Tahoe. Nessa figura, o limiar é, inicialmente, igual a 8 MSS. Nas primeiras oito sessões de transmissão, as duas versões possuem ações idênticas. A janela de congestionamento se eleva exponencialmente rápido durante a partida lenta e atinge o limiar na quarta sessão de transmissão. A janela de congestionamento, então, se eleva linearmente até que ocorra um evento ACK duplicado triplo, logo após a oitava sessão de transmissão. Observe que a janela de congestionamento é $12 \cdot MSS$ quando ocorre o evento de perda. O valor de $ssthresh$ é, então, ajustado para $0,5 \cdot cwnd = 6 \cdot MSS$. Sob o TCP Reno, a janela de congestionamento é ajustada para $cwnd = 6 \cdot MSS$, e depois cresce linearmente.

A Figura 3.52 apresenta a descrição FSM completa dos algoritmos de controle de congestionamento — partida lenta, prevenção de congestionamento e recuperação rápida. A figura também indica onde pode ocorrer transmissão de novos segmentos ou segmentos retransmitidos. Embora seja importante diferenciar controle/retransmissão de erro TCP de controle de congestionamento no TCP, também é importante avaliar como esses dois aspectos do TCP estão inseparavelmente ligados.

Controle de congestionamento no TCP: retrospectiva

Após nos aprofundarmos em detalhes sobre partida lenta, prevenção de congestionamento e recuperação rápida, vale a pena agora voltar e ver a floresta por entre as árvores. Desconsiderando o período inicial de partida lenta, quando uma conexão se inicia, e supondo que as perdas são indicadas por ACKs duplicados triplos e não por esgotamentos de temporização, o controle de congestionamento no TCP consiste em um aumento linear

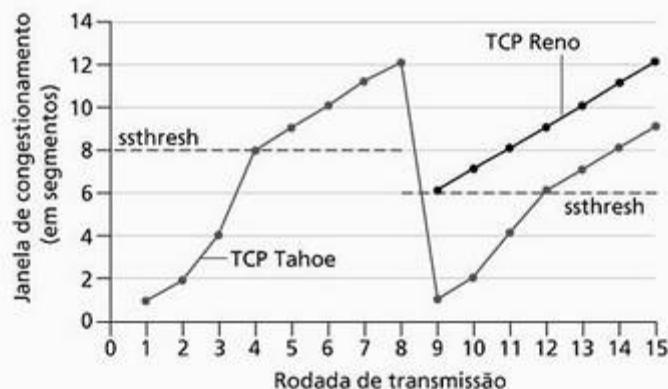


Figura 3.53 Evolução da janela de congestionamento do TCP (Tahoe e Reno)

(aditivo) em $cwnd$ de 1 MSS por RTT e, então, uma redução à metade (diminuição multiplicativa) de $cwnd$ em um evento ACK duplicado triplo. Por esta razão, o controle de congestionamento no TCP é frequentemente denominado aumento aditivo, diminuição multiplicativa (AIMD). O controle de congestionamento AIMD faz surgir o comportamento semelhante a “dentes de serra”, mostrado na Figura 3.54, a qual também ilustra de forma interessante nossa intuição anterior sobre a “busca” do TCP por largura de banda — o TCP aumenta linearmente o tamanho de sua janela de congestionamento (e, portanto, sua taxa de transmissão) até que um evento ACK duplicado triplo ocorra. Então, ele reduz o tamanho de sua janela por um fator de dois mas começa novamente a aumentá-la linearmente, buscando saber se há uma largura de banda adicional disponível.

Como observado anteriormente, a maioria das implementações TCP, atualmente, utiliza o algoritmo Reno [Padhye, 2001]. Foram propostas muitas variações do algoritmo Reno [RFC 3782; RFC 2018]. O algoritmo Vegas [Brakmo, 1995; Ahn, 1995] tenta evitar o congestionamento enquanto mantém uma boa vazão. A ideia básica de tal algoritmo é (1) detectar congestionamento nos roteadores entre a fonte e o destino *antes* que ocorra a perda de pacote e (2) diminuir linearmente a taxa ao ser detectada essa perda de pacote iminente, que pode ser prevista por meio da observação do RTT. Quanto maior for o RTT dos pacotes, maior será o congestionamento nos roteadores. O Linux suporta um número de algoritmos de controle de congestionamento (incluindo o TCP Reno e o TCP Vegas) e permite que um administrador de sistema configure qual versão do TCP será utilizada. A versão padrão do TCP no Linux versão 2.6.18 foi definida para CUBIC [Ha, 2008], uma versão do TCP desenvolvida para aplicações de alta largura de banda. O algoritmo AIMD do TCP foi desenvolvido com base na quantidade significativa de percepção de engenharia e experimentação com controle de congestionamento em redes operacionais. Dez anos após a criação do TCP, análises teóricas mostraram que o algoritmo de controle de congestionamento do TCP serve como um algoritmo de otimização assíncrona distribuída que resulta em diversos aspectos importantes sobre usuário e desempenho da rede sendo otimizados simultaneamente [Kelly, 1998]. Uma teoria aprimorada sobre controle de congestionamento tem sido desenvolvida desde então [Srikant, 2004].

Descrição macroscópica da dinâmica do TCP

Dado o comportamento semelhante a dentes de serra do TCP, é natural considerar qual seria a vazão média (isto é, a taxa média) de uma conexão TCP ativa há longo tempo. Nessa análise, vamos ignorar as fases de partida lenta que ocorrem após eventos de esgotamento de temporização. (Essas fases normalmente são muito curtas, visto que o remetente sai delas com rapidez exponencial.) Durante um determinado intervalo de viagem de ida e volta, a taxa à qual o TCP envia dados é uma função da janela de congestionamento e do RTT corrente. Quando o tamanho da janela for w bytes, e o tempo de viagem de ida e volta corrente for RTT segundos, a taxa de transmissão do TCP será aproximadamente w/RTT . Então, o TCP faz uma sondagem em busca de alguma largura de

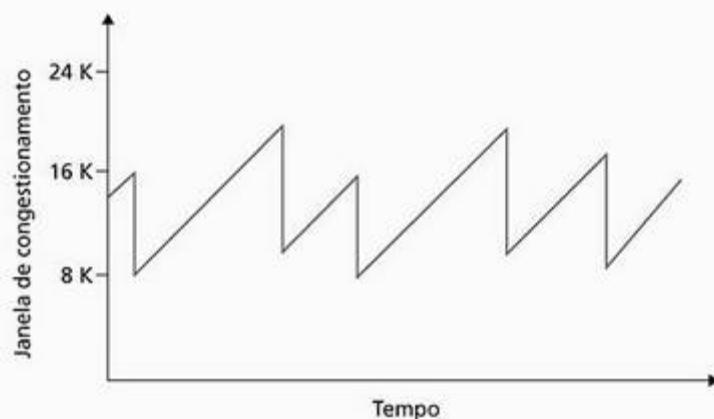


Figura 3.54 Controle de congestionamento por aumento aditivo, diminuição multiplicativa

banda adicional aumentando w de 1 MSS a cada RTT até ocorrer um evento de perda. Seja W o valor de w quando ocorre um evento de perda. Admitindo que RTT e W são aproximadamente constantes durante o período da conexão, a taxa de transmissão fica na faixa de $W/(2 \cdot RTT)$ a W/RTT .

Essas suposições levam a um modelo macroscópico muito simplificado para o comportamento do TCP em regime. A rede descarta um pacote da conexão quando a taxa aumenta para W/RTT ; então a taxa é reduzida à metade e, em seguida, aumenta de um MSS/RTT a cada RTT até alcançar W/RTT novamente. Esse processo se repete continuamente. Como a vazão do TCP (isto é, a taxa) aumenta linearmente entre os dois valores extremos, temos:

$$\text{Vazão média de uma conexão} = \frac{0,75 \cdot W}{RTT}$$

Usando esse modelo muito idealizado para a dinâmica de regime permanente do TCP, podemos também derivar uma interessante expressão que relaciona a taxa de perda de uma conexão com sua largura de banda disponível [Mahdavi, 1997]. Essa derivação está delineada nos exercícios de fixação. Um modelo mais sofisticado que demonstrou empiricamente estar de acordo com dados medidos é apresentado em [Padhye, 2000].

TCPs futuros

É importante perceber que o controle de congestionamento no TCP evoluiu ao longo dos anos e, na verdade, continua a evoluir. Um resumo do controle de congestionamento no TCP no final da década de 1990 pode ser encontrado no [RFC 2581]; se quiser uma discussão sobre desenvolvimentos recentes do controle de congestionamento no TCP, veja [Floyd, 2001]. O que era bom para a Internet quando a maior parte das conexões TCP carregava tráfego SMTP, FTP e Telnet não é necessariamente bom para a Internet de hoje, dominada pelo HTTP, ou para uma Internet futura, com serviços que ainda nem sonhamos.

A necessidade da evolução contínua do TCP pode ser ilustrada considerando as conexões TCP de alta velocidade que são necessárias para aplicações de computação em grade [Foster, 2002]. Por exemplo, considere uma conexão TCP com segmentos de 1.500 bytes e RTT de 100 ms, e suponha que queremos enviar dados por essa conexão a 10 Gbps. Seguindo o [RFC 3649] e utilizando a fórmula de vazão do TCP apresentada anteriormente, notamos que, para alcançar uma vazão de 10 Gbps, o tamanho médio da janela de congestionamento precisaria ser 83.333 segmentos. São muitos segmentos, e pensar que um deles poderia ser perdido em trânsito nos deixa bastante preocupados. O que aconteceria no caso de uma perda? Ou, em outras palavras, que fração dos segmentos transmitidos poderia ser perdida e ainda assim permitisse que o algoritmo de controle de congestionamento no TCP delineado na Tabela 3.52 alcançasse a desejada taxa de 10 Gbps? Nos exercícios de fixação deste capítulo você percorrerá a dedução de uma fórmula que relaciona a vazão de uma conexão TCP em função da taxa de perda (L), do tempo de viagem de ida e volta (RTT) e do tamanho máximo de segmento (MSS):

$$\text{Vazão média de uma conexão} = \frac{1,22 \cdot MSS}{RTT \sqrt{L}}$$

Usando essa fórmula, podemos ver que, para alcançar uma vazão de 10 Gbps, o algoritmo de controle de congestionamento no TCP de hoje pode tolerar uma probabilidade de perda de segmentos de apenas $2 \cdot 10^{-10}$ (equivalente a um evento de perda a cada 5 bilhões de segmentos) — uma taxa muito baixa. Essa observação levou muitos pesquisadores a investigar novas versões do TCP especificamente projetadas para esses ambientes de alta velocidade; [Jin, 2004; RFC 3649; Kelly, 2003; Ha, 2008] apresentam discussões sobre esses esforços.

3.7.1 Equidade

Considere K conexões TCP, cada uma com um caminho fim a fim diferente, mas todas passando pelo gargalo em um enlace com taxa de transmissão de R bps (aqui, *gargalo em um enlace* quer dizer que nenhum dos outros enlaces ao longo do caminho de cada conexão está congestionado e que todos dispõem de abundante capacidade de transmissão em comparação à capacidade de transmissão do enlace com gargalo). Suponha que cada conexão

está transferindo um arquivo grande e que não há tráfego UDP passando pelo enlace com gargalo. Dizemos que um mecanismo de controle de congestionamento é *justo* se a taxa média de transmissão de cada conexão for aproximadamente R/K ; isto é, cada uma obtém uma parcela igual da largura de banda do enlace.

O algoritmo AIMD do TCP é justo, considerando, em especial, que diferentes conexões TCP podem começar em momentos diferentes e, assim, ter tamanhos de janela diferentes em um dado instante? [Chiu, 1989] explica, de um modo elegante e intuitivo, por que o controle de congestionamento converge para fornecer um compartilhamento justo da largura de banda do enlace entre conexões TCP concorrentes.

Vamos considerar o caso simples de duas conexões TCP compartilhando um único enlace com taxa de transmissão R , como mostra a Figura 3.55. Vamos admitir que as duas conexões tenham os mesmos MSS e RTT (de modo que, se o tamanho de suas janelas de congestionamento for o mesmo, então terão a mesma vazão) e uma grande quantidade de dados para enviar e que nenhuma outra conexão TCP ou datagramas UDP atravesse esse enlace compartilhado. Vamos ignorar também a fase de partida lenta do TCP e admitir que as conexões TCP estão operando em modo prevenção de congestão (AIMD) todo o tempo.

A Figura 3.56 mostra a vazão alcançada pelas duas conexões TCP. Se for para o TCP compartilhar equitativamente a largura de banda do enlace entre as duas conexões, então a vazão alcançada deverá cair ao longo da linha a 45 graus (igual compartilhamento da largura de banda) que parte da origem. Idealmente, a soma das duas vazões seria igual a R . (Com certeza, não é uma situação desejável cada conexão receber um compartilhamento igual, mas igual a zero, da capacidade do enlace!) Portanto, o objetivo é que as vazões alcançadas fiquem em algum lugar perto da intersecção da linha de igual compartilhamento da largura de banda com a linha de utilização total da largura de banda da Figura 3.56.

Suponha que os tamanhos de janela TCP sejam tais que, em um determinado instante, as conexões 1 e 2 alcancem as vazões indicadas pelo ponto A na Figura 3.56. Como a quantidade de largura de banda do enlace consumida conjuntamente pelas duas conexões é menor do que R , não ocorrerá nenhuma perda e ambas as conexões aumentarão suas janelas de 1 por RTT como resultado do algoritmo de prevenção de congestionamento do TCP. Assim, a vazão conjunta das duas conexões continua ao longo da linha a 45 graus (aumento igual para as duas), começando no ponto A. Finalmente, a largura de banda do enlace consumida em conjunto pelas duas conexões será maior do que R e, assim, também fatalmente, ocorrerá perda de pacote. Suponha que as conexões 1 e 2 experimentem perda de pacote quando alcançarem as vazões indicadas pelo ponto B. As conexões 1 e 2 então reduzirão suas janelas por um fator de 2. Assim, as vazões resultantes são as do ponto C, a meio caminho do vetor que começa em B e termina na origem. Como a utilização conjunta da largura de banda é menor do que R no ponto C, as duas conexões novamente aumentam suas vazões ao longo da linha a 45 graus que começa no ponto C. Mais cedo ou mais tarde ocorrerá perda, por exemplo, no ponto D, e as duas conexões novamente reduzirão o tamanho de suas janelas por um fator de 2 — e assim por diante. Você pode ter certeza de que a largura de banda alcançada pelas duas conexões flutuará ao longo da linha de igual compartilhamento da largura de banda. E também estar certo de que as duas conexões convergirão para esse comportamento, não importando onde elas comecem no espaço bidimensional! Embora haja uma série de suposições idealizadas por trás desse cenário, ainda assim ele dá uma ideia intuitiva de por que o TCP resulta em igual compartilhamento da largura de banda entre conexões.

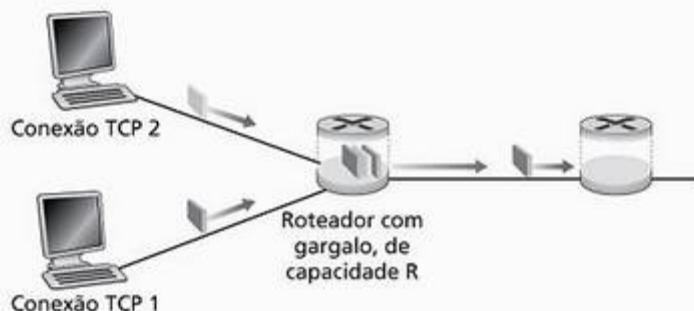


Figura 3.55 Duas conexões TCP compartilhando um único enlace congestionado

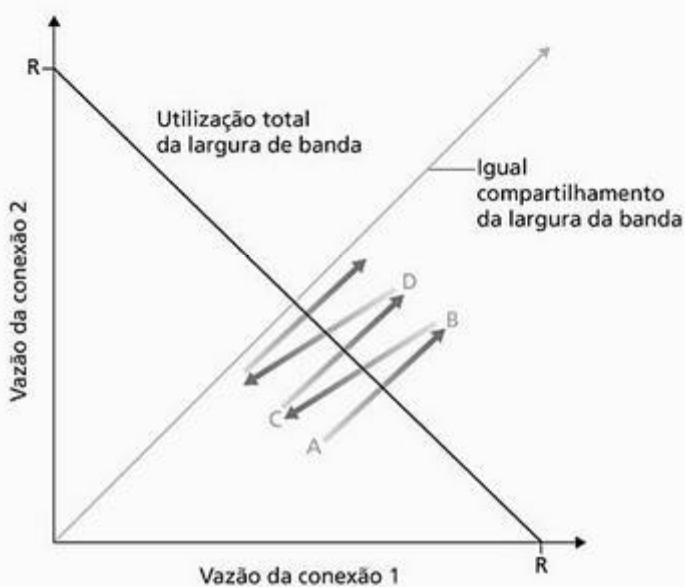


Figura 3.56 Vazão alcançada pelas conexões TCP 1 e TCP 2

Em nosso cenário idealizado, admitimos que somente conexões TCP atravessem o enlace com gargalo, que as conexões tenham o mesmo valor de RTT e que uma única conexão TCP esteja associada com um par hospedeiro/destinatário. Na prática, essas condições não são comumente encontradas e, assim, é possível que as aplicações cliente-servidor obtenham porções muito desiguais da largura de banda do enlace. Em especial, foi demonstrado que, quando várias conexões compartilham um único enlace com gargalo, as sessões cujos RTTs são menores conseguem obter a largura de banda disponível naquele enlace mais rapidamente (isto é, abre suas janelas de congestionamento mais rapidamente) à medida que o enlace fica livre. Assim, conseguem vazões mais altas do que conexões com RTTs maiores [Lakshman, 1997].

Equidade e UDP

Acabamos de ver como o controle de congestionamento no TCP regula a taxa de transmissão de uma aplicação por meio do mecanismo de janela de congestionamento. Muitas aplicações de multimídia, como telefone por Internet e videoconferência, muitas vezes não rodam sobre TCP exatamente por essa razão — elas não querem que sua taxa de transmissão seja limitada, mesmo que a rede esteja muito congestionada. Ao contrário, preferem rodar sobre UDP, que não tem controle de congestionamento. Quando rodam sobre esse protocolo, as aplicações podem passar seus áudios e vídeos para a rede a uma taxa constante e, ocasionalmente, perder pacotes, em vez de reduzir suas taxas a níveis ‘justos’ em horários de congestionamento e não perder nenhum deles. Da perspectiva do TCP, as aplicações de multimídia que rodam sobre UDP não são justas — elas não cooperam com as outras conexões nem ajustam suas taxas de transmissão de maneira adequada. Como o controle de congestionamento no TCP reduzirá sua taxa de transmissão quando houver aumento de congestionamento (perda), enquanto fontes UDP não precisam fazer o mesmo, é possível que essas fontes desalojem o tráfego TCP. Uma área importante da pesquisa hoje é o desenvolvimento de mecanismos de controle de congestionamento que impeçam que o tráfego de UDP leve a vazão da Internet a uma parada repentina [Floyd, 1999; Floyd, 2000; Kohler, 2006].

Equidade e conexões TCP paralelas

Mas, mesmo que pudéssemos obrigar o tráfego UDP a se comportar com equidade, o problema ainda não estaria completamente resolvido. Isso porque não há nada que impeça uma aplicação de rodar sobre TCP usando múltiplas conexões paralelas. Por exemplo, browsers Web frequentemente usam múltiplas conexões

TCP paralelas para transferir os vários objetos de uma página Web. (O número exato de conexões múltiplas pode ser configurado na maioria dos browsers.) Quando usa múltiplas conexões paralelas, uma aplicação consegue uma fração maior da largura de banda de um enlace congestionado. Como exemplo, considere um enlace de taxa R que está suportando nove aplicações cliente-servidor em curso, e cada uma das aplicações está usando uma conexão TCP. Se surgir uma nova aplicação que também utilize uma conexão TCP, então cada aplicação conseguirá aproximadamente a mesma taxa de transmissão igual a $R/10$. Porém, se, em vez disso, essa nova aplicação usar 11 conexões TCP paralelas, então ela conseguirá uma alocação injusta de mais do que $R/2$. Como a penetração do tráfego Web na Internet é grande, as múltiplas conexões paralelas não são incomuns.

3.8 Resumo

Começamos este capítulo estudando os serviços que um protocolo de camada de transporte pode prover às aplicações de rede. Por um lado, o protocolo de camada de transporte pode ser muito simples e oferecer serviços básicos às aplicações, provendo apenas uma função de multiplexação/demultiplexação para processos comunicantes. O protocolo UDP da Internet é um exemplo desse serviço básico de um protocolo de camada de transporte. Por outro lado, um protocolo de camada de transporte pode fornecer uma variedade de garantias às aplicações, como entrega confiável de dados, garantias contra atrasos e garantias de largura de banda. Não obstante, os serviços que um protocolo de transporte pode prover são frequentemente limitados pelo modelo de serviço do protocolo subjacente de camada de rede. Se o protocolo de camada de rede não puder proporcionar garantias contra atraso ou garantias de largura de banda para segmentos da camada de transporte, então o protocolo de camada de transporte não poderá fornecer essas garantias para as mensagens enviadas entre processos.

Aprendemos na Seção 3.4 que um protocolo de camada de transporte pode prover transferência confiável de dados mesmo que a camada de rede subjacente seja não confiável. Vimos que há muitos pontos sutis na transferência confiável de dados, mas que a tarefa pode ser realizada pela combinação cuidadosa de reconhecimentos, temporizadores, retransmissões e números de sequência.

Embora tenhamos examinado a transferência confiável de dados neste capítulo, devemos ter em mente que essa transferência pode ser fornecida por protocolos de camada de enlace, de rede, de transporte ou de aplicação. Qualquer uma das camadas superiores da pilha de protocolos pode implementar reconhecimentos, temporizadores, retransmissões e números de sequência e prover transferência confiável de dados para a camada situada acima dela. Na verdade, com o passar dos anos, engenheiros e cientistas da computação projetaram e implementaram, independentemente, protocolos de camada de enlace, de rede, de transporte e de aplicação que fornecem transferência confiável de dados (embora muitos desses protocolos tenham desaparecido silenciosamente).

Na Seção 3.5, examinamos detalhadamente o TCP, o protocolo de camada de transporte confiável orientado para conexão da Internet. Aprendemos que o TCP é complexo e envolve a conexão, controle de fluxo, estimativa de tempo de viagem de ida e volta, bem como transferência confiável de dados. Na verdade, o TCP é mais complexo do que nossa descrição — intencionalmente, não discutimos uma variedade de ajustes, acertos e melhorias que estão implementados em várias versões do TCP. Toda essa complexidade, no entanto, fica escondida da aplicação de rede. Se um cliente em um hospedeiro quiser enviar dados de maneira confiável para outro hospedeiro, ele simplesmente abre uma porta TCP para o servidor e passa dados para dentro dessa porta. A aplicação cliente-servidor felizmente fica alheia a toda a complexidade do TCP.

Na Seção 3.6, examinamos o controle de congestionamento de uma perspectiva mais ampla e, na Seção 3.7, demonstramos como o TCP implementa controle de congestionamento. Aprendemos que o controle de congestionamento é imperativo para o bem-estar da rede. Sem ele, uma rede pode facilmente ficar travada, com pouco ou nenhum dado sendo transportado fim a fim. Na Seção 3.7, aprendemos também que o TCP implementa um mecanismo de controle de congestionamento fim a fim que aumenta aditivamente sua taxa de transmissão quan-

do julga que o caminho da conexão TCP está livre de congestionamento e reduz multiplicativamente sua taxa de transmissão quando ocorre perda. Esse mecanismo também luta para dar a cada conexão TCP que passa por um enlace congestionado uma parcela igual da largura de banda da conexão. Examinamos ainda com um certo detalhe o impacto do estabelecimento da conexão TCP e da partida lenta sobre a latência. Observamos que, em muitos cenários importantes, o estabelecimento da conexão e a partida lenta contribuem de modo significativo para o atraso fim a fim. Enfatizamos mais uma vez que, embora tenha evoluído com o passar dos anos, o controle de congestionamento no TCP permanece como uma área de pesquisa intensa e, provavelmente, continuará a evoluir nos anos vindouros.

Nossa discussão sobre os protocolos específicos de transporte da Internet neste capítulo se focalizou no UDP e no TCP — os dois “burros de carga” da camada de transporte da Internet. Entretanto, duas décadas de experiência com esses dois protocolos identificaram circunstâncias nas quais nenhum deles é apropriado de maneira ideal. Desse modo, pesquisadores se ocuparam em desenvolver protocolos da camada de transporte adicionais, muitos dos quais são, agora, padrões propostos pelo IETF.

O Protocolo de Controle de Congestionamento de Datagrama (DCCP) [RFC4340] oferece um serviço de baixo consumo, orientado a mensagem e não confiável de forma semelhante ao UDP, mas com uma forma selecionada de aplicação de controle de congestionamento que é compatível com o TCP. Caso uma aplicação necessitar de uma transferência de dados confiável ou semiconfiável, então isso seria realizado dentro da própria aplicação, talvez utilizando os mecanismos que estudamos na Seção 3.4. O DCCP é utilizado em aplicações, como o fluxo de mídia (veja Capítulo 7), que podem se aproveitar da escolha entre conveniência e confiança do fornecimento de dados, mas que querem ser sensíveis ao congestionamento da rede.

O Protocolo de Controle de Fluxo de Transmissão (SCTP) [RFC 2960, RFC 3286] é um protocolo confiável e orientado a mensagens que permite que diferentes “fluxos” de aplicação sejam multiplexados através de uma única conexão SCTP (método conhecido como “múltiplos fluxos”). De um ponto de vista confiável, os diferentes fluxos dentro da conexão são controlados separadamente, de modo que uma perda de pacote em um fluxo não afete o fornecimento de dados em outros fluxos. O SCTP também permite que os dados sejam transferidos por meio de dois percursos de saída quando um hospedeiro está conectado a duas ou mais redes, fornecimento opcional de dados inadequados, e muitos outros recursos. Os algoritmos de controle de congestionamento e de fluxo do SCTP são basicamente os mesmos do TCP.

O protocolo TRFC (TCP — *Friendly Rate Control*) [RFC 5348] é mais um protocolo de controle de congestionamento do que um protocolo da camada de transporte completo. Ele especifica um mecanismo de controle de congestionamento que poderia ser utilizado em outro protocolo de transporte como o DCCP (de fato, um dos dois protocolos de aplicação disponível no DCCP é o TRFC). O objetivo do TRFC é estabilizar o comportamento dos “dentes de serra” (veja Figura 3.54) no controle de congestionamento no TCP, enquanto mantém uma taxa de envio de longo prazo “razoavelmente” próxima à do TCP. Com uma taxa de envio mais estável do que a do TCP, o TRFC é adequado às aplicações multimídia, como telefonia IP ou fluxo de mídia, em que uma taxa estável é importante. O TRFC é um protocolo baseado em equação que utiliza a taxa de perda de pacote calculada como suporte a uma equação [Padhye, 2000] que estima qual seria a vazão do TCP se uma sessão TCP sofrer taxa de perda. Essa taxa é então adotada como a taxa-alvo de envio do TRFC.

Somente o futuro dirá se o DCCP, o SCTP ou o TRFC possuirá uma implementação difundida. Enquanto esses protocolos fornecem claramente recursos avançados sobre o TCP e o UDP, estes já provaram a si mesmos ser “bons o bastante” com o passar dos anos. A vitória do “melhor” sobre o “bom o bastante” dependerá de uma mistura complexa de considerações técnicas, sociais e comerciais.

No Capítulo 1, dissemos que uma rede de computadores pode ser dividida em ‘periferia da rede’ e ‘núcleo da rede’. A periferia da rede abrange tudo o que acontece nos sistemas finais. Com o exame da camada de aplicação e da camada de transporte, nossa discussão sobre a periferia da rede está completa. É hora de explorar o núcleo da rede! Essa jornada começa no próximo capítulo, em que estudaremos a camada de rede, e continua no Capítulo 5, em que estudaremos a camada de enlace.



Exercícios de fixação

Capítulo 3 Questões de revisão

Seções 3.1 a 3.3

1. Suponha que uma camada de rede forneça o seguinte serviço. A camada de rede no computador-fonte aceita um segmento de tamanho máximo de 1.200 bytes e um endereço de computador-alvo da camada de transporte. Esta, então, garante encaminhar o segmento para a camada de transporte no computador-alvo. Suponha que muitos processos de aplicação de rede possam estar sendo executados no computador-alvo.
 - a. Crie, da forma mais simples, o protocolo da camada de transporte possível que levará os dados da aplicação para o processo desejado no computador-alvo. Suponha que o sistema operacional do computador-alvo determinou um número de porta de 4 bytes para cada processo de aplicação em execução.
 - b. Modifique esse protocolo de modo que ele forneça um "endereço de retorno" para o processo-alvo.
 - c. Em seus protocolos, a camada de transporte "tem de fazer algo" no núcleo da rede de computadores?
2. Considere um planeta onde todos possuam uma família com seis membros, cada família viva em sua própria casa, cada casa possua um endereço único e cada pessoa em certa casa possua um único nome. Suponha que esse planeta possua um serviço postal que entregue cartas da casa-fonte à casa-alvo. O serviço exige que (i) a carta esteja em um envelope e que (ii) o endereço da casa-alvo (e nada mais) esteja escrito claramente no envelope. Suponha que cada família possua um membro representante que recebe e distribui cartas para outros membros da família. As cartas não apresentam necessariamente qualquer indicação dos destinatários das cartas.
 - a. Utilizando a solução do Problema 1 como inspiração, descreva um protocolo que os representantes possam utilizar para entregar cartas de um membro remetente de uma família para um membro destinatário de outra família.
 - b. Em seu protocolo, o serviço postal precisa abrir o envelope e verificar a carta para fornecer o serviço?
3. Considere uma conexão TCP entre o hospedeiro A e o hospedeiro B. Suponha que os segmentos TCP que trafegam do hospedeiro A para o hospedeiro B tenham número de porta da fonte x e

número de porta do destino y . Quais são os números de porta da fonte e do destino para os segmentos que trafegam do hospedeiro B para o hospedeiro A?

4. Descreva por que um desenvolvedor de aplicação pode escolher rodar uma aplicação sobre UDP em vez de sobre TCP.
5. Por que o tráfego de voz e de vídeo é frequentemente enviado por meio do UDP e não do TCP na Internet de hoje? (Dica: A resposta que procuramos não tem nenhuma relação com o mecanismo de controle de congestionamento no TCP.)
6. É possível que uma aplicação desfrute de transferência confiável de dados mesmo quando roda sobre UDP? Caso a resposta seja afirmativa, como isso acontece?
7. Suponha que um processo no Computador C possua um socket UDP com número de porta 6789 e que o Computador A e o Computador B, individualmente, enviem um segmento UDP ao Computador C com número de porta de destino 6789. Esses dois segmentos serão encaminhados para o mesmo socket no Computador C? Se sim, como o processo no Computador C saberá que esses dois segmentos vieram de dois computadores diferentes?
8. Suponha que um servidor da Web seja executado no Computador C na porta 80. Esse servidor utiliza conexões contínuas e, no momento, está recebendo solicitações de dois Computadores diferentes, A e B. Todas as solicitações estão sendo enviadas através do mesmo socket no Computador C? Se eles estão passando por diferentes sockets, dois desses sockets possuem porta 80? Discuta e explique.

Seção 3.4

9. Nos nossos protocolos rdt, por que precisamos introduzir números de sequência?
10. Nos nossos protocolos rdt, por que precisamos introduzir temporizadores?
11. Suponha que o atraso de viagem de ida e volta entre o emissor e o receptor seja constante e conhecido para o emissor. Ainda seria necessário um temporizador no protocolo rdt 3.0, supondo que os pacotes podem ser perdidos? Explique.
12. Visite o applet Go-Back-N Java no Companion siteWeb.

- a. A fonte enviou cinco pacotes e depois interrompeu a animação antes que qualquer um dos cinco pacotes chegassem ao destino. Então, elimine o primeiro pacote e reinicie a animação. Descreva o que acontece.
 - b. Repita o experimento, mas agora deixe o primeiro pacote chegar ao destino e elimine o primeiro reconhecimento. Descreva novamente o que acontece.
 - c. Por fim, tente enviar seis pacotes. O que acontece?
13. Repita a questão 12, mas agora com o applet *Selective Repeat Java*. O que difere o *Selective Repeat* do Go-Back-N?

Seção 3.5

14. Verdadeiro ou falso:

- a. O hospedeiro A está enviando ao hospedeiro B um arquivo grande por uma conexão TCP. Suponha que o hospedeiro B não tenha dados para enviar para o hospedeiro A. O hospedeiro B não enviará reconhecimentos para o hospedeiro A porque ele não pode dar carona aos reconhecimentos dos dados.
- b. O tamanho de *rwnd* do TCP nunca muda enquanto dura a conexão.
- c. Suponha que o hospedeiro A esteja enviando ao hospedeiro B um arquivo grande por uma conexão TCP. O número de bytes não reconhecidos que o hospedeiro A envia não pode exceder o tamanho do buffer de recepção.
- d. Imagine que o hospedeiro A esteja enviando ao hospedeiro B um arquivo grande por uma conexão TCP. Se o número de sequência para um segmento dessa conexão for m , então o número de sequência para o segmento subsequente será necessariamente $m + 1$.
- e. O segmento TCP tem um campo em seu cabeçalho para *Rwnd*.
- f. Suponha que o último *SampleRTT* de uma conexão TCP seja igual a 1 segundo. Então, o

valor corrente de *TimeoutInterval* para a conexão será necessariamente ajustado para um valor ≥ 1 segundo.

- g. Imagine que o hospedeiro A envie ao hospedeiro B, por uma conexão TCP, um segmento com o número de sequência 38 e 4 bytes de dados. Nesse mesmo segmento, o número de reconhecimento será necessariamente 42.
15. Suponha que o hospedeiro A envie dois segmentos TCP um atrás do outro ao hospedeiro B sobre uma conexão TCP. O primeiro segmento tem número de sequência 90 e o segundo, número de sequência 110.
- a. Quantos dados tem o primeiro segmento?
 - b. Suponha que o primeiro segmento seja perdido, mas o segundo chegue a B. No reconhecimento que B envia a A, qual será o número de reconhecimento?
16. Considere o exemplo do Telnet discutido na Seção 3.5. Alguns segundos após o usuário digitar a letra 'C', ele digitará a letra 'R'. Depois disso, quantos segmentos serão enviados e o que será colocado nos campos de número de sequência e de reconhecimento dos segmentos?

Seção 3.7

17. Suponha que duas conexões TCP estejam presentes em algum enlace congestionado de velocidade R bps. Ambas as conexões têm um arquivo imenso para enviar (na mesma direção, pelo enlace congestionado). As transmissões dos arquivos começam exatamente ao mesmo tempo. Qual é a velocidade de transmissão que o TCP gostaria de dar a cada uma das conexões?
18. Verdadeiro ou falso: considere o controle de congestionamento no TCP. Quando um temporizador expira no remetente, o "valor de *ssthresh*" é ajustado para a metade de seu valor anterior.



Problemas

1. Suponha que o cliente A inicie uma sessão Telnet com o servidor S. Quase ao mesmo tempo, o cliente B também inicia uma sessão Telnet com o servidor S. Forneça possíveis números de porta da fonte e do destino para:
 - a. Os segmentos enviados de A para S.
 - b. Os segmentos enviados de B para S.
- c. Os segmentos enviados de S para A.
- d. Os segmentos enviados de S para B.
- e. Se A e B são hospedeiros diferentes, é possível que o número de porta da fonte nos segmentos de A para S seja o mesmo que nos de B para S?
- f. E se forem o mesmo hospedeiro?

2. Considere a Figura 3.5. Quais são os valores da porta de fonte e da porta de destino nos segmentos que fluem do servidor de volta aos processos clientes? Quais são os endereços IP nos datagramas de camada de rede que carregam os segmentos de camada de transporte?
3. O UDP e o TCP usam complementos de 1 para suas somas de verificação. Suponha que você tenha as seguintes três palavras de 8 bits: 01010011, 01010100 e 01110100. Qual é o complemento de 1 para as somas dessas palavras? (Note que, embora o UDP e o TCP usem palavras de 16 bits no cálculo da soma de verificação, nesse problema solicitamos que você considere somas de 8 bits). Mostre todo o trabalho. Por que o UDP toma o complemento de 1 da soma, isto é, por que não toma apenas a soma? Com o esquema de complemento de 1, como o destinatário detecta erros? É possível que um erro de 1 bit passe despercebido? E um erro de 2 bits?
4. a. Suponha que você tenha os seguintes bytes: 01011100 e 01010110. Qual é o complemento de da soma desses 2 bytes?
 b. Suponha que você tenha os seguintes bytes: 11011010 e 00110110. Qual é o complemento de 1 da soma desses 2 bytes?
 c. Para os bytes do item (a), dê um exemplo em que um bit é invertido em cada um dos 2 bytes e, mesmo assim, o complemento de um não muda.
5. Suponha que o receptor UDP calcule a soma de verificação da Internet para o segmento UDP recebido e encontre que essa soma coincide com o valor transportado no campo da soma de verificação. O receptor pode estar absolutamente certo de que não ocorreu nenhum erro de bit? Explique.
6. Considere nosso motivo para corrigir o protocolo rdt2.1. Demonstre que o destinatário apresentado na figura 3.57, quando em operação com o remetente mostrado na Figura 3.11, pode levar remetente e destinatário a entrar em estado de travamento, em que cada um espera por um evento que nunca vai ocorrer.
7. No protocolo rdt3.0, os pacotes ACK que fluem do destinatário ao remetente não têm números de sequência (embora tenham um campo de ACK que contém o número de sequência do pacote que estão reconhecendo). Por que nossos pacotes ACK não requerem números de sequência?
8. Elabore a FSM para o lado destinatário do protocolo rdt3.0.
9. Elabore um diagrama de mensagens para a operação do protocolo rdt3.0 quando pacotes de dados e de reconhecimento estão truncados. Seu diagrama deve ser semelhante ao usado na Figura 3.16.
10. Considere um canal que pode perder pacotes, mas cujo atraso máximo é conhecido. Modifique o protocolo rdt2.1 para incluir esgotamento de temporização do remetente e retransmissão. Informalmente, argumente por que seu protocolo pode se comunicar de modo correto por esse canal.

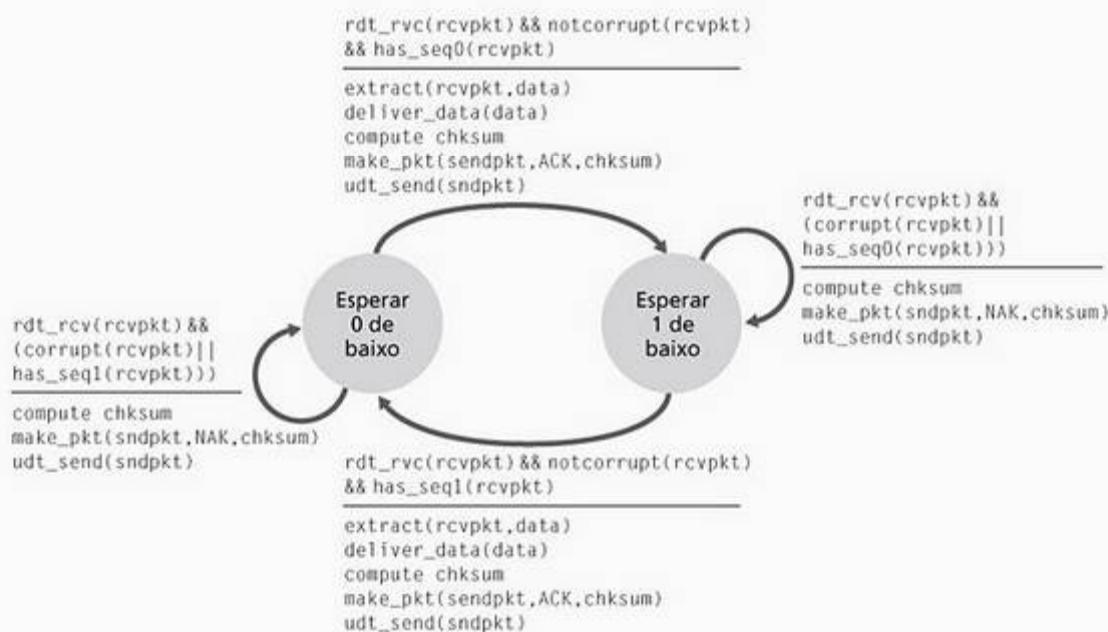


Figura 3.57 Um receptor incorreto para o protocolo rdt 2.1

11. O lado remetente do rdt3.0 simplesmente ignora (isto é, não realiza nenhuma ação) todos os pacotes recebidos que estão errados ou com o valor errado no campo acknum de um pacote de reconhecimento. Suponha que em tais circunstâncias o rdt3.0 devesse apenas retransmitir o pacote de dados corrente. Nesse caso, o protocolo ainda funcionaria? (Dica: considere o que aconteceria se houvesse apenas erros de bits; não há perdas de pacotes, mas podem ocorrer esgotamentos de temporização prematuros. Imagine quantas vezes o *enésimo* pacote é enviado, no limite em que n se aproximasse do infinito.)
12. Considere o protocolo rdt3.0. Elabore um diagrama mostrando que, se a conexão de rede entre o remetente e o destinatário puder alterar a ordem de mensagens (isto é, se for possível reordenar duas mensagens que se propagam no meio entre o remetente e o destinatário), então o protocolo bit alternante não funcionará corretamente (lembre-se de identificar claramente o sentido no qual o protocolo não funcionará corretamente). Seu diagrama deve mostrar o remetente à esquerda e o destinatário à direita; o eixo do tempo deverá estar orientado de cima para baixo na página e mostrar a troca de mensagem de dados (D) e de reconhecimento (A). Não esqueça de indicar o número de sequência associado com qualquer segmento de dados ou de reconhecimento.
13. Considere um protocolo de transferência confiável de dados que use somente reconhecimentos negativos. Suponha que o remetente envie dados com pouca frequência. Um protocolo que utiliza somente NAKs seria preferível a um protocolo que utiliza ACKs? Por quê? Agora suponha que o remetente tenha uma grande quantidade de dados para enviar e que a conexão fim a fim sofra poucas perdas. Nesse segundo caso, um protocolo que utilize somente NAKs seria preferível a um protocolo que utilize ACKs? Por quê?
14. Considere o exemplo em que se atravessa os Estados Unidos mostrado na Figura 3.17. Que tamanho deveria ter a janela para que a utilização do canal fosse maior do que 95 por cento? Suponha que o tamanho de um pacote seja 1.500 bytes, incluindo os campos do cabeçalho e os dados.
15. Suponha que uma aplicação utilize rdt3.0 como seu protocolo da camada de transporte. Como o protocolo pare e espere possui uma utilização do canal muito baixa (mostrada no exemplo de travessia dos Estados Unidos), os criadores dessa aplicação permitem que o receptor continue enviando de volta um número (mais do que dois) de ACK 0 alternado e ACK 1, mesmo que os dados correspondentes não chegarem ao receptor. O projeto dessa aplicação aumentaria a utilização do canal? Por quê? Há possíveis problemas com esse método? Explique.
16. No protocolo genérico SR que estudamos na Seção 3.4.4, o remetente transmite uma mensagem assim que ela está disponível (se ela estiver na janela), sem esperar por um reconhecimento. Suponha, agora, que queiramos um protocolo SR que envie duas mensagens de cada vez. Isto é, o remetente enviará um par de mensagens, e o par de mensagens subsequente somente deverá ser enviado quando o remetente souber que ambas as mensagens do primeiro par foram recebidas corretamente. Suponha que o canal possa perder mensagens, mas que não as corromperá nem as reordenará. Elabore um protocolo de controle de erro para a transferência confiável unidirecional de mensagens. Dê uma descrição FSM do remetente e do destinatário. Descreva o formato dos pacotes enviados entre o remetente e o destinatário e vice-versa. Se você usar quaisquer procedimentos de chamada que não sejam os da Seção 3.4 (por exemplo, udt_send(), start_timer(), rdt_rcv() etc.), esclareça as ações desses procedimentos. Dê um exemplo (um diagrama de mensagens para o remetente e para o destinatário) mostrando como seu protocolo se recupera de uma perda de pacote.
17. Considere um cenário em que o hospedeiro A queira enviar pacotes para os hospedeiros B e C simultaneamente. O hospedeiro A está conectado a B e a C por um canal broadcast — um pacote enviado por A é levado pelo canal a B e a C. Suponha que o canal broadcast que conecta A, B e C possa, independentemente, perder e corromper mensagens (e assim, por exemplo, uma mensagem enviada de A poderia ser recebida corretamente por B, mas não por C). Projete um protocolo de controle de erro do tipo pare e espere para a transferência confiável de um pacote de A para B e para C, tal que A não receba novos dados da camada superior até que saiba que B e C receberam corretamente o pacote em questão. Dê descrições FSM de A e C. (Dica: a FSM para B deve ser essencialmente a mesma que para C.) Também dê uma descrição do(s) formato(s) de pacote usado(s).
18. Considere um cenário em que os hospedeiros A e B queiram enviar mensagens ao Hospedeiro C. Os hospedeiros A e C estão conectados por um canal que pode perder e corromper (e não reordenar) mensagens. Os hospedeiros B e C estão conectados por um outro canal (independente do canal que conecta A e C) com as mesmas propriedades. A camada de transporte no hospedeiro C deve alternar o envio de mensagens de A e B para a camada acima (ou seja, ela deve primeiro transmitir os dados de um pacote de A e depois os dados de um pacote de B, e assim por

- diante.) Elabore um protocolo de controle de erro pare e espere para transferência confiável de pacotes de A e B para C, com envio alternado em C, como descrito acima. Dê descrições FSM de A e C. [Dica: a FSM para B deve ser basicamente a mesma de A.] Dê, também, uma descrição do(s) formato(s) de pacote utilizado(s).
19. Considere o protocolo GBN com um tamanho de janela remetente de 3 e uma faixa de números de sequência de 1.024. Suponha que, no tempo t , o pacote seguinte na ordem, pelo qual o destinatário está esperando, tenha um número de sequência k . Admita que o meio não reordene as mensagens. Responda às seguintes perguntas:
- Quais são os possíveis conjuntos de números de sequência dentro da janela do remetente no tempo t ? Justifique sua resposta.
 - Quais são todos os possíveis valores do campo ACK em todas as mensagens que estão correntemente se propagando de volta ao remetente no tempo t ? Justifique sua resposta.
20. Suponha que haja duas entidades de rede A e B e que B tenha um suprimento de mensagens de dados que será enviado a A de acordo com as seguintes convenções: quando A recebe uma solicitação da camada superior para obter a mensagem de dados (D) seguinte de B, A deve enviar uma mensagem de requisição (R) para B no canal A-a-B; somente quando B receber uma mensagem R, ele poderá enviar uma mensagem de dados (D) de volta a A pelo canal B a A; A deve entregar uma cópia de cada mensagem D à camada superior; mensagens R podem ser perdidas (mas não corrompidas) no canal A-a-B; mensagens (D), uma vez enviadas, são sempre entregues corretamente; o atraso entre ambos os canais é desconhecido e variável.
- Elabore um protocolo (dê uma descrição FSM) que incorpore os mecanismos apropriados para compensar a propensão à perda do canal A a B e implemente passagem de mensagem para a camada superior na entidade A, como discutido antes. Utilize apenas os mecanismos absolutamente necessários.
21. Considere os protocolos GBN e SR. Suponha que o espaço de números de sequência seja de tamanho k . Qual será o maior tamanho de janela permitível que evitará que ocorram problemas como os da Figura 3.27 para cada um desses protocolos?
22. Responda verdadeiro ou falso às seguintes perguntas e justifique resumidamente sua resposta:
- Com o protocolo SR, é possível o remetente receber um ACK para um pacote que caia fora de sua janela corrente.
 - Com o GBN, é possível o remetente receber um ACK para um pacote que caia fora de sua janela corrente.
 - O protocolo bit alternante é o mesmo que o protocolo SR com janela do remetente e do destinatário de tamanho 1.
 - O protocolo bit alternante é o mesmo que o protocolo GBN com janela do remetente e do destinatário de tamanho 1.
23. Dissemos que um aplicação pode escolher o UDP para um protocolo de transporte pois ele oferece um controle de aplicações melhor (do que o TCP) de quais dados são enviados em um segmento e quando isso ocorre.
- Por que uma aplicação possui mais controle de quais dados são enviados em um segmento?
 - Por que uma aplicação possui mais controle de quando o segmento é enviado?
24. Considere a transferência de um arquivo enorme de L bytes do hospedeiro A para o hospedeiro B. Suponha um MSS de 536 bytes.
- Qual é o máximo valor de L tal que não sejam exauridos os números de sequência TCP? Lembre-se de que o campo de número de sequência TCP tem quatro bytes.
 - Para o L que obtiver em (a), descubra quanto tempo demora para transmitir o arquivo. Admita que um total de 66 bytes de cabeçalho de transporte, de rede e de enlace de dados seja adicionado a cada segmento antes que o pacote resultante seja enviado por um enlace de 155 Mbps. Ignore controle de fluxo e controle de congestionamento de modo que A possa enviar os segmentos um atrás do outro e continuamente.
25. Os hospedeiros A e B estão se comunicando por meio de uma conexão TCP, e o hospedeiro B já recebeu de A todos os bytes até o byte 126. Suponha que o Hospedeiro A envie, então, dois segmentos para o Hospedeiro B sucessivamente. O primeiro e o segundo segmentos contêm 70 e 50 bytes de dados, respectivamente. No primeiro segmento, o número de sequência é 127, o número de porta de partida é 302, e o número de porta de destino é 80. O hospedeiro B envia um reconhecimento ao receber um segmento do hospedeiro A.
- No segundo segmento enviado do Hospedeiro A para N, quais são o número de sequência, a porta de partida e a porta de destino?
 - Se o primeiro segmento chegar antes do segundo, no reconhecimento do primeiro segmento que chegar, qual é o número do reconhecimento, da porta de partida e da porta de destino?

- c. Se o segundo segmento chegar antes do primeiro, no reconhecimento do primeiro segmento que chegar, qual é o número do reconhecimento?
- d. Suponha que dois segmentos enviados por A cheguem em ordem a B. O primeiro reconhecimento é perdido e o segundo chega após o primeiro intervalo do esgotamento de temporização. Elabore um diagrama de temporização, mostrando esses segmentos, e todos os outros, e os reconhecimentos enviados. (Admita que não haja nenhuma perda de pacote adicional.) Para cada segmento de seu desenho, apresente o número de sequência e o número de bytes de dados; para cada reconhecimento adicionado por você, apresente o número do reconhecimento.
26. Os hospedeiros A e B estão diretamente conectados com um enlace de 100 Mbps. Existe uma conexão TCP entre os dois hospedeiros, e o hospedeiro A está enviando ao B um arquivo enorme por meio dessa conexão. O hospedeiro A pode enviar seus dados da aplicação para o socket TCP a uma taxa que chega a 120 Mbps, mas o Hospedeiro B pode ler o buffer de recebimento TCP a uma taxa de 60 Mbps. Descreva o efeito do controle de fluxo do TCP.
27. Os cookies SYN foram discutidos na Seção 3.5.6.
- Por que é necessário que o servidor use um número de sequência especial no SYNACK?
 - Suponha que um atacante saiba que um hospedeiro-alvo utilize SYN cookies. O atacante consegue criar conexões semiaberta ou completamente abertas simplesmente enviando um pacote ACK para o alvo? Por que sim? Por que não?
 - Suponha que um atacante receba uma grande quantidade de números de sequência enviados pelo servidor. O atacante consegue fazer com que o servidor crie muitas conexões totalmente abertas enviando ACKs com esses números de sequência? Por quê?
28. Considere a rede mostrada no Cenário 2 na Seção 3.6.1. Suponha que os hospedeiros emissores A e B possuam valores de esgotamento de temporização fixos.
- Analise o fato de que aumentar o tamanho do buffer finito do roteador pode possivelmente reduzir a vazão (λ_{out}).
 - Agora suponha que os hospedeiros ajustem dinamicamente seus valores de esgotamento de temporização (como o que o TCP faz) baseado no atraso no buffer no roteador. Aumentar o tamanho do buffer ajudaria a aumentar a vazão? Por quê?
29. Considere o procedimento TCP para estimar RTT. Suponha que $\alpha = 0,1$. Seja SampleRTT₁ a amostra mais recente de RTT, SampleRTT₂ a seguinte amostra mais recente de RTT etc.
- Para uma dada conexão TCP, suponha que quatro reconhecimentos foram devolvidos com as amostras RTT correspondentes SampleRTT₄, SampleRTT₃, SampleRTT₂ e SampleRTT₁. Expressse EstimatedRTT em termos das quatro amostras RTT.
 - Generalize sua fórmula para n amostras de RTTs.
 - Para a fórmula em (b), considere n tendendo ao infinito. Comente por que esse procedimento de média é denominado média móvel exponencial.
30. Na Seção 3.5.3 discutimos estimativa de RTT para o TCP. Na sua opinião, por que o TCP evita medir o SampleRTT para segmentos retransmitidos?
31. Qual é a relação entre a variável SendBase na Seção 3.5.4 e a variável LastByteRcvd na Seção 3.5.5?
32. Qual é a relação entre a variável LastByteRcvd na Seção 3.5.5 e na seção 3.5.4?
33. Na Seção 3.5.4 vimos que o TCP espera até receber três ACKs duplicados antes de realizar uma retransmissão rápida. Na sua opinião, por que os projetistas do TCP preferiram não realizar uma retransmissão rápida após ser recebido o primeiro ACK duplicado para um segmento?
34. Compare o GBN, SR e o TCP (sem ACK retardado). Admita que os valores do esgotamento de temporização para os três protocolos sejam suficientemente longos de tal modo que cinco segmentos de dados consecutivos e seus ACKs correspondentes possam ser recebidos (se não perdidos no canal) por um hospedeiro receptor (hospedeiro B) e por um hospedeiro emissor (hospedeiro A), respectivamente. Suponha que o hospedeiro A envie cinco segmentos de dados para o hospedeiro B, e que o segundo segmento (enviado de A) esteja perdido. No fim, todos os 5 segmentos de dados foram corretamente recebidos pelo hospedeiro A.
- Quantos segmentos o hospedeiro A enviou no total e quantos ACKs o hospedeiro B enviou no total? Quais são seus números de sequência? Responda essa questão considerando os três protocolos.
 - Se os valores do esgotamento de temporização para os três protocolos forem muito maiores do que 5 RTT, então qual protocolo envia com sucesso todos os cinco segmentos de dados em um menor intervalo de tempo?
35. Em nossa descrição sobre o TCP na Figura 3.53, o valor do limiar, ssthresh, é definido para $ssthresh = cwnd/2$ em diversos lugares e o valor ssthresh é referido como sendo definido para metade do tama-

- nho da janela quando ocorreu um evento de perda. A taxa à qual o emissor está enviando quando ocorreu o evento de perda deve ser aproximadamente igual a segmentos $cwnd$ por RTT? Explique sua resposta. Se sua resposta for negativa, você pode sugerir uma maneira diferente pela qual $ssthresh$ deve ser definido?
36. Considere a Figura 3.46(b). Se λ'_{in} aumentar para mais do que $R/2$, λ_{out} poderá aumentar para mais do que $R/3$? Explique. Agora considere a Figura 3.46(c). Se λ'_{in} aumentar para mais do que $R/2$, λ_{out} poderá aumentar para mais de $R/4$ admitindo-se que um pacote será transmitido duas vezes, em média, do roteador para o destinatário? Explique.
37. Considere a Figura 3.58. Admitindo-se que TCP Reno é o protocolo que experimenta o comportamento mostrado no gráfico, responda às seguintes perguntas. Em todos os casos você deverá apresentar uma justificativa resumida para sua resposta.
- Quais os intervalos de tempo em que a partida lenta do TCP está em execução?
 - Quais os intervalos de tempo em que a prevenção de congestionamento do TCP está em execução?
 - Após a 16^a rodada de transmissão, a perda de segmento será detectada por três ACKs duplicados ou por um esgotamento de temporização?
 - Após a 22^a rodada de transmissão, a perda de segmento será detectada por três ACKs duplicados ou por um esgotamento de temporização?
 - Qual é o valor inicial de $ssthresh$ na primeira rodada de transmissão?
 - Qual é o valor inicial de $ssthresh$ na 18^a rodada de transmissão?
38. Consulte a Figura 3.56, que ilustra a convergência do algoritmo AIMD do TCP. Suponha que, em vez de uma diminuição multiplicativa, o TCP reduza o tamanho da janela de uma quantidade constante. O AIMD resultante convergiria a um algoritmo de igual compartilhamento? Justifique sua resposta usando um diagrama semelhante ao da Figura 3.56.
39. Na Seção 3.5.4 discutimos a duplicação do intervalo de temporização após um evento de esgotamento de temporização. Esse mecanismo é uma forma de controle de congestionamento. Por que o TCP precisa de um mecanismo de controle de congestionamento que utiliza janelas (como estudado na Seção 3.7) além desse mecanismo de duplicação do intervalo de esgotamento de temporização?

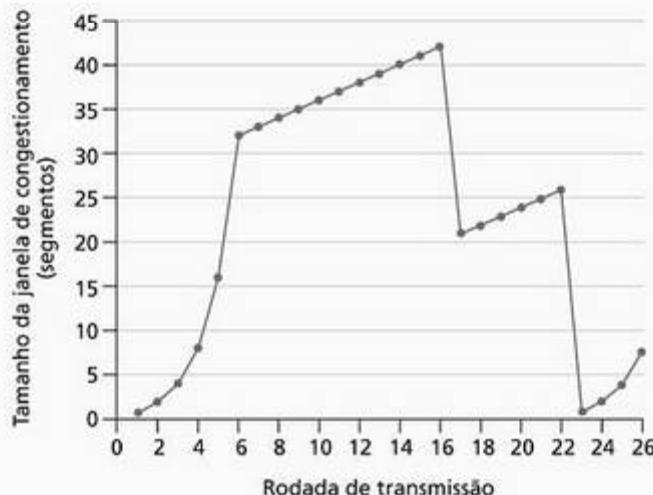


Figura 3.58 Tamanho da janela TCP como uma função de tempo

40. O hospedeiro A está enviando um arquivo enorme ao hospedeiro B por uma conexão TCP. Nessa conexão nunca há perda de pacotes e os temporizadores nunca se esgotam. Seja R bps a taxa de transmissão do enlace que liga o hospedeiro A à Internet. Suponha que o processo no hospedeiro A consiga enviar dados para seu socket TCP a uma taxa de S bps, em que $S = 10 \cdot R$. Suponha ainda que o buffer de recepção do TCP seja grande o suficiente para conter o arquivo inteiro e que o buffer de envio possa conter somente um por cento do arquivo. O que impediria o processo no hospedeiro A de passar dados continuamente para seu socket TCP à taxa de S bps: o controle de fluxo do TCP; o controle de congestionamento do TCP; ou alguma outra coisa? Elabore sua resposta.
41. Considere enviar um arquivo grande de um computador a outro por meio de uma conexão TCP em que não haja perda.
- Suponha que o TCP utilize AIMD para seu controle de congestionamento sem partida lenta. Admitindo que $cwnd$ aumenta 1 MSS sempre que um lote de ACK é recebido e os tempos da viagem de ida e volta constantes, quanto tempo leva para $cwnd$ aumentar de 5 MSS para 11 MSS? (admitindo nenhum evento de perda)?
 - Qual é a vazão média (em termos de MSS e RTT) para essa conexão sendo o tempo = 6 RTT?
42. Relembre a descrição macroscópica da vazão do TCP. No período de tempo transcorrido para a taxa da conexão variar de $W/(2 \cdot RTT)$ a W/RTT , apenas um pacote é perdido (bem ao final do período).
- Mostre que a taxa de perda (fração de pacotes perdidos) é igual a
- $$L = \text{fração de pacotes perdidos} = \frac{1}{\frac{3}{8} W^2 + \frac{3}{4} W}$$
- Use o resultado anterior para mostrar que, se uma conexão tiver taxa de perda L , sua largura de banda média é dada aproximadamente por:
- $$c. \frac{1.22 \cdot MSS}{RTT \sqrt{L}}$$
43. Considere que somente uma única conexão TCP (Reno) utiliza um enlace de 10 Mbps que não armazena nenhum dado. Suponha que esse enlace seja o único congestionado entre os hospedeiros emissor e receptor. Admita que o emissor TCP tenha um arquivo enorme para enviar ao receptor e o buffer de recebimento do receptor é muito maior do que a janela de congestionamento. Também fazemos as seguintes suposições: o tamanho de cada segmento TCP é 1.500 bytes; o atraso de propagação bidirecional dessa conexão é 10 milissegundos; e essa conexão está sempre na fase de prevenção de congestionamento, ou seja, ignore a partida lenta.
- Qual é o tamanho máximo da janela (em segmentos) que a conexão TCP pode atingir?
 - Qual é o tamanho médio da janela (em segmentos) e a vazão média (em bps) dessa conexão TCP?
 - Quanto tempo essa conexão TCP leva para alcançar sua janela máxima novamente após se recuperar de uma perda de pacote?
44. Considere o cenário descrito na questão anterior. Suponha que o enlace de 10 Mbps possa armazenar um número finito de segmentos. Demonstre que para o enlace sempre enviar dados, teríamos que escolher um tamanho de buffer que é, pelo menos, o produto da velocidade do enlace C e o atraso de propagação bidirecional entre o emissor e o receptor.
45. Repita a questão 43, mas substituindo o enlace de 10 Mbps por um de 10 Gbps. Observe que em sua resposta ao item (c), você verá que o tamanho da janela de congestionamento leva muito tempo para atingir seu tamanho máximo após se recuperar de uma perda de pacote. Elabore uma solução para resolver este problema.
46. Suponha que T (medido por RTT) seja o intervalo de tempo que uma conexão TCP leva para aumentar seu tamanho de janela de congestionamento de $W/2$ para W , sendo W o tamanho máximo da janela de congestionamento. Demonstre que T é uma função da vazão média do TCP.
47. Considere um algoritmo AIMD do TCP simplificado, sendo o tamanho da janela de congestionamento medido em número de segmentos e não em bytes. No aumento aditivo, o tamanho da janela de congestionamento aumenta por um segmento em cada RTT. Na diminuição multiplicativa, o tamanho da janela de congestionamento diminui para metade (se o resultado não for um número inteiro, arredondar para o número inteiro mais próximo). Suponha que duas conexões TCP, C_1 e C_2 , compartilhem um único enlace congestionado com 30 segmentos por segundo de velocidade. Admita que C_1 e C_2 estejam na fase de prevenção de congestionamento. GRTT da conexão C_1 é 100ms e o da conexão C_2 é de 200ms. Suponha que quando a taxa de dados no enlace excede a velocidade do enlace, todas as conexões TCP sofrem perda de segmento de dados.
- Se C_1 e C_2 no tempo t_0 possuem uma janela de congestionamento de 10 segmentos, quais são

- seus tamanhos de janela de congestionamento após 2.200 milissegundos?
- No final das contas, essas duas conexões obterão a mesma porção da largura de banda do enlace congestionado? Explique.
48. Considere a rede descrita na questão anterior. Agora suponha que as duas conexões TCP, C_1 e C_2 , possuam o mesmo RTT de 100 milissegundos e que no tempo t_0 o tamanho da janela de congestionamento de C_1 seja 15 segmentos, e que de C_2 seja 10 segmentos.
- Quais são os tamanhos de suas janelas de congestionamento após 2.200 milissegundos?
 - No final das contas, essas duas conexões irão obter a mesma porção da largura de banda do enlace congestionado?
 - Dizemos que duas conexões são sincronizadas se ambas atingirem o tamanho máximo e mínimo de janela ao mesmo tempo. No final das contas, essas duas conexões serão sincronizadas? Se sim, quais os tamanhos máximos de janela?
 - Essa sincronização ajudará a melhorar a utilização do enlace compartilhado? Por quê? Elabore alguma ideia para romper essa sincronização.
49. Considere uma modificação ao algoritmo de controle de congestionamento do TCP. Em vez do aumento aditivo, podemos utilizar a diminuição multiplicativa. Um emissor TCP aumenta seu tamanho de janela por uma constante positiva pequena ($0 < a < 1$) ao receber um ACK válido. Encontre a relação funcional entre a taxa de perda L e a janela máxima de congestionamento W . Para esse TCP modificado, demonstre, independentemente da vazão média TCP, que uma conexão TCP sempre gasta a mesma quantidade de tempo para aumentar seu tamanho da janela de congestionamento de $W/2$ para W .
50. Quando discutimos TCPs futuros na Seção 3.7, observamos que, para conseguir uma vazão de 10 Gbps, o TCP apenas poderia tolerar uma probabilidade de perda de segmentos de $2 \cdot 10^{-10}$ (ou, equivalentemente, uma perda para cada 5.000.000.000 segmentos). Mostre a derivação dos valores para $2 \cdot 10^{-10}$ ou 1: 5.000.000 a partir dos valores de RTT e do MSS dados na Seção 3.7. Se o TCP precisasse suportar uma conexão de 100 Gbps, qual seria a perda tolerável?
51. Quando discutimos controle de congestionamento em TCP na Seção 3.7, admitimos implicitamente que o remetente TCP sempre tinha dados para enviar. Agora considere o caso em que o remetente TCP envie uma grande quantidade de dados e então fique ocioso em t_1 (já que não há mais dados a enviar). O TCP permanecerá ocioso por um período de tempo relativamente longo e então irá querer enviar mais dados em t_2 . Quais são as vantagens e desvantagens do TCP utilizar os valores cwnd e ssthresh de t_1 quando começar a enviar dados em t_2 ? Que alternativa você recomendaria? Por quê?
52. Neste problema, verificamos se o UDP ou o TCP apresentam um grau de autenticação do ponto de chegada.
- Considere um servidor que recebe uma solicitação dentro de um pacote UDP e responde a essa solicitação dentro de um pacote UDP (por exemplo, como feito por um servidor DNS). Se um cliente com endereço IP X o engana com o endereço Y, para onde o servidor enviará sua resposta?
 - Suponha que um servidor receba um SYN de endereço-fonte IP Y com o número de reconhecimento correto. Admitindo que o servidor escolha um número de sequência aleatório e que não haja um "man-in-the-middle", o servidor pode ter certeza de que o cliente realmente está em Y (e não em outro endereço X que está enganando Y)?
53. Neste problema, consideramos o atraso apresentado pela fase de partida lenta do TCP. Considere um cliente e um servidor da Web diretamente conectados por um enlace de taxa R . Suponha que o cliente queira restaurar um objeto cujo tamanho seja exatamente igual a 15 S, sendo S o tamanho máximo do segmento (MSS). Considere RTT como o tempo de viagem de ida e volta entre o cliente e o servidor (admitindo que seja constante). Ignorando os cabeçalhos do protocolo, determine o tempo para restaurar o objeto (incluindo o estabelecimento da conexão TCP) quando
- $4 S/R > S/R + RTT > 2S/R$
 - $S/R + RTT > 4 S/R$
 - $S/R > RTT$



Questões dissertativas

- O que é sequestro de TCP? Como ele pode ser realizado?
- Na Seção 3.7, observamos que um cliente-servidor pode criar muitas conexões paralelas "de modo injusto". O que pode ser feito para tornar a Internet verdadeiramente justa?
- Consulte a literatura de pesquisa para saber o que quer dizer TCP *amigável*. Leia também a entrevista de Sally Floyd ao final deste capítulo. Redija uma página descrevendo a característica *amigável* do TCP.
- Ao final da Seção 3.7.1, discutimos o fato de uma aplicação querer abrir várias conexões TCP e obter uma vazão mais alta (ou, o que é equivalente, um tempo menor de transferência de dados). O que aconteceria se todas as aplicações tentassem melhorar seus desempenhos utilizando conexões múltiplas? Cite algumas das dificuldades envolvidas na utilização de um elemento da rede para determinar se uma aplicação está ou não usando conexões TCP múltiplas.
- Além da varredura de portas TCP e UDP, o nmap possui qual funcionalidade? Capture traços de pacote com o Ethereal (ou qualquer outro analisador de pacote) das trocas de pacote nmap. Utilize as trilhas para explicar como funcionam alguns recursos avançados.
- Em nossa descrição sobre o TCP na Figura 3.53, o valor inicial de cwnd é definido para 1. Consulte a literatura de pesquisa e uma RFC da Internet e discuta métodos alternativos que foram propostos para definir o valor inicial de cwnd.
- Consulte a literatura a respeito do SCTP [RFC 2960, RFC 3286]. Para quais aplicações os criadores do SCTP consideraram-no ser usado? Quais recursos do SCTP foram adicionados para atender às necessidades dessas aplicações?



Tarefas de programação

Detalhes completos das tarefas de programação podem ser encontrados no site http://www.aw.com/kurose_br.

Implementando um protocolo de transporte confiável

Nesta tarefa de programação de laboratório, você escreverá o código para a camada de transporte do remetente e do destinatário no caso da implementação de um protocolo simples de transferência confiável de dados. Há duas versões deste laboratório: a do protocolo de bit alternante e a do GBN. Essa tarefa será muito divertida, já que a sua implementação não será muito diferente da que seria exigida em uma situação real.

Como você provavelmente não tem máquinas autônomas (com um sistema operacional que possa modificar), seu código terá de rodar em um ambiente hardware/software simulado. Contudo, a interface de programação

fornecida às suas rotinas — isto é, o código que chama suas entidades de cima (da camada 5) e de baixo (da camada 3) — é muito próxima ao que é feito em um ambiente UNIX real. (Na verdade, as interfaces do software descritas nesta tarefa de programação são muito mais realistas do que os remetentes e destinatários de laço infinito descritos em muitos livros.) A parada e o acionamento dos temporizadores também são simulados, e as interrupções do temporizador farão com que sua rotina de tratamento de temporização seja ativada.

A tarefa completa de laboratório, assim como o código necessário para compilar seu próprio código, estão disponíveis no site: http://www.aw.com/kurose_br.



Wireshark Lab: Explorando o TCP

Neste laboratório, você usará seu browser Web para acessar um arquivo de um servidor Web. Como nos Wireshark Labs anteriores, você usará Wireshark para capturar os pacotes que estão chegando ao seu computador. Mas, diferentemente daqueles laboratórios, também

poderá descarregar do mesmo servidor Web do qual descarregou o arquivo, um relatório de mensagens (trace) que pode ser lido pelo Wireshark. Nesse relatório de mensagens do servidor, você encontrará os pacotes que foram gerados pelo seu próprio acesso ao servidor Web. Você

analisará os diagramas dos lados do cliente e do servidor de modo a explorar aspectos do TCP e, em especial, fará uma avaliação do desempenho da conexão TCP entre seu computador e o servidor Web. Utilizando o diagrama de mensagens, você analisará o comportamento da janela TCP e inferirá comportamentos de perda de pacote, de

retransmissão, de controle de fluxo e de controle de congestionamento e do tempo de ida e volta estimado.

Como acontece com todos os Wireshark Labs, a descrição completa deste pode ser encontrada, em inglês, no site http://www.aw.com/kurose_br.



Wireshark Lab: explorando o UDP

Neste pequeno laboratório, você realizará uma captura de pacote e uma análise de sua aplicação favorita que utiliza o UDP (por exemplo, o DNS ou uma aplicação multimídia, como o Skype). Como aprendemos na Seção 3.3, o UDP é um protocolo de transporte simples. Neste laboratório,

você examinará os campos do cabeçalho no segmento UDP, assim como o cálculo da soma de verificação.

Como acontece com todos os Wireshark Labs, a descrição completa deste pode ser encontrada, em inglês, no site http://www.aw.com/kurose_br.

Entrevista

Sally Floyd

Sally Floyd é cientista-pesquisadora no ICSI Center for Internet Research, um instituto dedicado a questões da Internet e de redes. Ela é conhecida no setor por seu trabalho em projetos de protocolos de Internet, em especial nas áreas de multicast confiável, controle de congestionamento (TCP), escalonamento de pacotes (RED) e análise de protocolos. Sally é bacharel em Sociologia pela Universidade da Califórnia em Berkeley, e mestre e doutora em ciência da computação por essa mesma universidade.



Como você decidiu estudar ciência da computação?

Após terminar o bacharelado em sociologia, comecei a me preocupar em obter meu próprio sustento. Acabei fazendo um curso de eletrônica de dois anos na faculdade local e, em seguida, passei dez anos trabalhando com eletrônica e ciência da computação. Nesses dez anos, estão incluídos os oito que trabalhei como engenheira de sistemas de computação, cuidando dos computadores dos trens da Bay Area Rapid Transit (BART). Mais tarde, decidi estudar ciência da computação mais formalmente e iniciei a pós-graduação no Departamento de Ciência da Computação da Universidade da Califórnia em Berkeley.

O que a fez se decidir pela especialização em redes?

Na pós-graduação, comecei a me interessar pela parte teórica da ciência da computação. Primeiramente, dediquei-me à análise probabilística de algoritmos e, mais tarde, à teoria do aprendizado da computação. Nessa época, trabalhava também no Lawrence Berkeley Laboratory (LBL) uma vez por mês, e meu escritório ficava em frente ao escritório de Van Jacobson, que pesquisava algoritmos de controle de congestionamento no TCP. Certo dia, Van me perguntou se eu gostaria de trabalhar na análise de alguns algoritmos para um problema relacionado a redes, que envolvia a indesejada sincronização de mensagens periódicas de roteamento. Isso me pareceu interessante, e foi o que fiz durante o verão.

Após eu ter concluído minha tese, Van me ofereceu um emprego de tempo integral para continuar trabalhando com redes. Eu não tinha planejado ficar trabalhando com redes durante dez anos, mas, para mim, a pesquisa é mais gratificante do que a ciência teórica da computação. Sinto mais satisfação em trabalhos aplicados, pois as consequências do que faço são mais tangíveis.

Qual foi seu primeiro emprego no setor de computação? O que implicava?

Meu primeiro emprego na área de computação foi na BART, onde trabalhei de 1975 a 1982 com os computadores que controlavam os trens. Comecei como técnica em manutenção e conserto dos vários sistemas distribuídos de computadores envolvidos na operação do sistema da BART.

Entre esses sistemas estavam: um sistema central de computadores e um sistema distribuído de minicomputadores, que controlavam a movimentação dos trens; um sistema de computadores da DEC, que exibia avisos e destinos de trens nos painéis, e um sistema de computadores Modcomp, que coletava informações das bilheterias. Meus últimos anos na BART foram dedicados a um projeto conjunto BART/LBL para estruturar um sistema substituto para o sistema de computadores de controle de trens da BART, que já estava ficando ultrapassado.

Qual parte de seu trabalho lhe apresenta mais desafios?

A pesquisa é a parte mais desafiadora. O primeiro tópico de pesquisa inclui explorar as questões referentes ao controle de congestionamento para aplicações, como o fluxo de mídia. Um segundo tópico é abordar obstáculos da rede para uma comunicação mais explícita entre os roteadores e os nós finais. Esses obstáculos podem incluir túneis IP e caminhos MPLS, roteadores ou dispositivos intermediários que soltam os pacotes contendo opções IP, redes complexas da camada 2 e potenciais para ataques na rede. Um terceiro tópico existente é explorar como nossa escolha de modelos de cenários para uso em análise, simulação e experimentos afeta nossa avaliação do desempenho dos mecanismos de controle de congestionamento. Mais informações sobre esses assuntos podem ser encontradas nas páginas Web do DCCP, Quick-Start e TMRG em <http://www.icir.org/floyd>.

Em sua opinião, qual é o futuro das redes e da Internet?

Uma possibilidade é que o congestionamento típico encontrado pelo tráfego da Internet se torne menos severo à medida que a largura de banda disponível aumentar mais rapidamente do que a demanda. Considero que a tendência está se dirigindo para um congestionamento menos intenso, embora não pareça de todo impossível contemplar um futuro de crescente congestionamento, a médio prazo, pontuado por colapso ocasional.

O futuro da Internet em si ou da arquitetura da Internet ainda não está claro para mim. Há muitos fatores que estão contribuindo para rápidas mudanças. Portanto, é difícil predizer como a Internet ou a arquitetura da Internet evoluirão ou mesmo prever o grau de sucesso que essa evolução será capaz de alcançar na prevenção das muitas armadilhas potenciais ao longo do caminho.

Uma tendência negativa bem conhecida é a dificuldade crescente de realizar mudanças na arquitetura da Internet, que não é mais um conjunto coerente, e os vários componentes como protocolos de transporte, mecanismos do roteador, firewalls,平衡adores de carga, mecanismos de segurança e seus semelhantes às vezes trabalham com objetivos contrários.

Quais pessoas a inspiraram profissionalmente?

Richard Karp, o orientador de minha tese, me ensinou a fazer, de fato, pesquisa. Van Jacobson, meu "chefe de grupo" no LBL, foi o responsável pelo meu interesse no estudo de redes e por grande parte do meu aprendizado sobre a infraestrutura da Internet. Dave Clark me inspirou por sua visão clara da arquitetura da Internet e por seu papel no desenvolvimento dessa arquitetura por meio de pesquisa, de artigos e da participação na IETF e em outros fóruns públicos. Deborah Estrin me inspirou com sua capacidade de concentração e efetividade, e sua habilidade em tomar decisões lúcidas sobre o que investigar e por quê.

Uma das razões por que aprecio trabalhar na pesquisa de rede é que encontrei nesse campo muitas pessoas de que gosto, a quem respeito e admiro. São pessoas inteligentes, batalhadoras e que estão profundamente comprometidas com o desenvolvimento da Internet, divergem (ou concordam) de modo amigável e são bons companheiros para uma cerveja após um dia de reuniões.



Capítulo 4

A camada de rede



Vimos no capítulo anterior que a camada de transporte provê várias formas de comunicação processo a processo com base no serviço de comunicação hospedeiro a hospedeiro da camada de rede. Vimos também que a camada de transporte faz isso sem saber como a camada de rede implementa esse serviço. Portanto, é bem possível que agora você esteja imaginando o que está por baixo do serviço de comunicação hospedeiro a hospedeiro; o que o faz funcionar.

Neste capítulo estudaremos exatamente como a camada de rede implementa o serviço de comunicação hospedeiro a hospedeiro. Veremos que há um pedaço da camada de rede em cada um dos hospedeiros e roteadores na rede, o que não acontece com a camada de transporte. Por causa disso, os protocolos de camada de rede estão entre os mais desafiadores (e, portanto, os mais interessantes!) da pilha de protocolos.

A camada de rede é, também, uma das camadas mais complexas da pilha de protocolos e, assim, temos um longo caminho a percorrer. Iniciaremos nosso estudo com uma visão geral da camada de rede e dos serviços que ela pode prover. Em seguida, examinaremos novamente as duas abordagens da estruturação da entrega de pacotes de camada de rede — o modelo de datagramas e o modelo de circuitos virtuais — que vimos pela primeira vez no Capítulo 1 e veremos o papel fundamental que o endereçamento desempenha na entrega de um pacote a seu hospedeiro de destino.

Faremos, neste capítulo, uma distinção importante entre as funções de **repasse** e **roteamento** da camada de rede. Repasse envolve a transferência de um pacote de um enlace de entrada para um enlace de saída dentro de um único roteador. Roteamento envolve *todos* os roteadores de uma rede, cujas interações coletivas por meio do protocolo de roteamento determinam os caminhos que os pacotes percorrem em suas viagens do nó de origem ao nó de destino. Essa será uma distinção importante para manter em mente ao ler este capítulo.

Para aprofundar nosso conhecimento do repasse de pacotes, examinaremos o “interior” de um roteador — a organização e a arquitetura de seu hardware. Então, examinaremos o repasse de pacotes na Internet, juntamente com o famoso Protocolo da Internet (IP). Investigaremos o endereçamento na camada de rede e o formato de datagrama IPv4. Em seguida, estudaremos a tradução de endereço de rede (*network address translation* — NAT), a fragmentação de datagrama, o Protocolo de Mensagem de Controle da Internet (*Internet Control Message Protocol* — ICMP) e IPv6.

Então voltaremos nossa atenção à função de roteamento da camada de rede. Veremos que o trabalho de um algoritmo de roteamento é determinar bons caminhos (ou rotas) entre remetentes e destinatários. Em primeiro lugar, estudaremos a teoria de algoritmos de roteamento concentrando nossa atenção nas duas classes de algoritmos mais predominantes: algoritmos de estado de enlace e de vetor de distâncias. Visto que a complexidade

dos algoritmos de roteamento aumenta consideravelmente com o aumento do número de roteadores na rede, também será interessante abordar o roteamento hierárquico. Em seguida veremos como a teoria é posta em prática quando falarmos sobre os protocolos de roteamento de intrassistemas autônomos da Internet (RIP, OSPF e IS-IS) e seu protocolo de roteamento de intersistemas autônomos, o BGP. Encerraremos este capítulo com uma discussão sobre roteamento *broadcast* e *multicast*.

Em resumo, este capítulo tem três partes importantes. A primeira, seções 4.1 e 4.2, aborda funções e serviços de camada de rede. A segunda, seções 4.3 e 4.4, examina o repasse. Finalmente, a terceira parte, seções 4.5 a 4.7, estuda roteamento.

4.1 Introdução

A Figura 4.1 mostra uma rede simples com dois hospedeiros, H1 e H2, e diversos roteadores no caminho entre H1 e H2. Suponha que H1 esteja enviando informações a H2 e considere o papel da camada de rede nesses hospedeiros e nos roteadores intervenientes. A camada de rede em H1 pegará segmentos da camada de transporte em H1, encapsulará cada segmento em um datagrama (isto é, em um pacote de camada de rede) e então dará inicio à jornada dos datagramas até seu destino, isto é, enviará os datagramas para seu roteador vizinho, R1. No hospedeiro receptor (H2), a camada de rede receberá os datagramas de seu roteador vizinho R2, extrairá os segmentos de camada de transporte e os entregará à camada de transporte em H2. O papel primordial dos roteadores é repassar datagramas de enlaces de entrada para enlaces de saída. Note que os roteadores da Figura 4.1 são mostrados com a pilha de protocolos truncada, isto é, sem as camadas superiores acima da camada de rede, porque (exceto para finalidades de controle) roteadores não rodam protocolos de camada de transporte e de aplicação como os que examinamos nos capítulos 2 e 3.

4.1.1 Repasse e roteamento

Assim, o papel da camada de rede é aparentemente simples — transportar pacotes de um hospedeiro remetente a um hospedeiro destinatário. Para fazê-lo, duas importantes funções da camada de rede podem ser identificadas:

Repasse. Quando um pacote chega ao enlace de entrada de um roteador, este deve conduzi-lo até o enlace de saída apropriado. Por exemplo, um pacote proveniente do hospedeiro H1 que chega ao roteador R1 deve ser repassado ao roteador seguinte por um caminho até H2. Na Seção 4.3 examinaremos o interior de um roteador e investigaremos como um pacote é realmente repassado de um enlace de entrada de um roteador até um enlace de saída.

Roteamento. A camada de rede deve determinar a rota ou caminho tomado pelos pacotes ao fluírem de um remetente a um destinatário. Os algoritmos que calculam esses caminhos são denominados **algoritmos de roteamento**. Um algoritmo de roteamento determinaria, por exemplo, o caminho ao longo do qual os pacotes fluiriam de H1 para H2.

Os termos *repasse* e *roteamento* são usados indistintamente por autores que discutem a camada de rede. Neste livro, usaremos esses termos com maior exatidão. *Repasse* refere-se à ação local realizada por um roteador para transferir um pacote da interface de um enlace de entrada para a interface de enlace de saída apropriada. *Roteamento* refere-se ao processo de âmbito geral da rede que determina os caminhos fim a fim que os pacotes percorrem desde a fonte até o destino. Para usar uma viagem como analogia, voltemos àquele nosso viajante da Seção 1.3.2, que vai da Pensilvânia à Flórida. Durante a viagem, nosso motorista passa por muitos cruzamentos de rodovias em sua rota até a Flórida. Podemos imaginar o repasse como o processo de passar por um único cruzamento: um carro chega ao cruzamento vindo de uma rodovia e determina qual rodovia ele deve pegar para sair do cruzamento. Podemos imaginar o roteamento como o processo de planejamento da viagem da Pensilvânia até a Flórida: antes de partir, o motorista consultou um mapa e escolheu um dos muitos caminhos possíveis. Cada um desses caminhos consiste em uma série de trechos de rodovias conectados por cruzamentos.