

A camada de rede: plano de controle

Neste capítulo, completamos nossa jornada pela camada de rede ao cobrirmos o componente de **plano de controle**, a lógica *em âmbito de rede* que controla a maneira como um datagrama é roteado por um caminho fim a fim, do hospedeiro de origem ao de destino, e como os serviços e componentes da camada de rede são configurados e gerenciados. Na Seção 5.2, examinaremos os algoritmos de roteamento tradicionais para calcular os caminhos de menor custo em um grafo, que são a base para dois protocolos de roteamento da Internet amplamente utilizados, o OSPF e o BGP, que trabalharemos nas Seções 5.3 e 5.4, respectivamente. Como veremos, o OSPF é um protocolo de roteamento que opera na rede em um único provedor de serviços de Internet (ISP, do inglês *Internet Service Provider*). O BGP é um protocolo de roteamento que serve para interconectar todas as redes na Internet, sendo muitas vezes chamado de a “cola” que une toda a Internet. Tradicionalmente, os protocolos de roteamento do plano de controle são implementados juntos com as funções de repasse do plano de dados, monoliticamente, no roteador. Como vimos na introdução ao Capítulo 4, as redes definidas por software (SDNs, do inglês *software-defined networking*) estabelecem uma separação clara entre os planos de dados e de controle, implementando as funções do plano de controle em um serviço “controlador” independente, distinto e remoto em relação aos componentes de repasse dos roteadores que controla. Analisaremos os controladores SDN na Seção 5.5.

Nas Seções 5.6 e 5.7, examinaremos um pouco dos mecanismos básicos do gerenciamento de uma rede IP: ICMP (do inglês *Internet Control Message Protocol* – Protocolo de Mensagens de Controle da Internet) e SNMP (do inglês *Simple Network Management Protocol* – Protocolo Simples de Gerenciamento de Rede).

5.1 INTRODUÇÃO

Para definir rapidamente o contexto do nosso estudo sobre o plano de controle da rede, retomemos as Figuras 4.2 e 4.3. Nelas, vimos que a tabela de repasse (no caso do repasse baseado em destino) e a tabela de fluxo (no caso do repasse generalizado) eram os principais elementos a ligar os planos de dados e de controle da camada de rede. Vimos também

que essas tabelas especificam o comportamento de repasse do plano de dados local de um roteador. No caso do repasse generalizado, aprendemos que as ações executadas podem incluir, além do repasse de um pacote para a porta de saída do roteador, também o descarte do pacote, a replicação do pacote e/ou a reescrita dos campos de cabeçalho do pacote da camada 2, 3 ou 4.

Neste capítulo, estudaremos como essas tabelas de repasse e de fluxo são calculadas, mantidas e instaladas. Na nossa introdução à camada de rede na Seção 4.1, aprendemos que há duas abordagens possíveis:

- *Controle por roteador*. A Figura 5.1 ilustra o caso em que um algoritmo de roteamento é executado em todos os roteadores, sem exceção; tanto a função de repasse quanto a de roteamento estão contidas em cada roteador. Cada um possui um componente de roteamento que se comunica com os seus equivalentes em outros roteadores para calcular os valores da sua tabela de repasse. Essa abordagem de controle por roteador é usada na Internet há décadas. Os protocolos OSPF e BGP, que estudaremos nas Seções 5.3 e 5.4, se baseiam nessa abordagem por roteador para o controle.
- *Controle logicamente centralizado*. A Figura 5.2 ilustra o caso em que um controlador logicamente centralizado calcula e distribui as tabelas de repasse a serem usadas por todos os roteadores. Como vimos nas Seções 4.4 e 4.5, a abstração “combinação mais ação” generalizada permite que o roteador realize o repasse de IP tradicional e um amplo conjunto de outras funções (compartilhamento de carga, firewall e tradução de endereços de rede [NAT, do inglês *network address translation*]), antes implementadas em *middleboxes* separadas.

O controlador interage com um agente de controle (CA, do inglês *control agent*) em cada um dos roteadores usando um protocolo claramente definido para configurar e gerenciar a tabela de fluxo do roteador. Em geral, o CA tem um mínimo de funcionalidade; a sua função é se comunicar com o controlador e seguir os seus comandos. Ao contrário dos algoritmos de roteamento da Figura 5.1, os CAs não interagem diretamente uns com os outros e não participam ativamente do cálculo da tabela de repasse. É uma diferença crítica entre o controle por roteador e o controle logicamente centralizado.

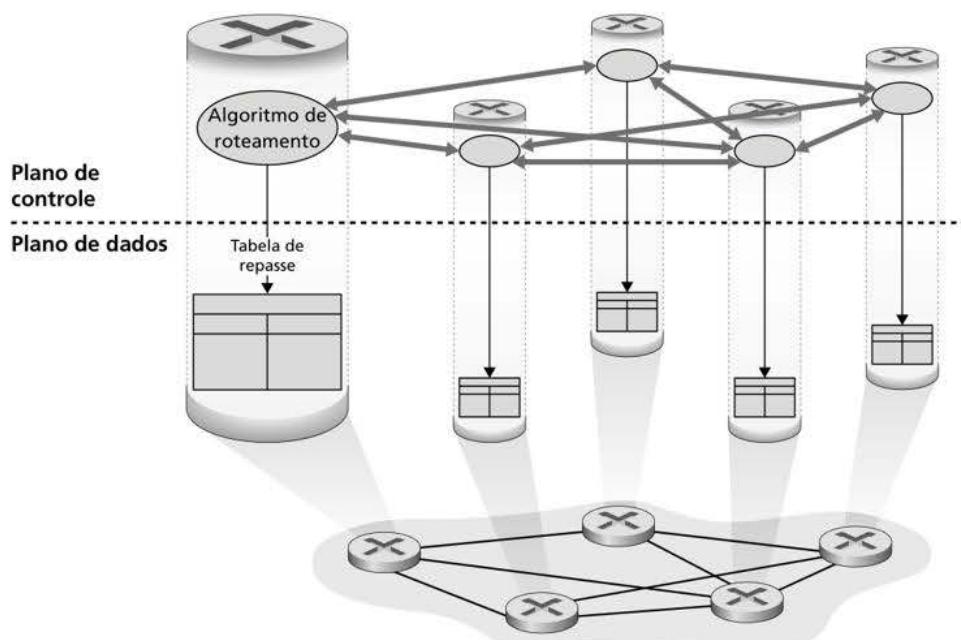


Figura 5.1 Controle por roteador: os componentes individuais do algoritmo de roteamento interagem no plano de controle.

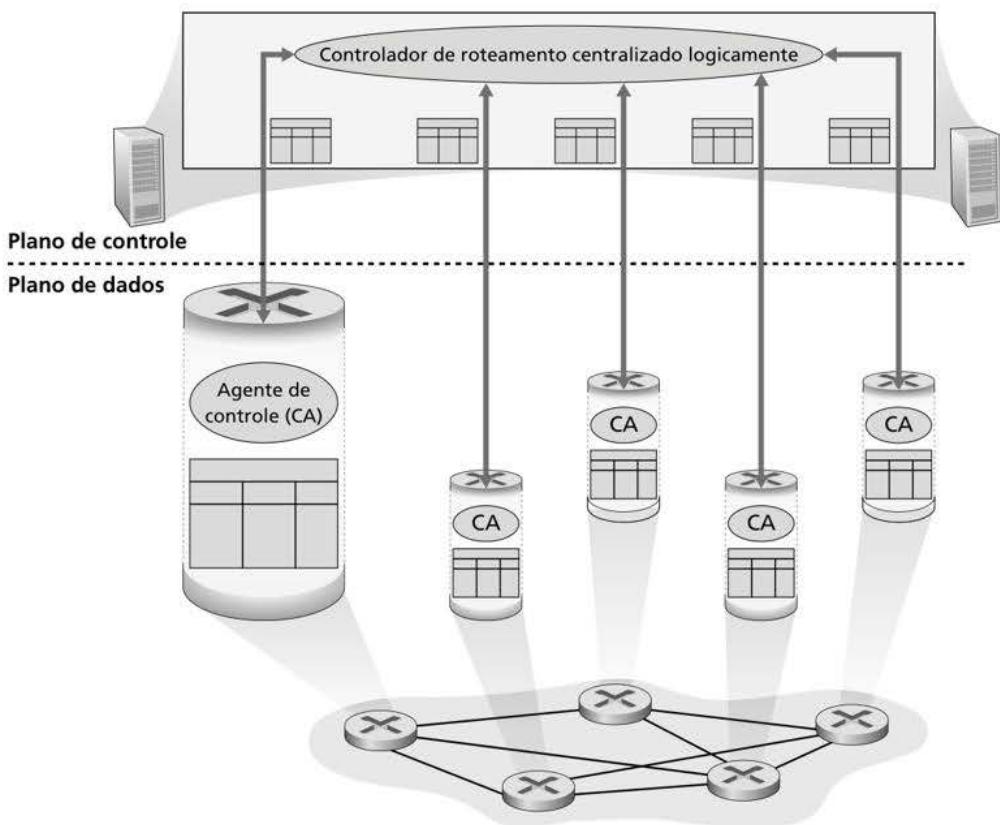


Figura 5.2 Controle logicamente centralizado: um controlador distinto, geralmente remoto, interage com os agentes de controle (CAs) locais.

Por controle “logicamente centralizado” (Levin, 2012), nos referimos ao serviço de controle de roteamento acessado como se fosse um único ponto de serviço central, ainda que o serviço provavelmente seja implementado por meio de múltiplos servidores para fins de tolerância a falhas e escalabilidade do desempenho. Como veremos na Seção 5.5, as SDNs adotam essa ideia de um controlador logicamente centralizado, uma abordagem cada vez mais utilizada em ambientes de produção. A Google usa SDN para controlar os roteadores da sua rede de longa distância global interna B4, que interconecta seus datacenters (Jain, 2013). A SWAN (Hong, 2013), da Microsoft Research, usa um controlador logicamente centralizado para gerenciar o roteamento e o repasse entre uma rede de longa distância e uma rede de datacenters. Grandes implementações por ISPs, incluindo o ActiveCore da COMCAST e o Access 4.0 da Deutsche Telecom, estão integrando SDNs ativamente às suas redes. E como veremos no Capítulo 8, o controle de SDNs também é fundamental para as redes celulares 4G/5G. A AT&T (2019) observa que “... a SDN não é uma visão, uma meta ou uma promessa. É uma realidade. Até o final do próximo ano, 75% das nossas funções de rede serão totalmente virtualizadas e controladas por software”. A China Telecom e a China Unicom usam SDNs dentro dos seus datacenters e entre eles (Li, 2015).

5.2 ALGORITMOS DE ROTEAMENTO

Nesta seção, estudaremos **algoritmos de roteamento**, cujo objetivo é determinar bons caminhos (em outras palavras, rotas) entre remetentes e destinatários através da rede de roteadores. Em geral, um “bom” caminho é aquele que tem o menor custo. No entanto, veremos que,

na prática, preocupações do mundo real, como questões de política (p. ex., uma regra que determina que “o roteador x , de propriedade da organização Y , não deverá repassar nenhum pacote originário da rede de propriedade da organização Z ”), também entram em jogo. Vale observar que, independentemente do plano de controle da rede adotar uma abordagem de controle por roteador ou logicamente centralizada, deve sempre haver uma sequência definida de roteadores que um pacote atravessará entre o hospedeiro de origem e o de destino. Assim, os algoritmos de roteamento que calculam esses caminhos são de suma importância e representam outro candidato para a nossa lista dos dez conceitos de rede fundamentais.

Um grafo é usado para formular problemas de roteamento. Lembre-se de que um **grafo** $G = (N, E)$ é um conjunto N de nós e uma coleção E de arestas, no qual cada aresta é um par de nós do conjunto N . No contexto do roteamento da camada de rede, os nós do grafo representam roteadores – os pontos nos quais são tomadas decisões de repasse de pacotes – e as arestas que conectam os nós representam os enlaces físicos entre esses roteadores. Uma abstração gráfica de uma rede de computadores está exibida na Figura 5.3. Quando estudarmos o protocolo de roteamento interdomínio BGP, veremos que os nós representam redes, e a aresta que conecta dois nós desse tipo representa a direção da conectividade (chamada de emparelhamento) entre as duas redes. Para ver alguns grafos representando mapas de rede reais, consulte (CAIDA, 2020); para uma discussão de como os diferentes modelos baseados em grafo modelam a Internet, consulte (Zegura, 1997; Faloutsos, 1999; Li, 2004).

Como ilustrado na Figura 5.3, uma aresta também tem um valor que representa seu custo. Em geral, o custo de uma aresta pode refletir o tamanho físico do enlace correspondente (p. ex., um enlace transoceânico poderia ter um custo mais alto do que um enlace terrestre de curta distância), a velocidade do enlace ou o custo monetário a ele associado. Para nossos objetivos, consideraremos os custos das arestas apenas como um dado, e não nos preoccuparemos com o modo como eles são determinados. Para qualquer aresta (x, y) em E , denominamos $c(x, y)$ o custo da aresta entre os nós x e y . Se o par (x, y) não pertencer a E , estabelecemos $c(x, y) = \infty$. Além disso, sempre consideraremos somente grafos não direcionados (i.e., grafos cujas arestas não têm uma direção), de modo que a aresta (x, y) é a mesma que a aresta (y, x) e $c(x, y) = c(y, x)$; contudo, os algoritmos que estudaremos podem ser estendidos facilmente para o caso de enlaces direcionados com um custo diferente em cada direção. Dizemos também que y é um **vizinho** do nó x se (x, y) pertencer a E .

Já que são atribuídos custos às várias arestas na abstração do grafo, uma meta natural de um algoritmo de roteamento é identificar o caminho de menor custo entre origens e destinos. Para tornar esse problema mais preciso, lembremos que um **caminho** em um grafo $G = (N, E)$ é uma sequência de nós (x_1, x_2, \dots, x_p) tal que cada um dos pares $(x_1, x_2), (x_2, x_3), \dots, (x_{p-1}, x_p)$ são arestas em E . O custo de um caminho (x_1, x_2, \dots, x_p) é apenas a soma de todos os custos das arestas ao longo do caminho, ou seja, $c(x_1, x_2) + c(x_2, x_3) + \dots + c(x_{p-1}, x_p)$. Dados quaisquer dois nós x e y , em geral há muitos caminhos entre os dois, e cada caminho tem um custo. Um ou mais desses caminhos é um **caminho de menor custo**. Por conseguinte,

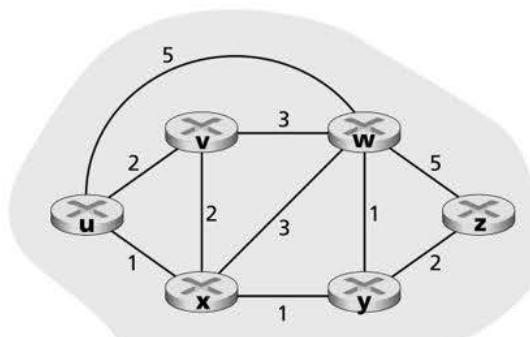


Figura 5.3 Modelo abstrato de grafo de uma rede de computadores.

o problema do menor custo é claro: descobrir um caminho entre a origem e o destino que tenha o menor custo. Na Figura 5.3, por exemplo, o caminho de menor custo entre o nó da origem u e o de destino w é (u, x, y, w) , cujo custo de caminho é 3. Note que, se todas as arestas do grafo tiverem o mesmo custo, o caminho de menor custo também é o caminho **mais curto** (i.e., o que tem o menor número de enlaces entre a origem e o destino).

Apenas como simples exercício, tente descobrir o caminho de menor custo entre os nós u a z na Figura 5.3 e reflita um pouco sobre como você o calculou. Se você for como a maioria das pessoas, descobriu o caminho de u a z examinando a figura, traçando algumas rotas de u a z , e se convencendo, de algum modo, que o caminho escolhido tinha o menor custo entre todos os possíveis. (Você verificou todos os 17 possíveis caminhos entre u e z ? Provavelmente, não!) Esse cálculo é um exemplo de um algoritmo de roteamento centralizado – o algoritmo é rodado em um local, o seu cérebro, com informações completas sobre a rede. De modo geral, uma maneira possível de classificar algoritmos de roteamento é como centralizados ou descentralizados.

- Um **algoritmo de roteamento centralizado** calcula o caminho de menor custo entre uma origem e um destino usando conhecimento completo e global sobre a rede. Em outras palavras, o algoritmo considera como entradas a conectividade entre todos os nós e todos os custos dos enlaces. E isso exige que o algoritmo obtenha essas informações, de algum modo, antes de realizar de fato o cálculo. Este pode ser rodado em um local (p. ex., um controlador logicamente centralizado, como na Figura 5.2) ou replicado no componente de roteamento de cada um dos roteadores (p. ex., como na Figura 5.1). Contudo, a principal característica distintiva, nesse caso, é que um algoritmo global tem informação completa sobre conectividade e custo de enlaces. Algoritmos com informação global de estado são com frequência denominados **algoritmos de estado de enlace (LS, do inglês link-state)**, já que devem estar a par dos custos de cada enlace na rede. Estudaremos algoritmos de estado de enlace na Seção 5.2.1.
- Em um **algoritmo de roteamento descentralizado**, o cálculo do caminho de menor custo é realizado de modo iterativo e distribuído pelos roteadores. Nenhum nó tem informação completa sobre os custos de todos os enlaces da rede. Em vez disso, cada nó começa sabendo apenas os custos dos enlaces diretamente ligados a ele. Então, por meio de um processo iterativo de cálculo e de troca de informações com seus nós vizinhos, um nó gradualmente calcula o caminho de menor custo até um destino ou um conjunto de destinos. O algoritmo de roteamento descentralizado que estudaremos logo adiante na Seção 5.2.2 é denominado algoritmo de vetor de distâncias (DV, do inglês *distance-vector*), porque cada nó mantém um vetor de estimativas de custos (distâncias) de um nó até todos os outros nós da rede. Esses algoritmos, com trocas de mensagens interativas entre roteadores vizinhos, podem ser mais naturalmente apropriados para os planos de controle em que roteadores interagem diretamente uns com os outros, como na Figura 5.1.

Uma segunda maneira geral de classificar algoritmos de roteamento é como estáticos ou dinâmicos. Em **algoritmos de roteamento estáticos**, as rotas mudam muito devagar ao longo do tempo, muitas vezes como resultado de intervenção humana (p. ex., uma pessoa editando manualmente os custos de enlace). **Algoritmos de roteamento dinâmicos** mudam os caminhos de roteamento à medida que mudam as cargas de tráfego ou a topologia da rede. Um algoritmo dinâmico pode ser rodado periodicamente ou como reação direta a mudanças de topologia ou de custo dos enlaces. Ao mesmo tempo em que são mais sensíveis a mudanças na rede, algoritmos dinâmicos também são mais suscetíveis a problemas como laços de roteamento e oscilação em rotas.

Uma terceira maneira de classificar algoritmos de roteamento é como sensíveis à carga ou insensíveis à carga. Em um **algoritmo sensível à carga**, custos de enlace variam dinamicamente para refletir o nível corrente de congestionamento no enlace subjacente. Se houver um alto custo associado com um enlace que está congestionado, um algoritmo de roteamento tenderá a escolher rotas que evitem esse enlace. Embora antigos algoritmos de roteamento da ARPAnet fossem sensíveis à carga (McQuillan, 1980), foram encontradas

várias dificuldades (Huitema, 1998). Os algoritmos de roteamento utilizados na Internet hoje (p. ex., RIP, OSPF e BGP) são **insensíveis à carga**, pois o custo de um enlace não reflete explicitamente seu nível de congestionamento atual (nem o mais recente).

5.2.1 O algoritmo de roteamento de estado de enlace (LS)

Lembre-se de que, em um algoritmo de estado de enlace, a topologia da rede e todos os custos de enlace são conhecidos, isto é, estão disponíveis como dados para o algoritmo de estado de enlace. Na prática, isso se consegue fazendo cada nó transmitir pacotes de estado de enlace a *todos* os outros nós da rede, uma vez que cada um desses pacotes contém as identidades e os custos dos enlaces ligados a ele. Na prática (p. ex., com o protocolo de roteamento OSPF da Internet, discutido na Seção 5.3), isso frequentemente é conseguido com um algoritmo de **transmissão por difusão de estado de enlace** (Perlman, 1999). O resultado da transmissão por difusão dos nós é que todos os nós têm uma visão idêntica e completa da rede. Cada um pode, então, rodar o algoritmo de estado de enlace e calcular o mesmo conjunto de caminhos de menor custo como todos os outros nós.

O algoritmo de roteamento de estado de enlace que apresentaremos adiante é conhecido como *algoritmo de Dijkstra*, o nome de seu inventor. Um algoritmo que guarda relações muito próximas com ele é o algoritmo de Prim; consulte Cormen (2001) para ver uma discussão geral sobre algoritmos de grafo. O algoritmo de Dijkstra calcula o caminho de menor custo entre um nó (a origem, que chamaremos de u) e todos os outros nós da rede. É um algoritmo iterativo e tem a propriedade de, após a k -ésima iteração, conhecer os caminhos de menor custo para k nós de destino e, entre os caminhos de menor custo até todos os nós de destino, esses k caminhos terão os k menores custos. Vamos definir a seguinte notação:

- $D(v)$: custo do caminho de menor custo entre o nó de origem e o destino v até essa iteração do algoritmo.
- $p(v)$: nó anterior (vizinho de v) ao longo do caminho de menor custo corrente desde a origem até v .
- N' : subconjunto de nós; v pertence a N' se o caminho de menor custo entre a origem e v for inequivocamente conhecido.

O algoritmo de roteamento centralizado consiste em um passo de inicialização seguido por um loop. O número de vezes que o loop é executado é igual ao número de nós da rede. Ao final, o algoritmo terá calculado os caminhos mais curtos do nó de origem u até todos os outros nós da rede.

Algoritmo de estado de enlace para o nó de origem u

```

1  Inicialização:
2   $N' = \{u\}$ 
3  para todos os nós  $v$ 
4    se  $v$  for um vizinho de  $u$ 
5      então  $D(v) = c(u,v)$ 
6      senão  $D(v) = \infty$ 
7
8  Loop
9  encontre  $w$  não em  $N'$  tal que  $D(w)$  é um mínimo
10 adicione  $w$  a  $N'$ 
11 atualize  $D(v)$  para cada vizinho  $v$  de  $w$  e não em  $N'$ :
12    $D(v) = \min(D(v), D(w)+c(w,v))$ 
13 /* o novo custo para  $v$  é o velho custo para  $v$  ou o custo do
14 menor caminho conhecido para  $w$  mais o custo de  $w$  para  $v$  */
15 até  $N' = N$ 

```

Como exemplo, vamos considerar a rede da Figura 5.3 e calcular os caminhos de menor custo de u até todos os destinos possíveis. Os cálculos do algoritmo estão resumidos na Tabela 5.1, na qual cada linha fornece os valores das variáveis do algoritmo ao final da iteração. Vamos examinar detalhadamente alguns dos primeiros estágios:

- No estágio de inicialização, os caminhos de menor custo mais conhecidos de u até os vizinhos diretamente ligados a ele (v , x e w) são inicializados em 2, 1 e 5, respectivamente. Note, em particular, que o custo até w é estabelecido em 5 (embora logo veremos que, na realidade, existe um trajeto cujo custo é ainda menor), já que este é o custo do enlace (um salto) direto u a w . Os custos até y e z são estabelecidos como infinito, porque eles não estão diretamente conectados a u .
- Na primeira iteração, examinamos os nós que ainda não foram adicionados ao conjunto N' e descobrimos o nó de menor custo ao final da iteração anterior. Este é o nó x , com um custo de 1, e, assim, x é adicionado ao conjunto N' . A linha 12 do algoritmo de vetor de distâncias (LS) é então rodada para atualizar $D(v)$ para todos os nós v , produzindo os resultados mostrados na segunda linha (Etapa 1) da Tabela 5.1. O custo do caminho até v não muda. Descobriremos que o custo do caminho até w pelo nó x (que era 5 ao final da inicialização) é 4. Por conseguinte, esse caminho de custo mais baixo é selecionado e o predecessor de w ao longo do caminho mais curto a partir de u é definido como x . De maneira semelhante, o custo até y (através de x) é calculado como 2, e a tabela é atualizada de acordo com isso.
- Na segunda iteração, verificamos que os nós v e y são os que têm os caminhos de menor custo (2); decidimos o empate arbitrariamente e adicionamos y ao conjunto N' de modo que N' agora contém u , x e y . O custo dos nós remanescentes que ainda não estão em N' (i.e., nós v , w e z) são atualizados pela linha 12 do algoritmo LS, produzindo os resultados mostrados na terceira linha da Tabela 5.1.
- E assim por diante...

Quando o algoritmo LS termina, temos, para cada nó, seu predecessor ao longo do caminho de menor custo a partir do nó de origem. Temos também o predecessor para cada um deles; assim, podemos construir o caminho inteiro da origem até todos os destinos. Então, a tabela de repasse em um nó, por exemplo, u , pode ser construída a partir dessas informações, armazenando, para cada destino, o nó do salto seguinte no caminho de menor custo de u até o destino. A Figura 5.4 mostra os caminhos de menor custo resultantes e a tabela de repasse em u para a rede na Figura 5.3.

Qual é a complexidade do cálculo desse algoritmo? Isto é, dados n nós (sem contar a origem), quanto cálculo é preciso efetuar, no pior caso, para descobrir os caminhos de menor custo entre a origem e todos os destinos? Na primeira iteração, precisamos pesquisar todos os n nós para determinar o nó w , que não está em N' , e que tem o custo mínimo. Na segunda, temos de verificar $n - 1$ nós para determinar o custo mínimo. Na terceira, $n - 2$ nós. E assim por diante. Em termos gerais, o número total de nós que precisamos pesquisar em todas as

TABELA 5.1 Execução do algoritmo de estado de enlace na rede da Figura 5.3

Passo	N'	$D(v), p(v)$	$D(w), p(w)$	$D(x), p(x)$	$D(y), p(y)$	$D(z), p(z)$
0	u	2, u	5, u	1, u	∞	∞
1	ux	2, u	4, x		2, x	∞
2	uxy	2, u	3, y			4, y
3	$uxyv$		3, y			4, y
4	$uxyvw$					4, y
5	$uxyvwz$					

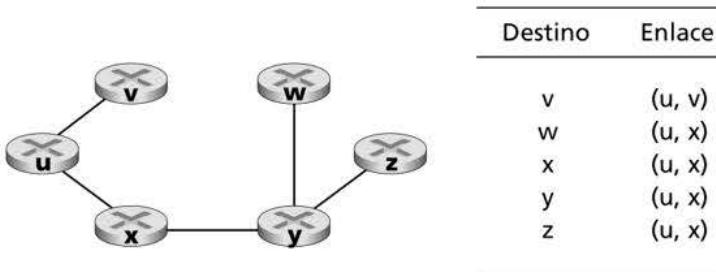


Figura 5.4 Caminhos de menor custo resultantes e tabela de repasse para o nó u.

iterações é $n(n + 1)/2$, e, assim, dizemos que a complexidade da implementação do algoritmo de estado de enlace para o pior caso é de ordem n ao quadrado: $O(n^2)$. (Uma execução mais sofisticada, que utiliza uma estrutura de dados conhecida como pilha, pode descobrir o mínimo na linha 9 em tempo logarítmico e não linear, reduzindo assim a complexidade.)

Antes de concluirmos nossa discussão sobre o algoritmo LS, vamos considerar uma patologia que pode surgir. A Figura 5.5 mostra uma topologia de rede simples em que os custos dos enlaces são iguais à carga transportada pelo enlace, refletindo, por exemplo, o atraso que seria experimentado. Nesse exemplo, os custos dos enlaces não são simétricos, isto é, $c(u,v)$ é igual a $c(v,u)$ apenas se a carga transportada em ambas as direções do enlace (u,v) for a mesma. Nesse exemplo, o nó z origina uma unidade de tráfego destinada a w, o nó x também origina uma unidade de tráfego destinada a w, e o nó y injeta uma quantidade de tráfego igual a e , também destinada a w. O roteamento inicial é mostrado na Figura 5.5(a) com os custos dos enlaces correspondentes à quantidade de tráfego transportada.

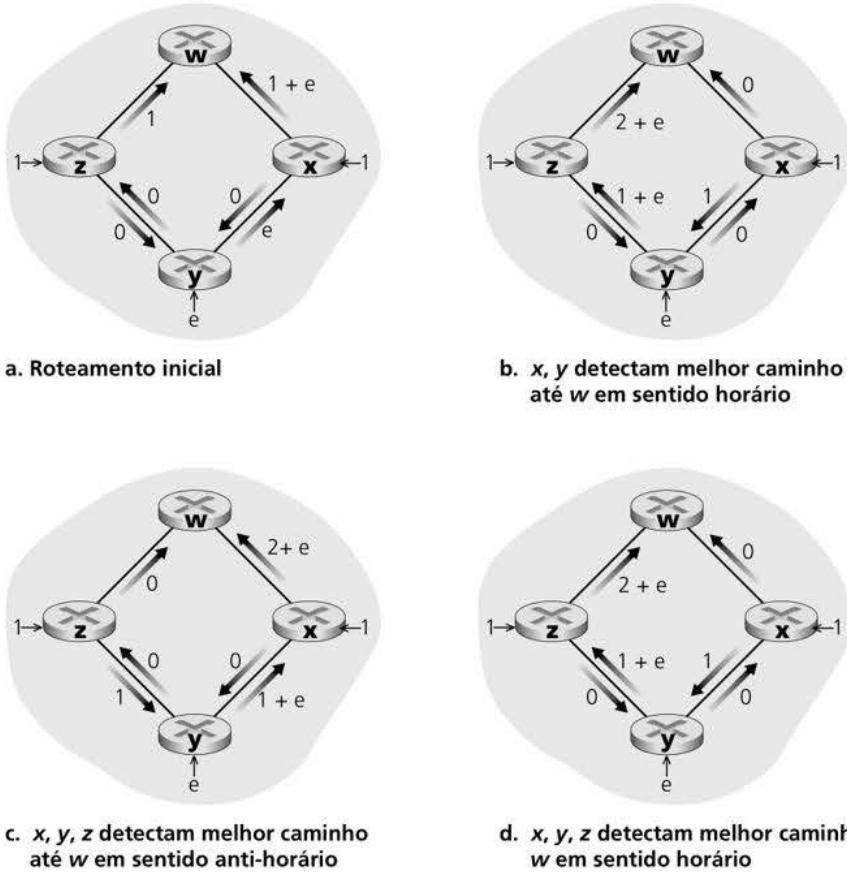


Figura 5.5 Oscilações com roteamento sensível ao congestionamento.

Quando o algoritmo LS é rodado de novo, o nó y determina – com base nos custos dos enlaces mostrados na Figura 5.5(a) – que o caminho em sentido horário até w tem um custo de 1, ao passo que o caminho em sentido anti-horário até w (que estava sendo usado) tem o custo de $1 + e$. Por conseguinte, o caminho de menor custo de y até w é agora em sentido horário. De maneira semelhante, x determina que seu novo caminho de menor custo até w é também em sentido horário, resultando nos custos mostrados na Figura 5.5(b). Na próxima vez em que o algoritmo LS é rodado, os nós x , y e z detectam um caminho de custo zero até w na direção anti-horária, e todos dirigem seu tráfego para as rotas anti-horárias. Na próxima vez em que o algoritmo LS é rodado, os nós x , y e z então dirigem seu tráfego para as rotas em sentido horário.

O que pode ser feito para evitar essas oscilações (que podem ocorrer com qualquer algoritmo, e não apenas com um algoritmo LS, que use uma métrica de enlace baseada em congestionamento ou em atraso)? Uma solução seria tornar obrigatório que os custos dos enlaces não dependessem da quantidade de tráfego transportada – uma solução inaceitável, já que um dos objetivos do roteamento é evitar enlaces muito congestionados (p. ex., enlaces com grande atraso). Outra solução seria assegurar que nem todos os roteadores rodassem o algoritmo LS ao mesmo tempo. Esta parece ser uma solução mais razoável, já que é de esperar que, mesmo que os roteadores rodem o algoritmo LS com idêntica periodicidade, o instante de execução do algoritmo não seja o mesmo em cada nó. O interessante é que os pesquisadores descobriram que os roteadores da Internet podem se autossincronizar (Floyd Synchronization, 1994). Isto é, mesmo que inicialmente rodem o algoritmo com o mesmo período, mas em diferentes momentos, a instância de execução do algoritmo pode finalmente se tornar, e permanecer, sincronizada nos roteadores. Um modo de evitar essa autossincronização é cada roteador variar aleatoriamente o instante em que envia um anúncio de enlace.

Agora que examinamos o algoritmo de estado de enlace, vamos analisar outro importante algoritmo usado hoje na prática – o algoritmo de roteamento de vetor de distâncias.*

5.2.2 O algoritmo de roteamento de vetor de distâncias (DV)

Enquanto o algoritmo LS usa informação global, o algoritmo de **DV** é iterativo, assíncrono e distribuído. É *distribuído* porque cada nó recebe alguma informação de um ou mais vizinhos *diretamente ligados* a ele, realiza cálculos e, em seguida, distribui os resultados de seus cálculos para seus vizinhos. É *iterativo* porque esse processo continua até que nenhuma informação seja trocada entre vizinhos. (O interessante é que este é um algoritmo finito – não há nenhum sinal de que o cálculo deve parar; ele apenas para.) O algoritmo é *assíncrono* porque não requer que todos os nós rodem simultaneamente. Veremos que um algoritmo assíncrono, iterativo, finito e distribuído é muito mais interessante e divertido do que um algoritmo centralizado!

Antes de apresentar o algoritmo DV, é bom discutir uma relação importante que existe entre os custos dos caminhos de menor custo. Seja $d_x(y)$ o custo do caminho de menor custo do nó x ao nó y . Então, os menores custos estão relacionados segundo a famosa equação de Bellman-Ford:

$$d_x(y) = \min_v \{c(x, v) + d_v(y)\}, \quad (5.1)$$

sendo o \min_v da equação calculado para todos os vizinhos de x . A equação de Bellman-Ford é bastante intuitiva. Realmente, se após transitarmos de x para v tomarmos o caminho de menor custo de v a y , o custo do caminho será $c(x, v) + d_v(y)$. Como devemos começar viajando até algum vizinho v , o caminho de menor custo de x a y é o mínimo do conjunto dos $c(x, v) + d_v(y)$ calculados para todos os vizinhos v .

*N. de T.: O problema exposto aqui é teórico. Protocolos LS como o OSPF não usam a carga nos enlaces em suas principais métricas de custo.

Mas para aqueles que ainda se mostrem céticos quanto à validade da equação, vamos verificá-la para o nó de origem u e o nó de destino z na Figura 5.3. O nó de origem u tem três vizinhos: nós v , x e w . Percorrendo vários caminhos no grafo, é fácil ver que $d_v(z) = 5$, $d_x(z) = 3$ e $d_w(z) = 3$. Passando esses valores para a Equação 5.1, junto com os custos $c(u, v) = 2$, $c(u, x) = 1$ e $c(u, w) = 5$, temos $d_u(z) = \min\{2 + 5, 5 + 3, 1 + 3\} = 4$, que é, claro, verdade e que é, exatamente, o resultado conseguido com o algoritmo de Dijkstra para a mesma rede. Essa verificação rápida deve ajudá-lo a vencer qualquer ceticismo que ainda possa ter.

A equação de Bellman-Ford não é apenas uma curiosidade intelectual. Na verdade, ela tem uma importância prática significativa: sua solução fornece os registros da tabela de repasse do nó x . Para verificar, seja v^* qualquer nó vizinho que represente o mínimo na Equação 5.1. Então, se o nó x quiser enviar um pacote ao nó y pelo caminho de menor custo, deverá, primeiro, repassá-lo para o nó v^* . Assim, a tabela de repasse do nó x especificaria o nó v^* como o roteador do próximo salto para o destino final y . Outra contribuição importante dessa equação é que ela sugere a forma da comunicação vizinho para vizinho que ocorrerá no algoritmo DV.

A ideia básica é a seguinte. Cada nó x começa com $D_x(y)$, uma estimativa do custo do caminho de menor custo entre ele mesmo e o nó y , para todos os nós, y , em N . Seja $\mathbf{D}_x = [D_x(y): y \in N]$ o vetor de distâncias do nó x , que é o vetor de estimativas de custo de x até todos os outros nós, y , em N . Com o algoritmo DV, cada nó x mantém os seguintes dados de roteamento:

- Para cada vizinho v , o custo $c(x, v)$ de x até o vizinho diretamente ligado a ele, v ;
- O vetor de distâncias do nó x , isto é, $\mathbf{D}_x = [D_x(y): y \in N]$, contendo a estimativa de x para seus custos até todos os destinos, y , em N ;
- Os vetores de distâncias de seus vizinhos, isto é, $\mathbf{D}_v = [D_v(y): y \in N]$ para cada vizinho v de x .

No algoritmo distribuído, assíncrono, cada nó envia, a intervalos regulares, uma cópia do seu vetor de distâncias a cada um de seus vizinhos. Quando um nó x recebe um novo vetor de distâncias de qualquer de seus vizinhos w , ele armazena o vetor de distâncias de w e então usa a equação de Bellman-Ford para atualizar seu próprio vetor de distâncias, como a seguir:

$$D_x(y) = \min_v \{c(x, v) + D_v(y)\} \quad \text{para cada nó } y \text{ em } N$$

Se o vetor de distâncias do nó x tiver mudado como resultado dessa etapa de atualização, o nó x então enviará seu vetor de distâncias atualizado para cada um de seus vizinhos que, por sua vez, podem atualizar seus próprios vetores de distâncias. Parece milagre, mas contanto que todos os nós continuem a trocar seus vetores de distâncias de forma assíncrona, cada estimativa de custo $D_x(y)$ convergirá para $d_x(y)$, que é, na verdade, o custo do caminho de menor custo do nó x ao nó y (Bertsekas, 1991)!

Algoritmo de vetor de distâncias (DV)

Para cada nó, x :

```

1   Inicialização:
2   para todos os destinos  $y$  em  $N$ :
3        $D_x(y) = c(x, y) /*$  se  $y$  não é um vizinho então  $c(x, y) = \infty */$ 
4   para cada vizinho  $w$ 
5        $D_w(y) = ?$  para todos os destinos  $y$  em  $N$ 
6   para cada vizinho  $w$ 
7       envia vetor de distâncias  $\mathbf{D}_x = [D_x(y): y \in N]$  para  $w$ 
8

```

```

9   loop
10    espere (até que ocorra uma mudança no custo do enlace ao vizinho
11        w ou até a recepção de um vetor de distâncias do vizinho w)
12
13   para cada y em N:
14      $D_x(y) = \min_v \{c(x,v) + D_v(y)\}$ 
15
16   se  $D_x(y)$  mudou para algum destino y
17     envia vetor de distâncias  $\mathbf{D}_x = [D_x(y) : y \in N]$  para todos os vizinhos
18
19   para sempre

```

No algoritmo DV, um nó x atualiza sua estimativa do vetor de distâncias quando percebe uma mudança de custo em um dos enlaces ligados diretamente a ele ou recebe uma atualização do vetor de distâncias de algum vizinho. Mas para atualizar sua própria tabela de repasse para um dado destino y , o que o nó x de fato precisa saber não é a distância do caminho mais curto até y , mas qual nó vizinho $v^*(y)$ é o roteador do próximo salto ao longo do caminho mais curto até y . Como era de se esperar, o roteador do próximo salto $v^*(y)$ é o vizinho v que representar o mínimo na Linha 14 do algoritmo DV. (Se houver vários vizinhos v que representem o mínimo, então $v^*(y)$ pode ser qualquer um dos vizinhos minimizadores.) Assim, nas Linhas 13 a 14, para cada destino y , o nó x também determina $v^*(y)$ e atualiza sua tabela de repasse para o destino y .

Lembre-se de que o algoritmo LS é um algoritmo global no sentido de que requer que cada nó obtenha, primeiro, um mapa completo da rede antes de rodar o algoritmo de Dijkstra. O algoritmo DV é *descentralizado* e não usa essa informação global. De fato, a única informação que um nó terá são os custos dos enlaces até os vizinhos diretamente ligados a ele e as informações que recebe desses vizinhos. Cada nó espera uma atualização de qualquer vizinho (Linhas 10–11), calcula seu novo vetor de distâncias ao receber uma atualização (Linha 14), e distribui seu novo vetor de distâncias a seus vizinhos (Linhas 16–17). Algoritmos semelhantes ao DV são utilizados em muitos protocolos de roteamento na prática, entre eles o RIP e o BGP da Internet, o ISO IDRP, o IPX da Novell e o ARPAnet original.

A Figura 5.6 ilustra a operação do algoritmo DV para a rede simples de três nós mostrada na parte superior da figura. A operação do algoritmo é ilustrada de um modo síncrono, no qual todos os nós recebem vetores de distâncias simultaneamente de seus vizinhos, calculam seus novos vetores de distâncias, e informam a seus vizinhos se esses vetores mudaram. Após estudar esse exemplo, você deve se convencer de que o algoritmo também opera corretamente em modo assíncrono, com cálculos de nós e atualizações de geração/recepção ocorrendo a qualquer instante.

A coluna mais à esquerda na figura mostra três **tabelas de roteamento** iniciais para cada um dos três nós. Por exemplo, a tabela no canto superior à esquerda é a tabela de roteamento inicial do nó x . Dentro de uma tabela de roteamento específica, cada linha é um vetor de distâncias – em especial, a tabela de roteamento de cada nó inclui seu próprio vetor de distâncias e os vetores de cada um de seus vizinhos. Assim, a primeira linha da tabela de roteamento inicial do nó x é $\mathbf{D}_x = [D_x(x), D_x(y), D_x(z)] = [0, 2, 7]$. A segunda linha e a terceira linha nessa tabela são os vetores de distâncias recebidos mais recentemente dos nós y e z . Como na inicialização o nó x não recebeu nada do nó y ou z , os registros da segunda linha e da terceira linha estão definidos como infinito.

Após a inicialização, cada nó envia seu vetor de distâncias a cada um de seus dois vizinhos. Isso é ilustrado na Figura 5.6 pelas setas que vão da primeira coluna de tabelas até a segunda. Por exemplo, o nó x envia seu vetor de distâncias $\mathbf{D}_x = [0, 2, 7]$ a ambos os nós

y e z . Após receber as atualizações, cada nó recalcula seu próprio vetor de distâncias. Por exemplo, o nó x calcula

$$D_x(x) = 0$$

$$D_x(y) = \min\{c(x,y) + D_y(y), c(x,z) + D_z(z)\} = \min\{2 + 0, 7 + 1\} = 2$$

$$D_x(z) = \min\{c(x,y) + D_y(z), c(x,z) + D_z(z)\} = \min\{2 + 1, 7 + 0\} = 3$$

Por conseguinte, a segunda coluna mostra, para cada nó, o novo vetor de distâncias do nó, junto com os vetores de distâncias que acabou de receber de seus vizinhos. Observe, por exemplo, que a estimativa do nó x para o menor custo até o nó z , $D_x(z)$, mudou de 7 para 3. Note também que, para o nó x , o nó vizinho y alcança o mínimo na linha 14 do algoritmo DV; assim, nesse estágio do algoritmo, temos que, no nó x , $v^*(y) = y$ e $v^*(z) = y$.

Depois que recalculam seus vetores de distâncias, os nós enviam novamente seus vetores de distâncias recalculados a seus vizinhos (se houver uma mudança). Isso é ilustrado na Figura 5.6 pelas setas que vão da segunda até a terceira coluna de tabelas. Note que apenas os nós x e z enviam atualizações: o vetor de distâncias do nó y não mudou, então esse nó não envia uma atualização. Após receber-las, os nós então recalculam seus vetores de distâncias e atualizam suas tabelas de roteamento, que são mostradas na terceira coluna.

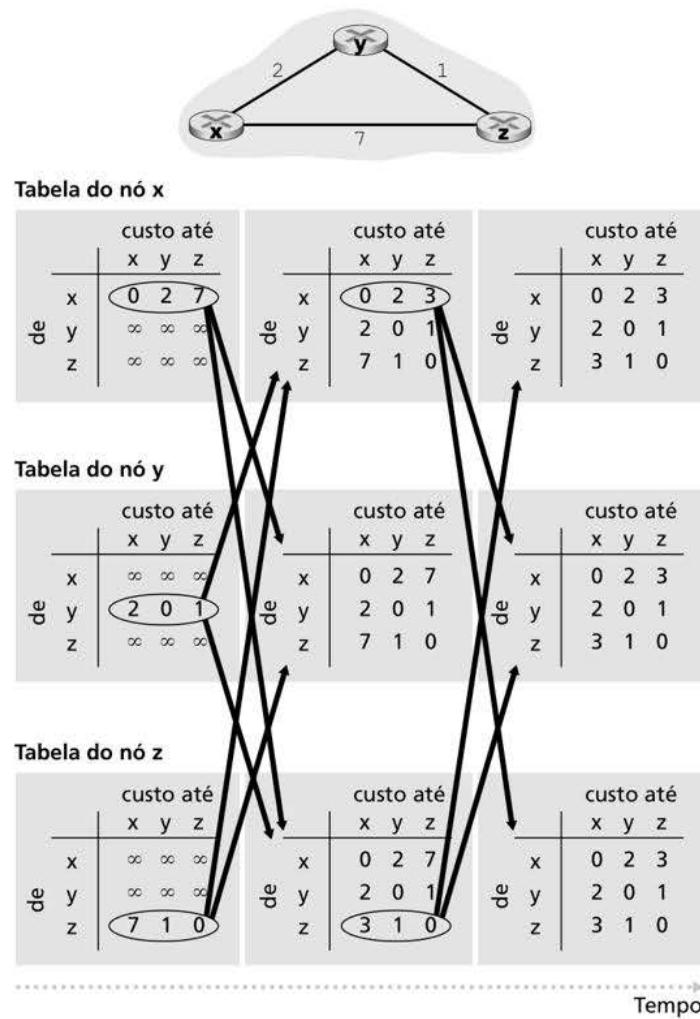


Figura 5.6 Algoritmo de vetor de distâncias (DV).

O processo de receber vetores de distâncias atualizados de vizinhos, recalcular os registros de tabelas de roteamento e informar aos vizinhos os custos modificados do caminho de menor custo até o destino continua até que mais nenhuma mensagem de atualização seja enviada. Nesse ponto, não ocorrerá mais nenhum cálculo de tabela de roteamento e o algoritmo entra em estado de inatividade; isto é, todos os nós estarão realizando a espera nas Linhas 10 a 11 do algoritmo DV. Este permanece no estado de inatividade até que o custo de um enlace mude, como veremos a seguir.

Algoritmo de vetor de distâncias: mudanças no custo do enlace e falha no enlace

Quando um nó que está rodando o algoritmo DV detecta uma mudança no custo do enlace dele mesmo até um vizinho (Linhas 10-11), ele atualiza seu vetor de distâncias (Linhas 13-14) e, se houver uma modificação no custo do caminho de menor custo, informa a seus vizinhos (Linhas 16-17) seu novo vetor de distâncias. A Figura 5.7(a) ilustra um cenário em que o custo do enlace de y a x muda de 4 para 1. Destacamos aqui somente os registros na tabela de distâncias de y e z até o destino x . O algoritmo DV faz ocorrer a seguinte sequência de eventos:

- No tempo t_0 , y detecta a mudança no custo do enlace (o custo mudou de 4 para 1), atualiza seu vetor de distâncias e informa essa mudança a seus vizinhos, já que o vetor mudou.
- No tempo t_1 , z recebe a atualização de y e atualiza sua própria tabela. Calcula um novo menor custo para x (cujo custo diminuiu de 5 para 2) e envia seu novo vetor de distâncias a seus vizinhos.
- No tempo t_2 , y recebe a atualização de z e atualiza sua tabela de distâncias. Os menores custos de y não mudaram, e, por conseguinte, y não envia nenhuma mensagem a z . O algoritmo entra em estado de inatividade.

Assim, apenas duas iterações são necessárias para o algoritmo DV alcançar o estado de inatividade. A boa notícia sobre a redução do custo entre x e y se propagou rapidamente pela rede.

Agora vamos considerar o que pode acontecer quando o custo de um enlace *aumenta*. Suponha que o custo do enlace entre x e y aumente de 4 para 60, conforme mostra a Figura 5.7(b).

1. Antes da mudança do custo do enlace, $D_y(x) = 4$, $D_y(z) = 1$, $D_z(y) = 1$ e $D_z(x) = 5$. No tempo t_0 , y detecta uma mudança no custo do enlace (o custo mudou de 4 para 60). y calcula seu novo caminho de custo mínimo até x , de modo a ter um custo de

$$D_y(x) = \min\{c(y, x) + D_x(x), c(y, z) + D_z(x)\} = \min\{60 + 0, 1 + 5\} = 6$$

É claro que, com nossa visão global da rede, podemos ver que esse novo custo via z está errado. Mas as únicas informações que o nó y tem é que seu custo direto até x é 60, e que z disse a y que z pode chegar a x com um custo de 5. Assim, para chegar a x , y teria

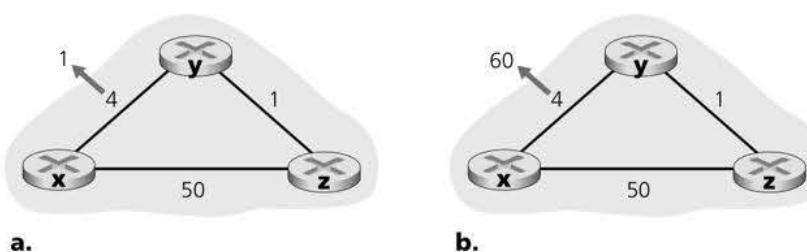


Figura 5.7 Mudanças no custo do enlace.

de fazer a rota através de z , com a expectativa de que z será capaz de chegar a x com um custo de 5. A partir de t_1 , temos um **loop de roteamento** – para poder chegar a x , y faz a rota através de z , que, por sua vez, faz a rota através de y . Um loop de roteamento é como um buraco negro – um pacote destinado a x que ao chegar a y ou a z a partir do momento t_1 vai ricochetear entre esses dois nós para sempre (ou até que as tabelas de repasse sejam mudadas).

2. Tão logo o nó y tenha calculado um novo custo mínimo até x , ele informará a z esse novo vetor de distâncias no tempo t_1 .
3. Algum tempo depois de t_1 , z recebe o novo vetor de distâncias de y , que indica que o custo mínimo de y até x é 6. z sabe que pode chegar até y com um custo de 1, e, por conseguinte, calcula um novo menor custo até x , $D_z(x) = \min\{50 + 0,1 + 6\} = 7$. Uma vez que o custo mínimo de z até x aumentou, z informa a y o seu novo vetor de distâncias em t_2 .
4. De maneira semelhante, após receber o novo vetor de distâncias de z , y determina $D_y(x) = 8$ e envia a z seu vetor de distâncias. Então z determina $D_z(x) = 9$ e envia a y seu vetor de distâncias e assim por diante.

Por quanto tempo esse processo continua? Pode ter certeza de que o loop persistirá por 44 iterações (trocas de mensagens entre y e z) até que z possa, enfim, calcular que o custo de seu caminho via y é maior do que 50. Nesse ponto, z (enfim!) determinará que seu caminho de menor custo até x é via sua conexão direta com x . Então, y fará a rota até x via z . O resultado das más notícias sobre o aumento do custo do enlace na verdade viajou devagar! O que teria acontecido se o custo do enlace $c(y, x)$ tivesse mudado de 4 para 10.000 e o custo $c(z, x)$ fosse 9.999? Em virtude de cenários como esses, o problema que acabamos de examinar é, às vezes, denominado problema de contagem ao infinito.

Algoritmo de vetor de distâncias: adição de reversão envenenada

O cenário específico de contagem ao infinito que acabamos de descrever pode ser evitado usando uma técnica denominada *reversão envenenada* (*Poisoned reverse*). A ideia é simples – se a rota de z para chegar a x passa por y , então z deve anunciar a y que sua distância a x é infinita, isto é, z anunciará a y que $D_z(x) = \infty$ (mesmo que z saiba que, na verdade, $D_z(x) = 5$). z continuará contando essa mentirinha inocente a y enquanto a rota de z a x estiver passando por y . Enquanto y acreditar que z não tem nenhum caminho até x , y jamais tentará a rota até x por z , contanto que a rota de z a x continue a passar por y (e ele minta sobre isso).

Agora vamos ver como a reversão envenenada resolve o problema específico da contagem ao infinito que encontramos antes na Figura 5.5(b). Como resultado da reversão envenenada, a tabela de distâncias de y indica que $D_z(x) = \infty$. Quando o custo do enlace (x, y) muda de 4 para 60 no tempo t_0 , y atualiza sua tabela e continua a estabelecer rotas diretamente para x , embora com um custo mais alto do que 60, e informa a z o seu novo custo até x , isto é, $D_y(x) = 60$. Após receber a atualização em t_1 , z imediatamente desloca sua rota para x , para que passe pelo enlace direto (z, x) a um custo de 50. Como este é um novo menor custo até x , e já que o caminho não passa mais por y , z agora informa a y que $D_z(x) = 50$ em t_2 . Após receber a atualização de z , y atualiza sua tabela de distâncias com $D_z(x) = 51$. E, também, como z está agora no caminho de menor custo de y até x , y envenena o caminho inverso de z a x , informando a z , no tempo t_3 , que $D_y(x) = \infty$ (mesmo que y saiba que, na verdade, $D_y(x) = 51$).

A reversão envenenada resolve o problema geral da contagem até o infinito? Não resolve. É bom que você se convença de que loops que envolvem três ou mais nós (e não apenas dois nós imediatamente vizinhos) não serão detectados pela técnica da reversão envenenada.

Uma comparação entre os algoritmos de roteamento de estado de enlace (LS) e de vetor de distâncias (DV)

Os algoritmos DV e LS adotam abordagens complementares em relação ao cálculo do roteamento. No algoritmo DV, cada nó fala *somente* com os vizinhos diretamente conectados

a ele, mas informa a esses vizinhos as estimativas de menor custo entre ele mesmo e *todos* os outros nós da rede (i.e., todos os que ele sabe que existem). O algoritmo LS exige informações globais. Por consequência, quando implementado em todos os roteadores, como nos exemplos das Figuras 4.2 e 5.1, cada nó precisaria se comunicar com *todos* os outros nós (por difusão), mas informar *somente* os custos dos enlaces diretamente ligados a ele. Vamos concluir nosso estudo sobre algoritmos de estado de enlace e de vetor de distâncias com uma rápida comparação de alguns de seus atributos. Lembre-se de que N é o conjunto de nós (roteadores) e E é o conjunto de arestas (enlaces).

- *Complexidade da mensagem.* Vimos que o LS requer que cada nó saiba o custo de cada enlace da rede. Isso exige que sejam enviadas $O(|N| |E|)$ mensagens. E, também, sempre que o custo de um enlace muda, o novo custo deve ser enviado a todos os nós. O algoritmo DV requer troca de mensagens entre vizinhos diretamente conectados a cada iteração. Já vimos que o tempo necessário para que o algoritmo converja pode depender de muitos fatores. Quando o custo do enlace muda, o algoritmo DV propaga os resultados do custo modificado do enlace apenas se o novo custo resultar em mudança no caminho de menor custo para um dos nós ligado ao enlace.
- *Velocidade de convergência.* Já vimos que nossa implementação de LS é um algoritmo $O(|N|^2)$ que requer $O(|N| |E|)$ mensagens. O algoritmo DV pode convergir lentamente e pode ter loops de roteamento enquanto estiver convergindo. O algoritmo DV também tem o problema da contagem até o infinito.
- *Robustez.* O que pode acontecer se um roteador falhar, se comportar mal ou for sabotado? Sob o LS, um roteador poderia transmitir um custo incorreto para um de seus enlaces diretos (mas não para outros). Um nó poderia também corromper ou descartar quaisquer pacotes recebidos como parte de uma difusão de estado de enlace. Mas um nó LS está calculando apenas suas próprias tabelas de roteamento; os outros nós estão realizando cálculos semelhantes para si próprios. Isso significa que, sob o LS, os cálculos de rota são, de certa forma, isolados, fornecendo um grau de robustez. Sob o DV, um nó pode anunciar incorretamente caminhos de menor custo para qualquer destino, ou para todos os destinos. (Na verdade, em 1997, um roteador que estava funcionando mal em um pequeno ISP forneceu aos roteadores nacionais de backbone tabelas de roteamento errôneas. Isso fez outros roteadores inundarem de tráfego o roteador que estava funcionando mal. Com isso, grandes porções da Internet ficaram desconectadas durante muitas horas [Neumann, 1997].) De modo geral, notamos que, a cada iteração, um cálculo de nó em DV é passado adiante a seu vizinho e, em seguida, indiretamente ao vizinho de seu vizinho na iteração seguinte.

Nesse sentido, sob o DV, um cálculo incorreto do nó pode ser difundido pela rede inteira. No final, nenhum algoritmo ganha do outro; na verdade, ambos são usados na Internet.

5.3 ROTEAMENTO INTRA-AS NA INTERNET: OSPF

Quando estudamos os algoritmos de roteamento, consideramos a rede apenas como uma coleção de roteadores interconectados. Um roteador não se distinguia de outro no sentido de que todos rodavam o mesmo algoritmo de roteamento para calcular os caminhos de roteamento pela rede inteira. Na prática, esse modelo e sua visão de um conjunto homogêneo de roteadores, todos rodando o mesmo algoritmo de roteamento, é simplista por duas razões importantes:

- *Escala.* À medida que aumenta o número de roteadores, a sobrecarga relativa à comunicação, ao cálculo e ao armazenamento e de informações de roteamento se torna proibitiva. A Internet pública de hoje consiste em centenas de milhões de roteadores. Armazenar informações de roteamento para destinos possíveis em cada um desses roteadores

evidentemente exigiria quantidades enormes de memória. O custo fixo para transmitir atualizações sobre o estado de enlace e os custos dos enlaces entre todos os roteadores da Internet seria enorme! Um algoritmo DV que fizesse iterações entre esse número tão grande de roteadores decerto jamais convergiria! Fica claro que algo deve ser feito para reduzir a complexidade do cálculo de rotas em redes tão grandes como a Internet pública.

- *Autonomia administrativa.* Como descrito na Seção 1.3, a Internet é uma rede de ISPs, com cada ISP composto por sua própria rede de roteadores. Em geral, o ISP quer operar a sua rede como bem entende (p. ex., executar os algoritmos de roteamento que quiser dentro da sua rede) ou ocultar do mundo exterior aspectos da organização interna da sua rede. Em mundo ideal, a organização deveria poder operar e administrar sua rede como quisesse ao mesmo tempo que ainda seria capaz de conectar a sua rede a outras redes externas.

Esses problemas podem ser resolvidos agrupando roteadores em **sistemas autônomos** (ASs, do inglês *autonomous systems*), com cada AS consistindo em um grupo de roteadores sob o mesmo controle administrativo. Muitas vezes, os roteadores em um ISP e os enlaces que os interconectam constituem um único AS. Alguns ISPs, no entanto, dividem sua rede em vários ASs. Em especial, alguns ISPs de nível 1 utilizam um AS para toda a sua rede; outros a subdividem em dezenas de ASs. Um sistema autônomo é identificado por seu número de sistema autônomo (ASN, do inglês *autonomous system number*) globalmente exclusivo (RFC 1930). Números de ASs, como endereços IP, são designados por entidades regionais ICANN de registro (ICANN, 2020).

Todos os roteadores dentro do mesmo AS rodam o mesmo algoritmo de roteamento e dispõem das informações sobre cada um dos outros. O algoritmo de roteamento que roda dentro de um AS é denominado um **protocolo de roteamento intrassistema autônomo**.

Open Shortest Path First (OSPF)

O OSPF e seu primo, IS-IS, muito parecido com ele, são amplamente utilizados para roteamento intra-As na Internet. O *Open* do OSPF significa que as especificações do protocolo de roteamento estão abertas ao público (ao contrário do protocolo EIGRP da Cisco, p. ex., que só foi aberto recentemente (Savage, 2015), após cerca de 20 anos como protocolo proprietário da Cisco). A versão mais recente do OSPF, versão 2, está definida no RFC 2328, um documento público.

O OSPF é um protocolo de estado de enlace que usa inundação de informação de estado de enlace e um algoritmo de caminho de menor custo de Dijkstra. Com o OSPF, um roteador constrói um mapa topológico completo (i.e., um grafo) de todo o sistema autônomo. O roteador então roda localmente o algoritmo do caminho mais curto de Dijkstra para determinar uma árvore de caminho mais curto para todas as *sub-redes*, sendo ele próprio o nó raiz. Os custos de enlaces individuais são configurados pelo administrador da rede (veja “Princípios na prática: Configurando os pesos de enlaces no OSPF”). O administrador pode optar por estabelecer todos os custos de enlace em 1, conseguindo assim o roteamento com o mínimo de saltos, ou por designar para os enlaces pesos inversamente proporcionais à capacidade do enlace, de modo a desencorajar o tráfego a usar enlaces de largura de banda baixa. O OSPF não impõe uma política para o modo como são determinados os pesos dos enlaces (essa tarefa é do administrador da rede); em vez disso, oferece os mecanismos (protocolo) para determinar o caminho de roteamento de menor custo para um dado conjunto de pesos de enlaces.

Com OSPF, um roteador transmite por difusão informações de roteamento a *todos* os outros roteadores no sistema autônomo, não apenas a seus vizinhos. Um roteador transmite informações de estado de enlace por difusão sempre que houver uma mudança no estado de um enlace (p. ex., uma mudança de custo ou uma mudança de estado para cima/para baixo). Também transmite o estado de um enlace periodicamente (pelo menos a cada 30 minutos), mesmo que não tenha havido mudança. O RFC 2328 observa que “essa atualização

PRINCÍPIOS NA PRÁTICA

CONFIGURANDO OS PESOS DE ENLACES NO OSPF

Nossa discussão sobre o roteamento de estado de enlace pressupõe que os pesos dos enlaces são fixos, que um algoritmo de roteamento como o OSPF é executado, e que o tráfego flui de acordo com as tabelas de roteamento calculadas pelo algoritmo de estado de enlace. Em termos de causa e efeito, os pesos dos enlaces são dados (i.e., vêm primeiro) e resultam (por meio do algoritmo de Dijkstra) nos caminhos de roteamento que minimizam o custo total. Por esse ponto de vista, os pesos dos enlaces refletem o custo de se utilizar um enlace (p. ex., se os pesos dos enlaces são inversamente proporcionais à capacidade, o uso de enlaces de alta capacidade teria um peso menor e, logo, seriam mais atraentes do ponto de vista do roteamento) e o algoritmo de Dijkstra serve para minimizar o custo total.

Na prática, a relação de causa e efeito entre os pesos dos enlaces e os caminhos de roteamento pode ser invertida, com os operadores da rede configurando os

pesos de enlaces para obter caminhos de roteamento que cumprem determinadas metas de engenharia de tráfego (Fortz, 2000; Fortz, 2002). Por exemplo, suponha que um operador de rede possua uma estimativa de fluxo de tráfego de entrada em cada ponto de ingresso e destinado a cada ponto de egresso. O operador poderia então querer instalar um roteamento específico dos fluxos de ingresso para egresso que minimizasse a utilização máxima nos enlaces da rede como um todo. Mas com um algoritmo de roteamento como o OSPF, os principais “botões” do operador para fazer a sintonia fina do roteamento de fluxos pela rede são os pesos de enlaces. Assim, para cumprir o objetivo de minimizar a utilização de enlaces máximas, o operador precisa encontrar um conjunto de pesos de enlaces que atenda aos seus critérios. É uma inversão da relação de causa e efeito: o roteamento desejado dos fluxos é conhecido, e precisa-se determinar os pesos de enlaces OSPF tais que o algoritmo de roteamento OSPF resulte no roteamento desejado dos fluxos.

periódica de anúncios de enlace adiciona robustez ao algoritmo de estado de enlace”. Anúncios OSPF são contidos em mensagens OSPF carregadas diretamente por IP, com um código protocolo de camada superior 89 para OSPF. Assim, o próprio protocolo OSPF tem de executar funcionalidades como transferência confiável de mensagem e transmissão de estado de enlace por difusão. O protocolo OSPF também verifica se os enlaces estão operacionais (via uma mensagem HELLO enviada a um vizinho ligado ao enlace) e permite que um roteador OSPF obtenha o banco de dados de um roteador vizinho referente ao estado do enlace no âmbito da rede.

Alguns dos avanços incorporados ao OSPF são:

- *Segurança.* Trocas entre roteadores OSPF (p. ex., atualizações do estado de enlace) podem ser autenticadas. A autenticação garante que apenas roteadores de confiança consigam participar do protocolo OSPF dentro de um AS, evitando, assim, que intrusos mal-intencionados (ou estudantes de rede testando, por brincadeira, seu conhecimento recém-adquirido) injetem informações incorretas em tabelas de roteamento. Como padrão, pacotes OSPF entre roteadores não são autenticados e poderiam ser forjados. Dois tipos de autenticação podem ser configurados – simples e MD5 (veja o Capítulo 8 para obter uma discussão sobre MD5 e autenticação em geral). Com autenticação simples, a mesma senha é configurada em cada roteador. Quando um roteador envia um pacote OSPF, inclui a senha em texto aberto (não criptografado). Logicamente, a autenticação simples não é segura. A autenticação MD5 é baseada em chaves secretas compartilhadas que são configuradas em todos os roteadores. Para cada pacote OSPF enviado, o roteador calcula o hash MD5 do conteúdo do pacote adicionado com a chave secreta. (Consulte a discussão sobre códigos de autenticação de mensagem no Capítulo 8.) Então, o roteador inclui no pacote OSPF o valor de hash resultante. O roteador receptor, usando a chave secreta pré-configurada, calculará um hash MD5 do pacote e o comparará com o valor de hash que este transporta, verificando assim sua autenticidade.

Números de sequência também são utilizados com autenticação MD5 para proteção contra ataques por reenvio.

- *Caminhos múltiplos com o mesmo custo.* Quando vários caminhos até o destino têm o mesmo custo, o OSPF permite que sejam usados diversos caminhos (i.e., não é preciso escolher um único para carregar todo o tráfego quando existem vários de igual custo).
- *Suporte integrado para roteamento individual e em grupo (unicast e multicast).* O *multicast* OSPF (MOSPF) (RFC 1584) fornece extensões simples ao OSPF para prover roteamento em grupo. O MOSPF usa o banco de dados de enlaces existente no OSPF e acrescenta um novo tipo de anúncio de estado de enlace ao mecanismo OSPF de transmissão de estado de enlace por difusão.
- *Suporte para hierarquia dentro de um AS individual.* Um sistema autônomo OSPF pode ser configurado hierarquicamente em áreas. Cada área roda seu próprio algoritmo de roteamento de estado de enlace OSPF, e cada roteador em uma área transmite seu estado de enlace, por difusão, a todos os outros roteadores daquela área. Dentro de cada área, um ou mais roteadores de borda de área são responsáveis pelo roteamento de pacotes fora da área. Por fim, exatamente uma área OSPF no AS é configurada para ser a área de backbone. O papel primordial da área de backbone é rotear tráfego entre as outras áreas do AS. O backbone sempre contém todos os roteadores de borda de área que estão dentro do AS e pode conter também roteadores que não são de borda. O roteamento interárea dentro do AS requer que o pacote seja roteado primeiro até um roteador de borda de área (roteamento intra-área), em seguida roteado por meio do backbone até o roteador de borda de área que está na área de destino e, então, roteado até seu destino final.

O OSPF é um protocolo bastante complexo e, aqui, nosso tratamento teve de ser breve; Huitema (1998), Moy (1998) e RFC 2328 oferecem detalhes adicionais.

5.4 ROTEAMENTO ENTRE OS ISPS: BGP

Acabamos de ver que o OSPF é um exemplo de protocolo de roteamento intra-AS. Quando um pacote é roteado entre uma origem e um destino no mesmo AS, a rota seguida pelo pacote é totalmente determinada pelo protocolo de roteamento intra-AS. Contudo, para rotear um pacote entre múltiplos ASs (p. ex., de um smartphone em Timbuktu até um servidor em um datacenter no Vale do Silício), precisamos de um **protocolo de roteamento intersistema autônomo**. Como um protocolo de roteamento inter-AS envolve a coordenação entre múltiplos ASs, os ASs que se comunicam entre si devem executar o mesmo protocolo inter-AS. Na verdade, na Internet, todos os ASs executam o mesmo protocolo de roteamento inter-AS, chamado Protocolo de Roteador de Borda (BGP, do inglês *Border Gateway Protocol*) (RFC 4271; Stewart, 1999).

O BGP é talvez o mais importante de todos os protocolos da Internet (o único concorrente à altura seria o protocolo IP, que estudamos na Seção 4.3), pois é o protocolo que une os milhares de ISPs do mundo. Como veremos em seguida, o BGP é um protocolo descentralizado e assíncrono, no estilo do roteamento de vetor de distâncias descrito na Seção 5.2.2. O BGP é um protocolo complexo e difícil, mas para entender a Internet em suas entradas, precisamos nos familiarizar com os seus fundamentos e a sua operação. O tempo que dedicarmos a aprender o BGP será um ótimo investimento.

5.4.1 O papel do BGP

Para entender as responsabilidades do BGP, considere um AS e um roteador arbitrário em tal AS. Lembre-se que todos os roteadores possuem uma tabela de repasse, que tem um papel central no processo de repassar os pacotes que chegam para os enlaces de saída do

roteador. Como vimos, para destinos no mesmo AS, as linhas na tabela de repasse do roteador são determinadas pelo protocolo de roteamento intra-AS do AS. Mas e quanto a destinos externos ao AS? É exatamente aí que entra o BGP.

No BGP, os pacotes não são roteados para um endereço de destino específico, mas para prefixos “ciderizados”, com cada prefixo representando uma sub-rede ou conjunto de sub-redes. No mundo do BGP, um destino poderia ter a forma 138.16.68/22, que, para este exemplo, inclui 1.024 endereços IP. Assim, a tabela de repasse do roteador terá linhas com a forma (x, I) , em que x é um prefixo (como 138.16.68/22) e I é um número de interface para uma das interfaces do roteador.

Enquanto protocolo de roteamento inter-AS, o BGP provê a cada roteador um meio de:

1. *Obter informações de alcançabilidade do prefixo de ASs vizinhos.* O BGP permite, sobretudo, que cada sub-rede anuncie sua existência ao restante da Internet. Uma sub-rede grita “Eu existo e estou aqui”, e o BGP garante que todos os ASs da Internet saibam de sua existência e como chegar até ela. Não fosse o BGP, cada sub-rede ficaria isolada – sozinha e desconhecida pelo restante da Internet.
2. *Determinar as “melhores” rotas até os prefixos.* O roteador pode ser informado de duas ou mais rotas diferentes até um prefixo específico. Para determinar a melhor rota, o roteador executa localmente um procedimento de seleção de rota BGP (usando as informações de alcançabilidade do prefixo obtidas de ASs vizinhos). A melhor rota será determinada com base em uma política e nas informações de alcançabilidade.

Vamos agora nos aprofundar em como o BGP executa essas duas tarefas.

5.4.2 Anúncio de informações de rota BGP

Considere a rede mostrada na Figura 5.8. Como vemos, essa rede simples possui três sistemas autônomos: AS1, AS2 e AS3. Como mostrado, AS3 inclui uma sub-rede com prefixo x . Para cada AS, cada roteador é um **roteador de borda** ou então um **roteador interno**. Um roteador de borda (*gateway router*) é um roteador na borda de um AS que se conecta diretamente a um ou mais roteadores em outros ASs. Um **roteador interno** se conecta apenas a hospedeiros e roteadores dentro do seu próprio AS. No AS1, por exemplo, o roteador 1c é um roteador de borda, enquanto os roteadores 1a, 1b e 1d são roteadores internos.

Consideremos agora a tarefa de anunciar as informações de alcançabilidade do prefixo x para todos os roteadores mostrados na Figura 5.8. Em alto nível, é simples. Primeiro, AS3 envia uma mensagem de BGP para AS2, informando que x existe e está no AS3; denotemos essa mensagem por “AS3 x”. A seguir, AS2 envia uma mensagem BGP para AS1, informando que x existe e que pode ser alcançado passando primeiro por AS2 e então indo para AS3; denotemos essa mensagem por “AS2 AS3 x”. Dessa maneira, cada um dos sistemas

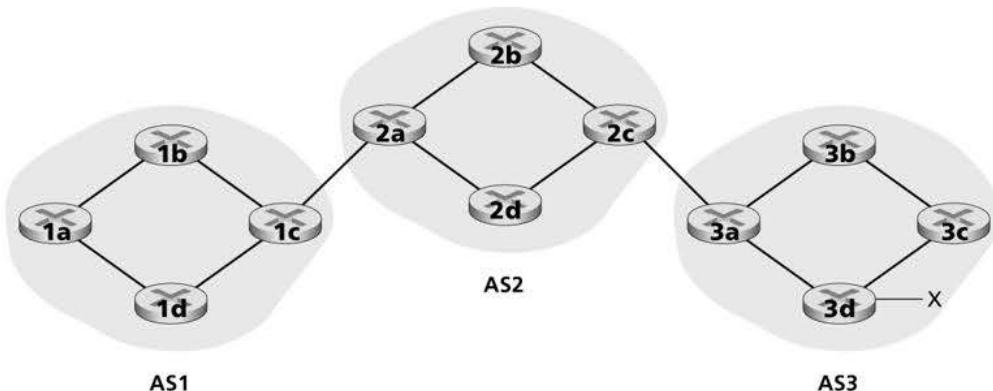


Figura 5.8 Rede com três sistemas autônomos. AS3 inclui uma sub-rede com prefixo x .

autônomos, além de descobrir a existência de x , será também informado sobre um caminho composto por sistemas autônomos que leva a x .

A discussão no parágrafo acima sobre anunciar as informações de alcançabilidade BGP deve bastar para comunicar a ideia geral, mas não é exata no sentido de que sistemas autônomos não mandam de fato mensagens uns para os outros; quem faz isso são os roteadores. Para entender o processo, vamos reanalisar o exemplo da Figura 5.8. No BGP, pares de roteadores trocam informações de roteamento através de conexões TCP semipermanentes usando a porta 179. Cada uma dessas conexões TCP, junto com todas as mensagens BGP enviadas através dela, é chamada de **conexão BGP**. Além disso, uma conexão BGP que abrange dois ASs é chamada de conexão **BGP externa (eBGP)**, enquanto uma sessão BGP entre roteadores no mesmo AS é chamada de conexão **BGP interna (iBGP)**. A Figura 5.9 mostra exemplos de conexões BGP para a rede da Figura 5.8. Em geral, há uma conexão eBGP para cada enlace que liga diretamente os roteadores de borda em diferentes ASs; assim, na Figura 5.9, há uma conexão eBGP entre os roteadores de borda 1c e 2a e uma conexão eBGP entre os roteadores de borda 2c e 3a.

Também há conexões iBGP entre os roteadores em cada um dos ASs. Em especial, a Figura 5.9 mostra uma configuração comum de uma conexão BGP para cada par de roteadores interno a um AS, criando uma malha de conexões TCP dentro de cada AS. Na Figura 5.9, as conexões eBGP são mostradas com traços longos, enquanto as conexões iBGP são mostradas com traços curtos. Observe que as conexões iBGP nem sempre correspondem a enlaces físicos.

Para propagar as informações de alcançabilidade, são usadas sessões iBGP e eBGP. Considere mias uma vez o anúncio das informações de alcançabilidade referentes ao prefixo x para todos os roteadores em AS1 e AS2. Nesse processo, o roteador de borda 3a envia uma mensagem eBGP “AS3 x ” para o roteador de borda 2c. Em seguida, o roteador de borda 2c envia a mensagem iBGP “AS3 x ” para todos os outros roteadores em AS2, incluindo o roteador de borda 2a. O roteador de borda 2a envia então a mensagem eBGP “AS2 AS3 x ” para o roteador de borda 1c. Por fim, o roteador de borda 1c usa o iBGP para enviar a mensagem “AS2 AS3 x ” para todos os roteadores em AS1. Após esse processo estar completo, cada roteador em AS1 e AS2 está ciente da existência de x e de um caminho AS que leva a x .

Obviamente, em uma rede real, pode haver muitos caminhos diferentes até um determinado destino a partir de um roteador qualquer, cada um através de uma sequência diferente de ASs. Considere, por exemplo, a rede da Figura 5.10, que é a rede original da Figura 5.8 com um enlace físico adicional do roteador 1d ao 3d. Nesse caso, há dois caminhos de AS1 a x : o caminho “AS2 AS3 x ” através do roteador 1c e o novo caminho “AS3 x ” através do roteador 1d.

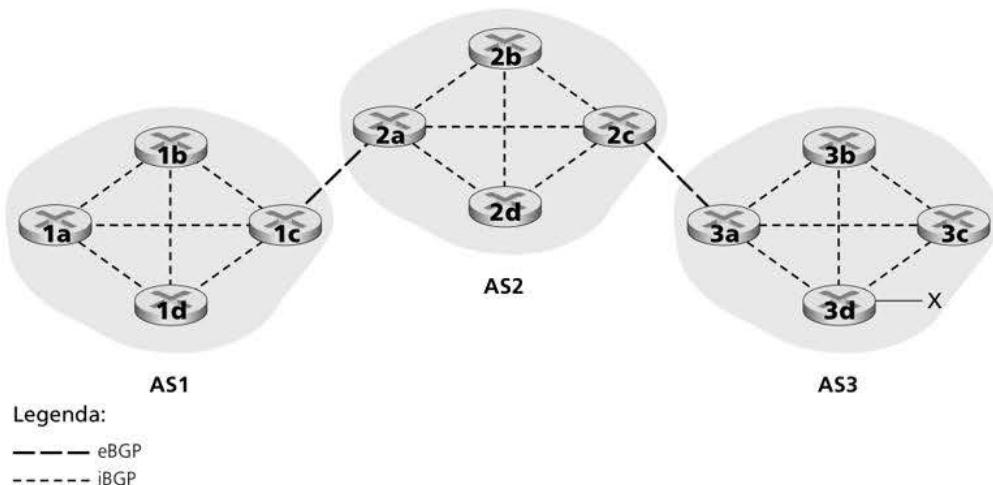


Figura 5.9 Conexões eBGP e iBGP.

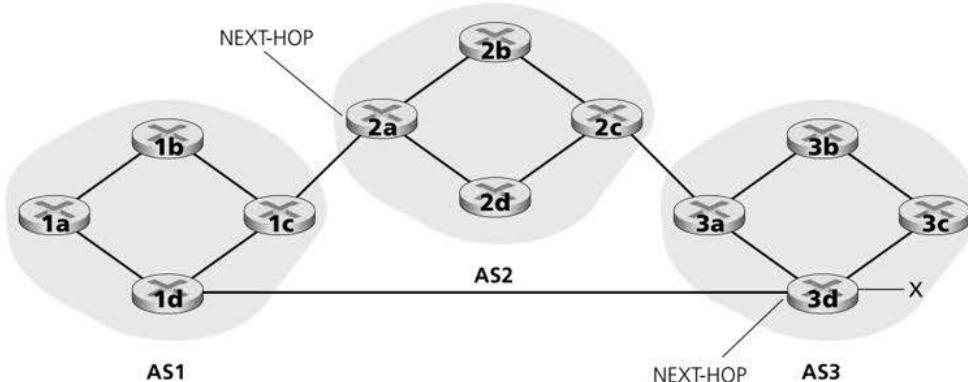


Figura 5.10 Rede ampliada com enlace de parceria (*peering*) entre AS1 e AS3.

5.4.3 Determinando as melhores rotas

Como acabamos de aprender, pode haver muitos caminhos entre um determinado roteador e uma sub-rede de destino. Na verdade, na Internet, os roteadores muitas vezes recebem informações de alcançabilidade sobre dezenas de caminhos possíveis diferentes. Como o roteador escolhe entre esses caminhos (e configura a sua tabela de repasse para tanto)?

Antes de responder a essa questão crítica, precisamos apresentar um pouco mais da terminologia do BGP. Quando anuncia um prefixo através de uma conexão BGP, o roteador inclui com o prefixo diversos **atributos BGP**. No jargão do BGP, um prefixo somado aos seus atributos é chamado de **rota**. Dois dos atributos mais importantes são AS-PATH e NEXT-HOP. O atributo AS-PATH contém a lista dos ASs pelos quais o anúncio passou, como vimos em nossos exemplos acima. Para gerar o valor AS-PATH, quando um prefixo passa para um AS, este adiciona o seu ASN à lista existente no AS-PATH. Por exemplo, na Figura 5.10, há duas rotas entre AS1 e a sub-rede x: uma que usa o AS-PATH “AS2 AS3” e outra que usa o AS-PATH “AS3”. Roteadores BGP usam o atributo AS-PATH para detectar e evitar laços de anúncios; mais especificamente, se um roteador perceber que seu AS está contido na lista de caminhos, rejeitará o anúncio.

Fornecendo o enlace crítico entre os protocolos de roteamento inter-AS e intra-AS, o atributo NEXT-HOP possui uma sutil mas importante utilidade. O NEXT-HOP é o *endereço IP da interface do roteador que inicia o AS-PATH*. Para compreender mais esse atributo, vamos, de novo, nos referir à Figura 5.10. Como indicado nessa figura, o atributo NEXT-HOP para a rota “AS2 AS3 x” de AS1 a x que passa por AS2 é o endereço IP da interface esquerda no roteador 2a. O atributo NEXT-HOP da rota “AS3 x” de AS1 a x que não passa por AS2 é o endereço IP da interface da extremidade esquerda do roteador 3d. Em suma, nesse minie exemplo, cada roteador em AS1 é informado de duas rotas BGP até o prefixo X:

Endereço IP da interface da extremidade esquerda para o roteador 2a; AS2 AS3; x;
Endereço IP da interface da extremidade esquerda do roteador 3d; AS3; x.

Aqui, cada rota BGP é escrita na forma de uma lista com três componentes: NEXT-HOP; AS-PATH; prefixo de destino. Na prática, uma rota BGP inclui atributos adicionais, que ignoraremos por enquanto. Observe que o atributo NEXT-HOP é um endereço IP de um roteador que *não* pertence a AS1; contudo, a sub-rede que contém esse endereço IP está ligada diretamente a AS1.

Roteamento da batata quente

Agora, *finalmente* estamos preparados para falar sobre algoritmos de roteamento BGP de forma precisa. Começaremos com um dos algoritmos de roteamento mais simples, a saber, o **roteamento da batata quente**.

Considere o roteador 1b na rede da Figura 5.10. Como descrito, esse roteador descobrirá duas rotas BGP possíveis até o prefixo x. No roteamento da batata quente, a rota escolhida (entre todas as rotas possíveis) é aquela com o menor custo até o próximo roteador NEXT-HOP a partir daquela rota. Nesse exemplo, o roteador 1b consulta as suas informações de roteamento intra-AS para descobrir o caminho de menor custo intra-AS até o roteador NEXT-HOP 2a e o caminho de menor custo intra-AS até o roteador NEXT-HOP 3d, e então seleciona a rota com o menor desses caminhos de menor custo. Por exemplo, suponha que o custo é definido como o número de enlaces atravessados. Nesse caso, o menor custo do roteador 1b ao roteador 2a é 2, o menor custo do roteador 1b ao roteador 2d é 3; logo, o roteador 2a seria selecionado. Assim, o roteador 1b consultaria a sua tabela de repasse (configurada pelo seu algoritmo intra-AS) e descobriria a interface I que está no caminho de menor custo até o roteador 2a. A seguir, ele adiciona (x, I) à sua tabela de repasse.

A Figura 5.11 resume os passos para se adicionar um prefixo fora do AS à tabela de repasse do roteador para roteamento da batata quente. É importante observar que, ao adicionar um prefixo fora do AS a uma tabela de repasse, utiliza-se ambos os protocolos de roteamento inter-AS (BGP) e de roteamento intra-AS (p. ex., OSPF).

A ideia por trás do roteamento da batata quente é que o roteador 1b tire os pacotes do seu AS o mais rápido possível (mais especificamente, com o menor custo possível) sem se preocupar com o custo das porções remanescentes do caminho fora do seu AS até o destino. No nome “roteamento da batata quente”, o pacote é análogo à batata quente que queima suas mãos. Como está queimando, você quer passá-la para outra pessoa (outro AS) o quanto antes. O roteamento da batata quente é, assim, um algoritmo egoísta, pois tenta reduzir o custo no seu próprio AS enquanto ignora os outros componentes dos custos de fim a fim fora do seu AS. Observe que, com o roteamento da batata quente, dois roteadores no mesmo AS podem escolher dois caminhos AS diferentes até o mesmo prefixo. Por exemplo, acabamos de ver que o roteador 1b enviaria pacotes através de AS2 para alcançar x. Contudo, o roteador 1d contornaria AS2 e enviaria pacotes diretamente a AS3 para alcançar x.

Algoritmo de seleção de rota

Na prática, o BGP usa um algoritmo mais complicado do que o roteamento de batata quente, mas que ainda o incorpora. Para um determinado prefixo de destino, os dados usados pelo algoritmo de seleção de rota do BGP são o conjunto de todas as rotas que o roteador descobriu e aceitou. Se houver apenas uma rota, obviamente, o BGP a seleciona. Se houver duas ou mais para o mesmo prefixo, então o BGP invoca sequencialmente as seguintes regras de eliminação, até sobrar apenas uma.

1. A rota recebe, como um de seus atributos, um valor de **preferência local** (além dos atributos AS-PATH e NEXT-HOP). A preferência local de uma rota pode ter sido estabelecida pelo roteador ou ter sido descoberta por um outro roteador no mesmo AS. O valor do atributo preferência local é uma decisão política que fica absolutamente a cargo do administrador de rede do AS. (Mais adiante, discutiremos detalhes sobre questões de política do BGP.) São selecionadas as rotas que têm os valores de preferência local mais altos.

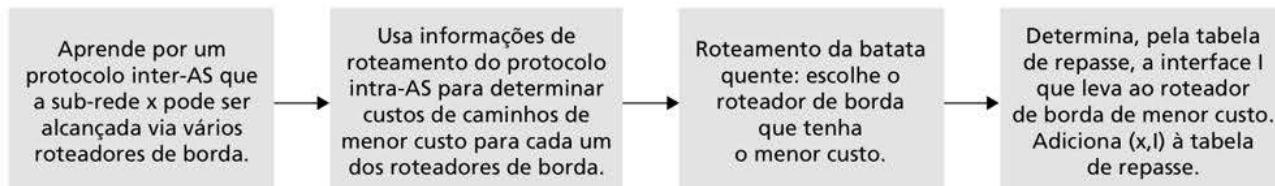


Figura 5.11 Etapas da adição de um destino fora do AS à tabela de repasse de um roteador.

2. No caso das outras rotas remanescentes (todas com o mesmo valor de preferência local), é selecionada a que tenha o AS-PATH mais curto. Se essa fosse a única regra de seleção, então o BGP estaria usando um algoritmo DV para determinação de caminho no qual a métrica da distância utiliza o número de saltos de AS em vez do número de saltos de roteadores.
3. Entre as rotas remanescentes (todas com o mesmo valor de preferência local e com o mesmo comprimento de AS-PATH), utiliza-se o roteamento da batata quente, ou seja, é selecionada a que tenha o roteador NEXT-HOP mais próximo.
4. Se ainda restar mais de uma rota, o roteador usa identificadores BGP para selecionar a rota; veja Stewart (1999).

Por exemplo, consideremos novamente o roteador 1b na Figura 5.10. Lembre-se de que há exatamente duas rotas BGP até o prefixo x, uma que passa por AS2 e outra que evita AS2. Lembre-se também que se usássemos apenas o roteamento da batata quente, o BGP rotearia pacotes através de AS2 até o prefixo x. Mas no algoritmo de seleção de rota acima, a regra 2 é aplicada antes da regra 3, fazendo o BGP selecionar a rota que contorna AS2, pois esta tem um AS-PATH menor. Assim, vemos que com o algoritmo de seleção de rota acima, o BGP deixa de ser um algoritmo egoísta, pois primeiro busca as rotas com caminhos AS curtos (e, logo, provavelmente reduz o atraso fim a fim).

Como já observamos, o BGP é um padrão na prática para roteamento inter-AS na Internet. Para ver o conteúdo de várias tabelas de roteamento BGP (grandes!) extraídas de roteadores pertencentes a ISPs de nível 1, consulte <<http://www.routeviews.org>>. Tabelas de roteamento BGP em geral contêm mais de meio milhão de rotas (i.e., prefixo e atributos correspondentes). Estatísticas sobre tamanho e características de tabelas de roteamento BGP são apresentadas em Huston (2019b).

5.4.4 IP-Anycast

Além de ser o protocolo de roteamento inter-AS da Internet, o BGP é muito usado para implementar o serviço de IP-anycast (RFC 1546, RFC 7094), usado comumente no DNS. Para entender o motivo por trás do IP-anycast, considere que, em muitas aplicações, estamos interessados em (1) replicar o mesmo conteúdo em diversos servidores, em muitos locais geograficamente dispersos, e (2) fazer com que cada usuário acesse o conteúdo do servidor mais próximo. Por exemplo, uma rede de distribuição de conteúdo (CDN, do inglês *content distribution network*) pode replicar vídeos e outros objetos em servidores espalhados por diversos países. Da mesma forma, o sistema DNS pode replicar registros DNS em servidores DNS em todo o mundo. Quando um usuário quer acessar esse conteúdo replicado, deseja-se que o usuário seja guiado para o servidor com o conteúdo replicado “mais próximo”. O algoritmo de seleção de rota do BGP oferece um mecanismo fácil e natural para se fazer isso.

Para que nossa discussão seja concreta, vamos descrever como uma CDN utilizaria IP-anycast. Como mostrado na Figura 5.12, durante o estágio de configuração do IP-anycast, a companhia de CDN associa o *mesmo* endereço IP a cada um dos seus servidores. Quando um roteador BGP recebe múltiplos anúncios de rotas para esse endereço IP, trata esses anúncios como diferentes caminhos disponíveis para a mesma localização física (quando, na verdade, os anúncios são para diferentes caminhos para diferentes localizações físicas). Ao configurar sua tabela de roteamento, cada roteador usará localmente o algoritmo de seleção de rota do BGP para selecionar a “melhor” (p. ex., a mais próxima, como determinado pelos contadores de salto de AS) rota para o endereço IP. Por exemplo, se uma rota BGP (correspondente a uma localização) está a apenas um salto de AS do roteador e todas as demais rotas BGP (correspondentes às outras localizações) estão a dois ou mais saltos de AS, então o roteador BGP escolherá rotear pacotes para a localidade que está a um salto de distância. Depois dessa fase inicial de anúncio do endereço, a CDN pode fazer o seu trabalho principal, que é distribuir conteúdo. Quando um cliente solicita um vídeo, a CDN retorna ao cliente o endereço IP comum usado pelos servidores geograficamente dispersos, não

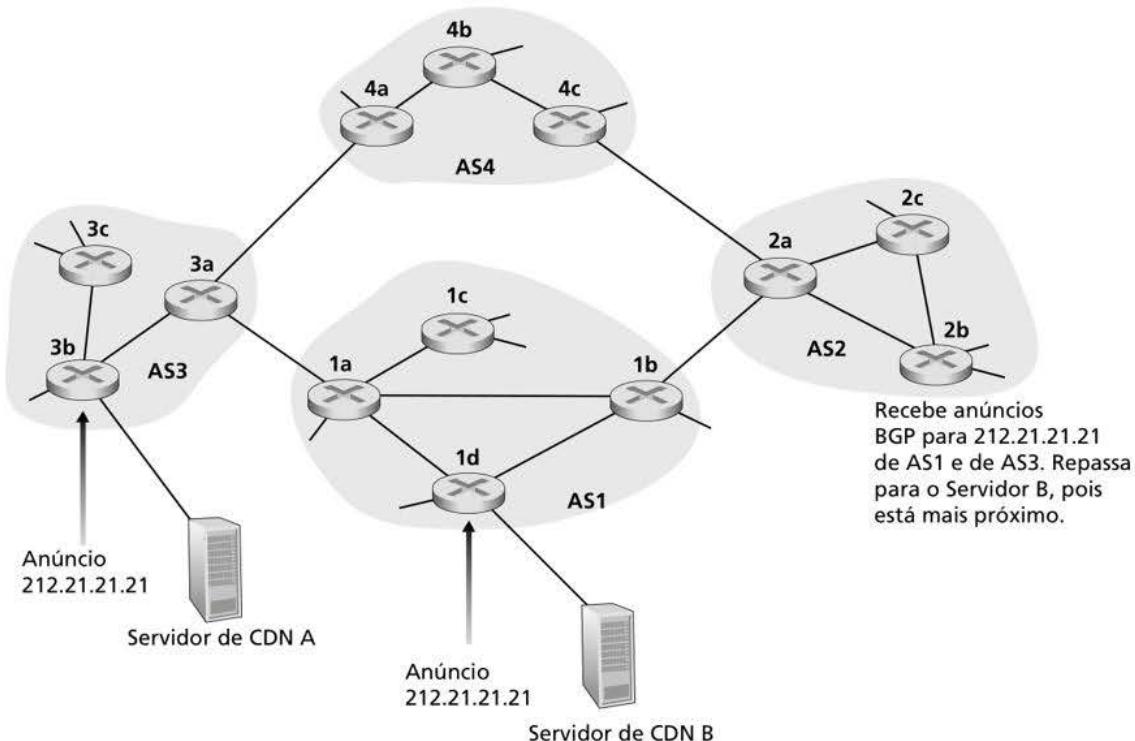


Figura 5.12 Uso de IP-anycast para levar usuários ao servidor CDN mais próximo.

importa onde o cliente esteja localizado. Quando o cliente envia uma requisição para esse endereço IP, os roteadores da Internet encaminham o pacote de solicitação para o servidor “mais próximo”, conforme definido pelo algoritmo de seleção de rota do BGP.

Embora o exemplo de CDN acima ilustre bem como o IP-anycast pode ser utilizado, na prática, as CDNs geralmente escolhem não usá-lo, pois as mudanças no roteamento BGP podem fazer diferentes pacotes da mesma conexão TCP chegarem em instâncias diferentes no servidor Web. Mas o IP-anycast é bastante utilizado por sistemas DNS para direcionar consultas de DNS para o servidor DNS raiz mais próximo. Como vimos na Seção 2.4, atualmente há 13 endereços IP para servidores DNS raiz, mas há múltiplos servidores DNS raiz correspondentes a cada um desses endereços, com alguns desses endereços tendo mais de 100 servidores DNS raiz espalhados pelos quatro cantos do mundo. Quando uma consulta de DNS é enviada a um desses 13 endereços IP, o IP-anycast é utilizado para rotear a consulta para os servidores DNS raiz mais próximos responsáveis pelo endereço. Li (2018) apresenta medições recentes que ilustram o uso, o desempenho e os desafios do IP-anycast.

5.4.5 Política de roteamento

Quando um roteador seleciona uma rota até um destino, a política de roteamento AS pode superar todas as outras considerações, como o caminho AS mais curto ou o roteamento da batata quente. Na verdade, no algoritmo de seleção de rota, primeiro as rotas são selecionadas de acordo com o atributo de preferência local, cujo valor é fixado pela política do AS local.

Vamos ilustrar alguns dos conceitos básicos da política de roteamento BGP com um exemplo simples. A Figura 5.13 mostra seis sistemas autônomos interconectados: A, B, C, W, X e Y. É importante notar que A, B, C, W, X e Y são ASs, e não roteadores. Vamos admitir que os sistemas autônomos W, X e Y são ISPs de acesso, e que A, B e C são redes

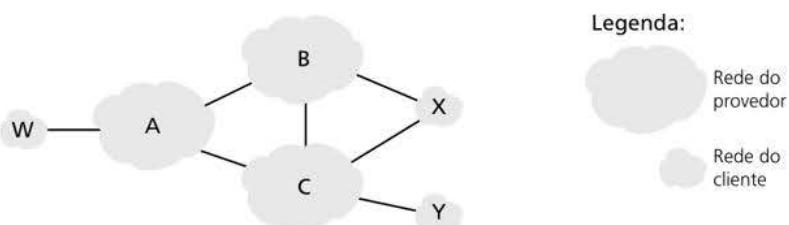


Figura 5.13 Um cenário BGP simples.

provedoras de backbone. Vamos supor também que A, B e C enviam tráfego diretamente uns para os outros e fornecem informação completa sobre o BGP a suas redes clientes. Todo o tráfego que entrar em uma rede de ISP de acesso deve ser destinado a essa rede, e todo o tráfego que sair da rede de ISP de acesso deve ter sido originado naquela rede. W e Y são claramente ISPs de acesso. X é um **ISP de acesso com múltiplas interconexões (multi-homed)**, visto que está ligado ao resto da rede por meio de dois provedores diferentes (um cenário que está se tornando cada vez mais comum na prática). Todavia, tal como W e Y, o próprio X deve ser a origem/destino de todo o tráfego que entra/sai de X. Porém, como esse comportamento da rede será executado e imposto? Como X será impedido de repassar tráfego entre B e C? Isso pode ser conseguido facilmente controlando o modo como as rotas BGP são anunciatas. Em particular, X funcionará como uma rede de ISP de acesso se anunciar (a seus vizinhos B e C) que não há nenhum caminho para quaisquer outros destinos a não ser ele mesmo. Isto é, mesmo que X conheça um caminho, digamos, XCY, que chegue até a rede Y, ele não anunciará esse caminho a B. Como B não fica sabendo que X tem um caminho para Y, B nunca repassaria tráfego destinado a Y (ou a C) por meio de X. Esse exemplo simples ilustra como uma política seletiva de anúncio de rota pode ser usada para implementar relacionamentos de roteamento cliente/provedor.

Em seguida, vamos focalizar uma rede provedora, digamos, o AS B. Suponha que B ficasse sabendo (por A) que A tem um caminho AW para W. Assim, B pode instalar a rota AW em sua base de informações de roteamento. É claro que B também quer anunciar o caminho BAW a seu cliente, X, de modo que X saiba que pode rotear para W via B. Porém, B deveria anunciar o caminho BAW a C? Se o fizer, então C poderá rotear tráfego para W via BAW. Se A, B e C forem todos provedores de backbone, então B poderá sentir-se no direito de achar que não deveria ter de suportar a carga (e o custo!) de transportar o tráfego em trânsito entre A e C. B poderá sentir-se no direito de achar que é de A e C o trabalho (e o custo!) de garantir que C possa rotear de/para clientes de A por meio de uma conexão direta entre A e C. Hoje, não existe nenhum padrão oficial que determine como ISPs de backbone devem rotear entre si. Todavia, os ISPs comerciais adotam uma regra prática que diz que qualquer tráfego que esteja fluindo por uma rede de backbone de um ISP deve ter ou uma origem ou um destino (ou ambos) em uma rede que seja cliente daquele ISP; caso contrário, o tráfego estaria pegando uma carona gratuita na rede do ISP. Acordos individuais de parceria (*peering*) (para reger questões como as levantadas) costumam ser negociados entre pares de ISPs e, em geral, são confidenciais; Huston (1999a; 2012) provê uma discussão interessante sobre acordos de parceria. Se quiser uma descrição detalhada sobre como a política de roteamento reflete os relacionamentos comerciais entre ISPs, veja Gao (2001) e Dimitropoulos (2007). Para ver uma abordagem recente sobre políticas de roteamento BGP, de um ponto de vista do ISP, consulte Caesar (2005b).

Com isso, concluímos nossa breve introdução ao BGP. Entender esse protocolo é importante, porque ele desempenha um papel central na Internet. Aconselhamos você a consultar as referências Stewart (1999); Huston (2019a); Labovitz (1997); Halabi (2000); Huitema (1998); Gao (2001); Feamster (2004), Caesar (2005b); Li (2007) para aprender mais sobre BGP.

PRINCÍPIOS NA PRÁTICA

POR QUE HÁ DIFERENTES PROTOCOLOS DE ROTEAMENTO INTER-AS E INTRA-AS?

Agora que já examinamos os detalhes de protocolos de roteamento inter-AS e intra-AS específicos utilizados pela Internet, vamos concluir considerando a questão talvez mais fundamental que, antes de tudo, poderíamos levantar sobre esses protocolos (esperamos que você tenha estado preocupado com isso o tempo todo e que não tenha deixado de enxergar o quadro geral em razão dos detalhes!). Por que são usados diferentes protocolos de roteamento inter-AS e intra-AS?

A resposta a essa pergunta expõe o âmago da diferença entre os objetivos do roteamento dentro de um AS e entre ASs:

- *Política.* Entre ASs, as questões políticas dominam. Pode até ser importante que o tráfego que se origina em determinado AS não possa passar por outro AS específico. De maneira semelhante, determinado AS pode muito bem querer controlar o tráfego em trânsito que ele carrega entre outros ASs. Vimos que o BGP carrega atributos de caminho e oferece distribuição controlada de informação de roteamento, de modo que as decisões de roteamento baseadas em políticas possam ser tomadas. Dentro de um AS, tudo está nominalmente no mesmo controle administrativo. Assim, as questões de políticas de roteamento

desempenham um papel bem menos importante na escolha de rotas no AS.

- *Escalabilidade.* A escalabilidade de um algoritmo de roteamento e de suas estruturas de dados para manipular o roteamento para/entre grandes números de redes é uma questão fundamental para o roteamento inter-AS. Dentro de um AS, a escalabilidade é uma preocupação menor. Isso porque, se um único ISP ficar muito grande, é sempre possível dividi-lo em dois ASs e realizar roteamento inter-AS entre esses dois novos ASs. (Lembre-se de que o OSPF permite que essa hierarquia seja construída dividindo um AS em áreas.)
- *Desempenho.* Dado que o roteamento inter-AS é bastante orientado pelas políticas, a qualidade (p. ex., o desempenho) das rotas usadas é muitas vezes uma preocupação secundária (i.e., uma rota mais longa ou de custo mais alto que satisfaça a certos critérios políticos pode muito bem prevalecer sobre uma que é mais curta, mas que não satisfaz a esses critérios). Na verdade, vimos que entre ASs não há nem mesmo a ideia de custo associado às rotas (exceto a contagem de saltos do AS). Dentro de um AS individual, contudo, essas preocupações com as políticas têm menos importância, permitindo que o roteamento se concentre mais no nível de desempenho atingido em uma rota.

5.4.6 Juntando o quebra-cabeça: obtendo presença na Internet

Embora não seja exatamente sobre BGP, esta subseção reúne muitos dos protocolos e conceitos que vimos até aqui, incluindo endereçamento IP, DNS e BGP.

Suponha que você tenha acabado de criar uma pequena rede com diversos servidores, incluindo um servidor Web público, que descreve os produtos e serviços da sua empresa, um de correio, do qual seus funcionários obtêm suas mensagens de correio eletrônico, e um de DNS. Claro, você gostaria que o mundo inteiro pudesse navegar em seu site para descobrir seus incríveis produtos e serviços. Além do mais, gostaria que seus funcionários pudessem enviar e receber correio eletrônico para clientes em potencial no mundo inteiro.

Para atender a esses objetivos, primeiro você precisa obter conectividade com a Internet, o que é feito contratando e conectando-se a um ISP local. Sua empresa terá um roteador de borda, que estará conectado a um roteador no seu ISP local. Essa conexão poderia ser uma DSL pela infraestrutura telefônica, uma linha privada com o roteador do ISP, ou uma das muitas outras soluções de acesso descritas no Capítulo 1. Seu ISP local também lhe oferecerá uma faixa de endereços IP, por exemplo, uma faixa de endereços /24 consistindo em 256 endereços. Ao obter sua conectividade física e sua faixa de endereços IP, você designará

um dos seus endereços IP (na sua faixa de endereços) para o seu servidor Web, um para o seu servidor de correio, um para o seu servidor DNS, um para o seu roteador de borda e outros endereços IP para outros servidores e dispositivos na rede da sua empresa.

Além de contratar um ISP, você também precisará contratar um registrador da Internet para obter um nome de domínio para a sua empresa, conforme descrevemos no Capítulo 2. Por exemplo, se a sua empresa tiver o nome Xanadu Inc., você decerto tentará obter o nome de domínio xanadu.com. Sua empresa também deverá obter presença no sistema DNS. Especificamente, como as pessoas de fora desejaram entrar em contato com seu servidor DNS para obter os endereços IP dos seus servidores, você também precisará oferecer ao seu registrador o endereço IP do seu servidor DNS. Seu registrador, então, colocará um registro para o seu servidor DNS (nome de domínio e endereço IP correspondente) nos servidores do domínio .com de nível superior, conforme descrevemos no Capítulo 2. Concluída essa etapa, qualquer usuário que saiba seu nome de domínio (p. ex., xanadu.com) poderá obter o endereço IP do seu servidor DNS por meio do sistema DNS.

Para que as pessoas consigam descobrir os endereços IP do seu servidor Web, você terá de incluir nele registros que mapeiem o nome de hospedeiro do servidor (p. ex., www.xanadu.com) ao endereço IP. Você desejará ter registros semelhantes para outros servidores publicamente disponíveis em sua empresa, incluindo o de correio. Dessa forma, se Alice quiser navegar pelo seu servidor Web, o sistema DNS entrará em contato com seu servidor DNS, achará o endereço IP do seu servidor Web e o dará a Alice. Assim, ela poderá estabelecer uma conexão TCP diretamente com o seu servidor Web.

Todavia, ainda resta uma etapa necessária e decisiva para permitir que outros do mundo inteiro accessem seu servidor Web. Considere o que acontece quando Alice, que conhece o endereço IP do seu servidor Web, envia um datagrama IP (p. ex., um segmento TCP SYN) a esse endereço IP. Esse datagrama será direcionado pela Internet, visitando uma série de roteadores em muitos ASs diferentes, para, enfim, alcançar seu servidor Web. Quando qualquer um dos roteadores recebe o datagrama, ele procura um registro em sua tabela de repasse para determinar em qual porta de saída deverá encaminhá-lo. Portanto, cada roteador precisa saber a respeito do prefixo /24 da sua empresa (ou de algum registro agregado). Como um roteador pode saber o prefixo da sua empresa? Como já vimos, isso é feito por meio do BGP! Especificamente, quando sua empresa contrata um ISP local e recebe um prefixo (i.e., uma faixa de endereços), seu ISP local usará o BGP para anunciar esse prefixo aos ISPs aos quais se conecta. Tais ISPs, por sua vez, usarão o BGP para propagar o anúncio. Por fim, todos os roteadores da Internet saberão a respeito do seu prefixo (ou sobre algum agregado que o inclua) e, desse modo, poderão repassar datagramas destinados a seus servidores Web e de correio de forma apropriada.

5.5 O PLANO DE CONTROLE DA SDN

Nesta seção, analisaremos o plano de controle da SDN, a lógica em âmbito de rede que controla o repasse de pacotes entre os dispositivos da rede que utilizam SDN, além da configuração e do gerenciamento desses dispositivos e dos seus serviços. Aqui, nosso estudo se baseia na nossa discussão anterior sobre repasse SDN generalizado na Seção 4.4, então pode ser útil revisar essa seção, assim como a Seção 5.1 deste capítulo, antes de continuar. Assim como na Seção 4.4, mais uma vez adotaremos a terminologia utilizada na literatura sobre SDN e chamaremos os dispositivos de repasse da rede de “comutadores de pacotes” (ou simplesmente de “comutadores”, pois os “pacotes” estão subentendidos), pois as decisões de repasse podem ser tomadas com base nos endereços de origem/destino da camada de rede, endereços de origem/destino da camada de enlace e diversos outros valores de campos de cabeçalho de pacote das camadas de transporte, rede e enlace.

Podemos identificar quatro características críticas de uma arquitetura de SDN (Kreutz, 2015):

- *Repasso baseado em fluxo.* O repasse de pacotes por comutadores controlados por SDN pode se basear em diversos valores de campos nos cabeçalhos das camadas de transporte, rede ou enlace. Vimos na Seção 4.4 que a abstração OpenFlow 1.0 permite o repasse baseado em 11 valores de campos de cabeçalho diferentes, em forte contraste com a abordagem tradicional ao repasse baseado em roteadores, estudada nas Seções 5.2 a 5.4, na qual o repasse de datagramas IP se baseia exclusivamente no endereço IP de destino do datagrama. Voltando à Figura 5.2, observe que as regras de repasse de pacotes são especificadas na tabela de fluxo do comutador; é função do plano de controle SDN calcular, gerenciar e instalar linhas da tabela de fluxo em todos os comutadores da rede.
- *Separação do plano de dados e plano de controle.* Essa separação é mostrada claramente nas Figuras 5.2 e 5.14. O plano de dados é composto por comutadores da rede, os dispositivos relativamente simples (mas rápidos) que executam as regras de “combinação mais ação” das suas tabelas de fluxo. O plano de controle é composto por servidores e software que determinam e gerenciam as tabelas de fluxo dos comutadores.
- *Funções de controle de rede: externas aos comutadores do plano de dados.* Dado que o “S” em SDN significa “software”, talvez não surpreenda que o plano de controle das SDNs seja implementado em software. Ao contrário dos roteadores tradicionais, no entanto, esse software é executado em servidores distintos e remotos em relação aos comutadores da rede. Como mostra a Figura 5.14, o plano de controle em si é composto por dois componentes: um controlador SDN (ou sistema operacional de rede [Gude, 2008]) e um conjunto de aplicações de controle de rede. O controlador mantém informações precisas sobre o estado da rede (p. ex., o estado de hospedeiros, comutadores e enlaces remotos); fornece essas informações às aplicações de controle de rede executadas no plano de controle; e cria os meios pelos quais essas aplicações podem monitorar, programar e controlar os dispositivos de rede subjacentes. Apesar do controlador da Figura

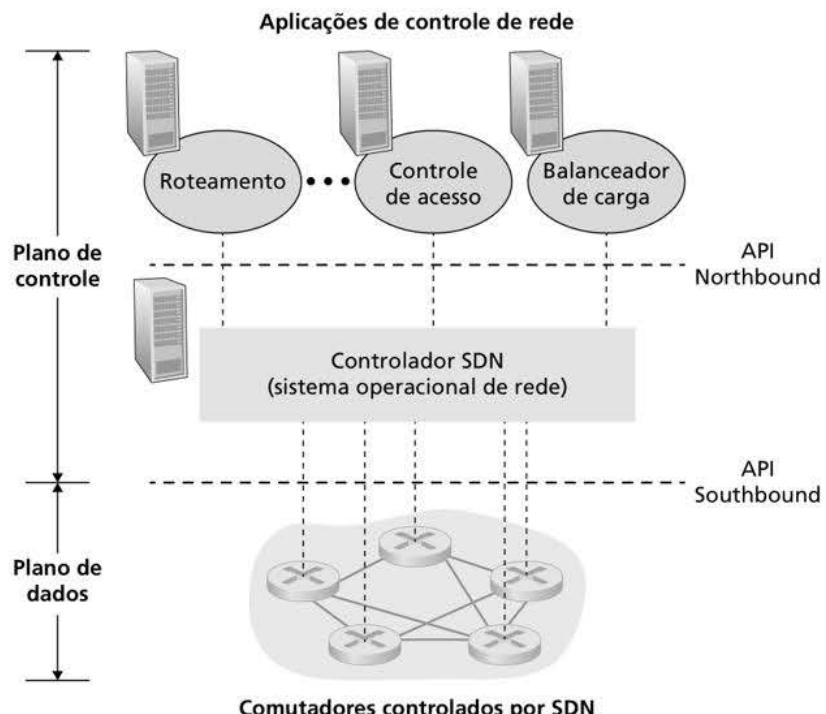


Figura 5.14 Componentes da arquitetura SDN: comutadores controlados por SDN, o controlador SDN e aplicações de controle de rede.

5.14 ser mostrado como um único servidor central, na prática, este é apenas logicamente centralizado; em geral, ele é implementado em diversos servidores, que oferecem desempenho coordenado e escalável e alta disponibilidade.

- *Uma rede programável.* A rede é programável por meio das aplicações de controle de rede que rodam no plano de controle. Elas representam o “cérebro” do plano de controle da SDN, usando as APIs oferecidas pelo controlador SDN para especificar e controlar o plano de dados nos dispositivos de rede. Por exemplo, uma aplicação de controle de rede de roteamento poderia determinar os caminhos fim a fim entre origens e destinos (p. ex., ao executar o algoritmo de Dijkstra usando informações de estado de nó e de enlace mantidas pelo controlador SDN). Outra aplicação de rede poderia realizar o controle de acesso, ou seja, determinar quais pacotes devem ser bloqueados no comutador, como em nosso terceiro exemplo na Seção 4.4.3. Outra aplicação poderia fazer os comutadores repassarem pacotes de maneiras que balanceassem a carga do servidor (o segundo exemplo que consideramos na Seção 4.4.3).

A partir dessa discussão, vemos que a SDN representa uma “desagregação” significativa da funcionalidade de rede – comutadores do plano de dados, controladores SDN e aplicações de controle de rede são entidades independentes que podem ser fornecidas por diferentes organizações e fornecedores. Essa situação contrasta com o modelo pré-SDN, no qual um comutador/roteador (junto com o software do plano de controle e implementações de protocolo embutidos) era monolítico, verticalmente integrado e vendido por um único fornecedor. Essa desagregação da funcionalidade de rede na SDN já foi comparada com a evolução anterior dos mainframes (em que hardware, software de sistema e aplicações eram vendidos por um único fornecedor) para os PCs (que têm hardware, sistema operacional e aplicações separados). A desagregação do hardware, software de sistema e aplicações levou à criação de um ecossistema rico e aberto, guiado pela inovação nas três áreas; espera-se que a SDN continue a promover e capacitar essa inovação abundante.

Dado o nosso entendimento sobre a arquitetura SDN da Figura 5.14, naturalmente, surgem muitas perguntas. Como e onde as tabelas de fluxo são calculadas? Como são atualizadas em resposta a eventos em dispositivos controlados por SDN (p. ex., um enlace direto sendo estabelecido ou interrompido)? Como as linhas da tabela de fluxo em múltiplos comutadores são coordenadas de modo a produzirem uma funcionalidade orquestrada e consistente em âmbito de rede (p. ex., caminhos fim a fim para repasse de pacotes de origens a destinos, ou firewalls distribuídos coordenados)? O papel de oferecer essas capacidades, e muitas outras, pertence ao plano de controle da SDN.

5.5.1 O plano de controle da SDN: controlador SDN e aplicações de controle de rede SDN

Começemos nossa discussão sobre o plano de controle da SDN em abstrato, considerando as capacidades genéricas que o plano de controle deve oferecer. Como veremos, essa abordagem abstrata, baseada em “primeiros princípios”, nos levará a uma arquitetura geral que reflete como os planos de controle SDN são implementados na prática.

Como observado acima, de forma geral, o plano de controle da SDN se divide em dois componentes, o controlador SDN e as aplicações de controle de rede SDN. Exploraremos primeiro o controlador. Muitos controladores SDN foram desenvolvidos desde o seu surgimento (Gude, 2008); para um levantamento extremamente completo, consulte Kreutz (2015). A Figura 5.15 apresenta uma visão mais detalhada de um controlador SDN genérico. A funcionalidade de um controlador pode ser organizada inicialmente em três camadas. Invertendo a lógica que organiza este livro, vamos considerar essas camadas de baixo para cima:

- *Uma camada de comunicação: a comunicação entre o controlador SDN e os dispositivos de rede controlados.* Claramente, se um controlador SDN vai controlar a operação de um comutador, hospedeiro ou outro dispositivo com SDN remoto, é preciso um

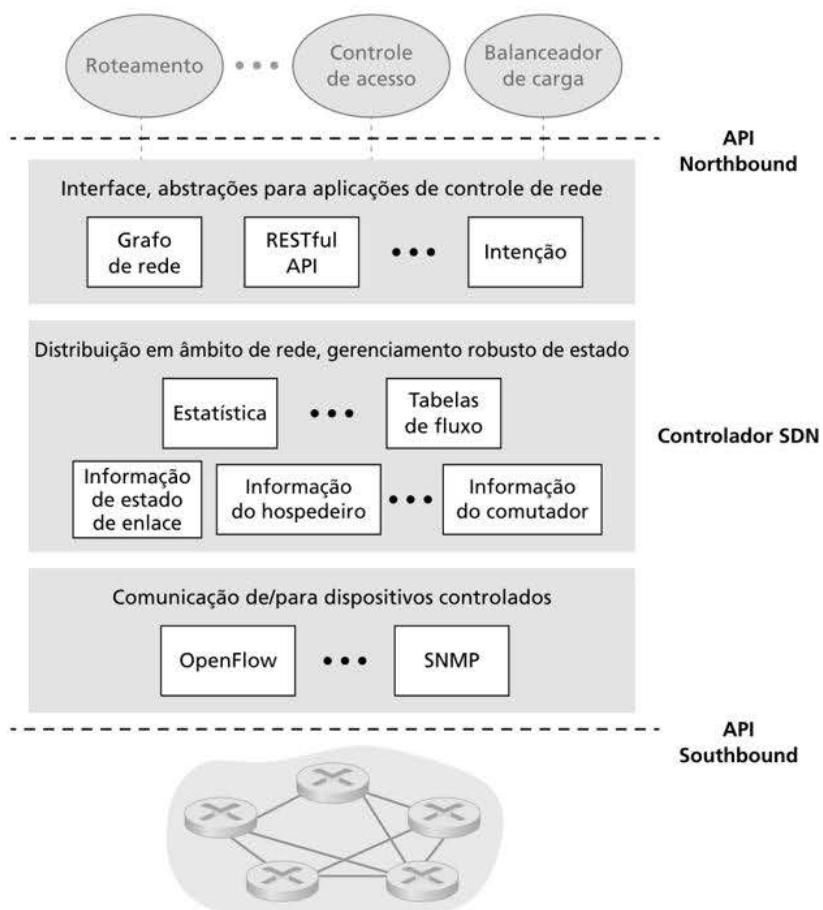


Figura 5.15 Componentes de um controlador SDN.

protocolo para transferir informações entre o controlador e o dispositivo. Além disso, o dispositivo deve poder comunicar eventos observados localmente ao controlador (p. ex., uma mensagem indicando que um enlace direto foi ativado ou interrompido, que um dispositivo se juntou à rede, ou um pulso indicando que um dispositivo está ativo e operacional). Esses eventos dão ao controlador SDN uma visão atualizada do estado da rede. Esse protocolo representa a camada inferior da arquitetura do controlador, como mostra a Figura 5.15. A comunicação entre o controlador e os dispositivos controlados cruza a chamada interface “southbound” (“em direção ao sul”) do controlador. Na Seção 5.5.2, estudaremos o OpenFlow, um protocolo específico que oferece essa funcionalidade de comunicação. O OpenFlow é implementado na maioria dos controladores SDN, se não em todos.

- *Uma camada de gerenciamento de estado em âmbito de rede.* As decisões de controle finais tomadas pelo plano de controle SDN (p. ex., configurar tabelas de fluxo em todos os comutadores para obter o repasse fim a fim desejado, implementar balanceamento de carga ou instalar uma determinada capacidade de firewall) exigirão que o controlador possua informações atualizadas sobre o estado dos hospedeiros, enlaces, comutadores e outros dispositivos controlados por SDN da rede. A tabela de fluxo de um comutador contém contadores cujos valores podem ser extremamente úteis para as aplicações de controle de rede e devem, então, estar disponíveis para as aplicações. Como o objetivo final do plano de controle é determinar as tabelas de fluxo para os diversos dispositivos controlados, o controlador também poderia manter uma cópia dessas tabelas.

Todas essas informações representam exemplos do “estado” geral da rede mantido pelo controlador SDN.

- *A interface com a camada de aplicação de controle de rede.* O controlador interage com as aplicações de controle de rede por meio da sua interface “northbound” (“em direção ao norte”). Essa API permite que as aplicações de controle de rede leiam/gravem o estado da rede e as tabelas de fluxo na camada de gerenciamento de estado. As aplicações podem se registrar para serem notificadas quando ocorre um evento de mudança de estado para que possam agir em resposta às notificações de eventos de rede enviadas de dispositivos controlados por SDN. Diferentes tipos de API podem ser fornecidos; veremos que dois controladores SDN populares se comunicam com suas aplicações usando a interface de requisição-resposta REST (Fielding, 2000).

Observamos diversas vezes que um controlador SDN podem ser considerado “logicamente centralizado”, ou seja, que o controlador pode ser visto externamente (p. ex., do ponto de vista de dispositivos controlados por SDN e aplicações de controle de rede externas) como um único serviço monolítico. Contudo, esses serviços e os bancos de dados usados para guardar informações de estado são implementados na prática como um conjunto *distribuído* de servidores para fins de tolerância a falhas, alta disponibilidade ou desempenho. Com as funções de controle implementadas por um *conjunto* de servidores, a semântica das operações internas do controlador (p. ex., manter o ordenamento temporal lógico dos eventos, consistência, consenso etc.) deve ser considerada (Panda, 2013). Essas preocupações são comuns a muitos sistemas distribuídos diferentes; para soluções elegantes a esses desafios, consulte Lamport (1989) e Lampson (1996). Controladores modernos, como OpenDaylight (OpenDaylight, 2020) e ONOS (ONOS, 2020) (veja nota em destaque) dão ênfase considerável à arquitetura de uma plataforma de controlador logicamente centralizada, mas fisicamente distribuída, que oferece serviços com escalabilidade e alta disponibilidade igualmente para os dispositivos controlados e para as aplicações de controle de rede.

A arquitetura representada na Figura 5.15 lembra de perto a arquitetura do controlador NOX proposta originalmente em 2008 (Gude, 2008), assim como os atuais controladores SDN OpenDaylight (OpenDaylight, 2020) e ONOS (ONOS, 2020) (veja nota em destaque). Trabalharemos um exemplo de operação de controlador na Seção 5.5.3. Primeiro, no entanto, vamos examinar o protocolo OpenFlow, o primeiro e, hoje, um de vários protocolos que pode ser utilizado para comunicação entre um controlador SDN e um dispositivo controlado, que fica na camada de comunicação do controlador.

5.5.2 Protocolo OpenFlow

O protocolo OpenFlow (OpenFlow, 2009; ONF, 2020) opera entre um controlador SDN e um comutador controlado por SDN ou outro dispositivo implementando a API OpenFlow que estudamos anteriormente na Seção 4.4. O protocolo OpenFlow opera sobre Protocolo de Controle de Transmissão (TCP, do inglês *Transmission Control Protocol*) e o seu número de porta é 6653.

As mensagens importantes que fluem do controlador para o comutador controlado incluem:

- *Configuração (Configuration).* Essa mensagem permite que o controlador consulte e defina os parâmetros de configuração de um comutador.
- *Modificação de estado (Modify-State).* Essa mensagem é usada pelo controlador para adicionar/deletar ou modificar linhas na tabela de fluxo do comutador e para configurar as propriedades de porta do comutador.
- *Leitura de estado (Read-State).* Essa mensagem é usada pelo controlador para coletar estatísticas e valores do contador das portas e tabela de fluxo do comutador.
- *Envio de pacote (Send-Packet).* Essa mensagem é usada pelo controlador para enviar um pacote específico de uma determinada porta no comutador controlado. A mensagem em si contém o pacote a ser enviado na sua carga útil.

As mensagens que fluem do comutador controlado por SDN para o controlador incluem:

- *Fluxo removido (Flow-Removed)*. Essa mensagem informa o controlador que uma linha da tabela de fluxo foi removida; por exemplo, por esgotamento de temporização ou devido a uma mensagem de *modificação de estado (modify-state)* recebida.
- *Estado da porta (Port-status)*. Essa mensagem é usada por um comutador para informar o controlador sobre uma mudança no estado da porta.
- *Entrada de pacotes (Packet-in)*. Voltando à Seção 4.4, lembre-se que um pacote que chega em uma porta de comutador e que não combina com nenhuma linha da tabela de fluxo é enviado ao controlador para processamento adicional. Pacotes que combinam com uma linha de fluxo também podem ser enviados ao controlador enquanto ação executada após uma combinação. A mensagem de *entrada de pacotes (packet-in)* é usada para enviar tais pacotes ao controlador.

Mensagens OpenFlow adicionais são definidas em (OpenFlow, 2009; ONF, 2020).

PRINCÍPIOS NA PRÁTICA

A REDE DEFINIDA POR SOFTWARE GLOBAL DA GOOGLE

No estudo de caso da Seção 2.6, vimos que a Google possui uma rede de longa distância (WAN, do inglês *wide-area network*) exclusiva que interconecta seus datacenters e *clusters* de servidores (em IXPs e ISPs). Essa rede, chamada de B4, possui um plano de controle SDN projetado pela Google e baseado em OpenFlow. A rede da Google é capaz de atingir quase 70% de utilização dos enlaces da WAN no longo prazo (duas ou três vezes mais que os índices de utilização de enlaces típicos) e dividir os fluxos de aplicações entre múltiplos caminhos com base na prioridade da aplicação e em demandas de fluxo existentes (Jain, 2013).

A rede B4 da Google é particularmente adequada para SDN: (i) a Google controla todos os dispositivos, dos servidores de borda nos IXPs e ISPs aos roteadores no núcleo da rede; (ii) as aplicações que mais consomem banda são cópias de dados em larga escala entre diversos locais, que podem dar espaço para aplicações interativas de alta prioridade durante momentos de congestão de recursos; (iii) com apenas algumas dúzias de datacenters conectados, o controle centralizado se torna viável.

A rede B4 da Google usa comutadores customizados, cada um dos quais implementa uma versão ligeiramente estendida do OpenFlow, com um Agente de OpenFlow (OFA, do inglês *OpenFlow Agent*) no mesmo espírito do agente de controle que encontramos

na Figura 5.2. Cada OFA, por sua vez, se conecta a um Controlador OpenFlow (OFC, do inglês *OpenFlow Controller*) no servidor de controle de rede (NCS, do inglês *network control server*), usando uma rede “fora de banda” independente, separada da rede que transmite o tráfego dos datacenters entre eles. Assim, o OFC presta serviços usados pelo NCS para se comunicar com os seus comutadores controlados, no mesmo espírito da camada inferior da arquitetura SDN mostrada na Figura 5.15. Na B4, o OFC também executa funções de gerenciamento de estado, mantendo os estados de nós e enlaces em uma Base de Informações de Rede (NIB, do inglês *Network Information Base*). A implementação do OFC por parte da Google se baseia no controlador SDN ONIX (Koponen, 2010). Dois protocolos de roteamento, BGP (para roteamento entre os datacenters) e IS-IS (um parente próximo do OSPF, para roteamento dentro de um mesmo datacenter), são implementados. O Paxos (Chandra, 2007) é usado para executar réplicas “quentes” (*hot replicas*) de componentes NCS como proteção contra falhas.

Uma aplicação de controle de rede de engenharia de tráfego, instalada logicamente acima do conjunto de servidores de controle de rede, interage com tais servidores para estabelecer um provisionamento de largura de banda global em âmbito de rede para grupos de fluxos de aplicações. Com a B4, a SDN deu um salto importante e entrou nas redes operacionais de um provedor de rede global. Para uma descrição detalhada da rede B4, consulte Jain (2013) e Hong (2018).

5.5.3 Interação entre o plano de dados e o de controle: Um exemplo

Para consolidar o nosso entendimento sobre a interação entre comutadores controlados por SDN e o controlador SDN, vamos considerar o exemplo mostrado na Figura 5.16, no qual o algoritmo de Dijkstra (que estudamos na Seção 5.2) é usado para determinar as rotas de caminho mais curto. O cenário SDN da Figura 5.16 possui duas diferenças importantes em relação ao cenário anterior de controle por roteador das Seções 5.2.1 e 5.3, no qual o algoritmo de Dijkstra foi implementado em todos os roteadores, e atualizações de estado de enlace eram inundadas entre todos os roteadores da rede:

- O algoritmo de Dijkstra é executado como uma aplicação separada, externa aos comutadores de pacotes.
- Os comutadores de pacotes enviam atualizações de enlaces para o controlador SDN, e não uns para os outros.

Neste exemplo, vamos pressupor que o enlace entre os comutadores s1 e s2 caia; que o roteamento de caminho mais curto é implementado e, por consequência, que as regras de repasse do fluxo que entra e que sai em s1, s3 e s4 são afetadas, mas a operação de s2 não muda. Vamos pressupor também que o OpenFlow é usado como protocolo da camada de comunicação, e que o plano de controle não tem nenhuma outra função além do roteamento de estado de enlace.

1. O comutador s1, sofrendo falha no enlace com s2, avisa o controlador SDN sobre a mudança no estado de enlace usando a mensagem de *estado da porta* (*port-status*) do OpenFlow.

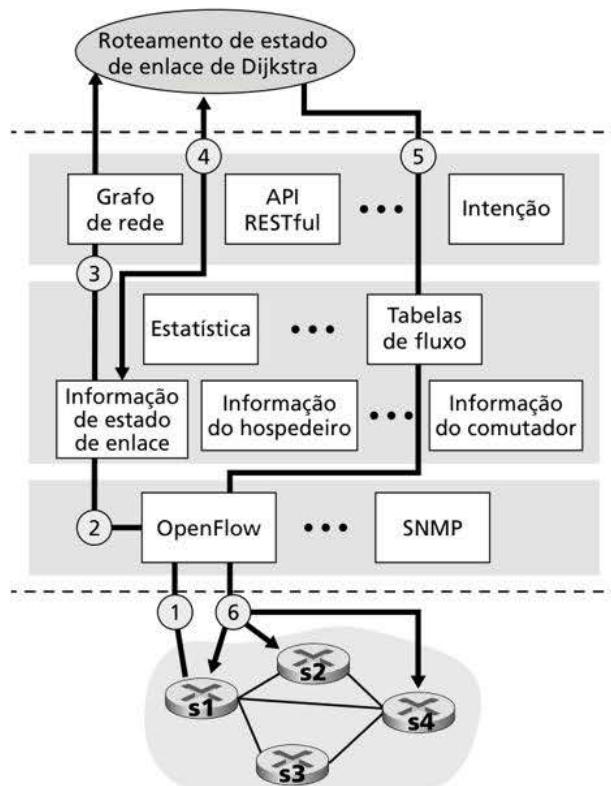


Figura 5.16 Cenário de controlador SDN: mudança de estado de enlace.

2. O controlador SDN recebe a mensagem do OpenFlow indicando a mudança do estado de enlace e avisa o gerenciador de estado de enlace, que atualiza um banco de dados de estados de enlace.
3. A aplicação de controle de rede que implementa o roteamento de estado de enlace de Dijkstra registrou-se anteriormente para ser notificada quando ocorre uma mudança no estado de enlace. Essa aplicação recebe a notificação sobre a mudança no estado de enlace.
4. A aplicação de roteamento de estado de enlace interage com o gerenciador de estado de enlace para obter o estado atualizado; ela também poderia consultar outros componentes na camada de gerenciamento do estado. A seguir, ela calcula os novos caminhos de menor custo.
5. A aplicação de roteamento de estado de enlace interage então com o gerenciador da tabela de fluxo, que determina que as tabelas de fluxo sejam atualizadas.
6. O gerenciador de tabela de fluxo utiliza o protocolo OpenFlow para atualizar as linhas da tabela de fluxo nos comutadores afetados – s1 (que agora roteia pacotes destinados para s2 através de s4), s2 (que agora começará a receber pacotes de s1 através do comutador intermediário s4) e s4 (que agora deve repassar pacotes de s1 destinados a s2).

O exemplo é simples, mas ilustra como o plano de controle da SDN fornece serviços do plano de controle (nesse caso, roteamento da camada de rede) antes implementados com controle por roteador exercido em cada um dos roteadores da rede. É fácil entender como um ISP com SDN poderia facilmente passar do roteamento por caminho de menor custo para uma abordagem mais customizada de roteamento. Na verdade, como o controlador pode adaptar as tabelas de fluxo como bem entende, é possível implementar *qualquer* forma de repasse que se desejar, basta alterar o software de controle de aplicação. Essa facilidade de mudança deve ser comparada com o caso do plano de controle com controle por roteador tradicional, no qual é preciso alterar o software em todos os roteadores (que o ISP pode adquirir de múltiplos fornecedores independentes).

5.5.4 SDN: passado e futuro

O interesse intenso por SDN é um fenômeno relativamente recente, mas as raízes técnicas das SDNs e a separação dos planos de dados e de controle em especial são muito mais antigas. Em 2004, Feamster (2004), Lakshman (2004) e o RFC 3746 defenderam a separação dos planos de dados e de controle da rede. Conforme van der Merwe (1998), a estrutura de controle para redes ATM (Black, 1995) com múltiplos controladores, cada um controlando um determinado número de comutadores ATM. O projeto Ethane (Casado, 2007) foi pioneiro no conceito de uma rede de comutadores de Ethernet simples baseados em fluxo, com tabelas de fluxo de combinação mais ação, um controlador centralizado que gerencia a admissão de fluxo e o roteamento, e o repasse de pacotes que não combinam do comutador para o controlador. Uma rede com mais de 300 comutadores Ethane estava em operação em 2007. O Ethane logo evoluiu e se transformou no projeto OpenFlow, e o resto, como dizem, é história!

Diversos esforços de pesquisa se concentram no desenvolvimento de arquiteturas de capacidades de SDN futuras. Como vimos, a revolução SDN está levando à substituição disruptiva de comutadores e roteadores dedicados (com planos de dados e de controle) por hardware de comutação genérico simples e um plano de controle por software sofisticado. Da mesma forma, uma generalização da SDN, chamada de virtualização das funções de rede (NFV, do inglês *network functions virtualization*) (que discutimos anteriormente na Seção 4.5), quer promover a substituição disruptiva de *middleboxes* sofisticadas (como aquelas com hardware dedicado e software proprietário para serviços/cache de mídia) por comutação, armazenamento e servidores genéricos simples. Uma segunda área de pesquisa importante busca estender os conceitos de SDN do contexto intra-AS para o âmbito inter-AS (Gupta, 2014).

PRINCÍPIOS NA PRÁTICA

ESTUDOS DE CASO DE CONTROLADORES SDN: OS CONTROLADORES OPENDAYLIGHT E ONOS

Nos primórdios da SDN, havia um único protocolo SDN (OpenFlow [McKeown, 2008; OpenFlow, 2009]) e um único controlador SDN (NOX [Gude, 2008]). Desde então, o número de controladores SDN cresceu de forma especialmente significativa (Kreutz, 2015). Alguns controladores SDN são proprietários e específicos de algumas empresas, principalmente quando usados para controlar redes proprietárias internas (p. ex., dentro ou entre os datacenters de uma empresa). Mas muitos mais controladores são de código aberto e implementados em diversas linguagens de programação (Erickson, 2013). Mais recentemente, os controladores OpenDaylight (OpenDaylight, 2020) e ONOS (ONOS, 2020) conquistaram apoio significativo no setor. Ambos são de código aberto e foram desenvolvidos em parceria com a Linux Foundation.

O controlador OpenDaylight

A Figura 5.17 é uma representação simplificada da plataforma do controlador OpenDaylight (ODL) (OpenDaylight, 2020; Eckel, 2017).

As Funções Básicas de Rede do ODL estão no âmbito do controlador e correspondem muito bem às capacidades de gerenciamento de estado em âmbito de rede que encontramos na Figura 5.15. A camada de

abstração de serviço (SAL, do inglês *Service Abstraction Layer*) é o centro nervoso do controlador, permitindo que os componentes e as aplicações do controlador invoquem os serviços uns dos outros, acessem dados de configuração e operacionais e subscrevam os eventos que geram. A SAL também oferece uma interface abstrata uniforme para protocolos específicos que operam entre o controlador ODL e os dispositivos controlados. Esses protocolos incluem o OpenFlow (que analisamos na Seção 4.5), o Protocolo Simples de Gerenciamento de Rede (SNMP) e o Protocolo de Configuração de Rede (NETCONF, do inglês *Network Configuration Protocol*), ambos os quais analisaremos na Seção 5.7. O Open vSwitch Database Management Protocol (OVSDB) é usado para gerenciar comutação em datacenters, uma área de aplicação importante para a tecnologia SDN. Apresentaremos as redes de datacenters no Capítulo 6.

Orquestrações e aplicações de rede determinam como o repasse do plano de dados e outros serviços, tais como firewalls e balanceamento de carga, são realizados nos dispositivos controlados. O ODL oferece duas maneiras para as aplicações interoperarem com serviços controladores nativos (e, logo, dispositivos) e uns com os outros. Na abordagem orientada por API (AD-SAL – *API-Driven*), mostrada na Figura 5.17, as aplicações se comunicam com os módulos controladores usando um API de requisição-resposta REST que roda sobre

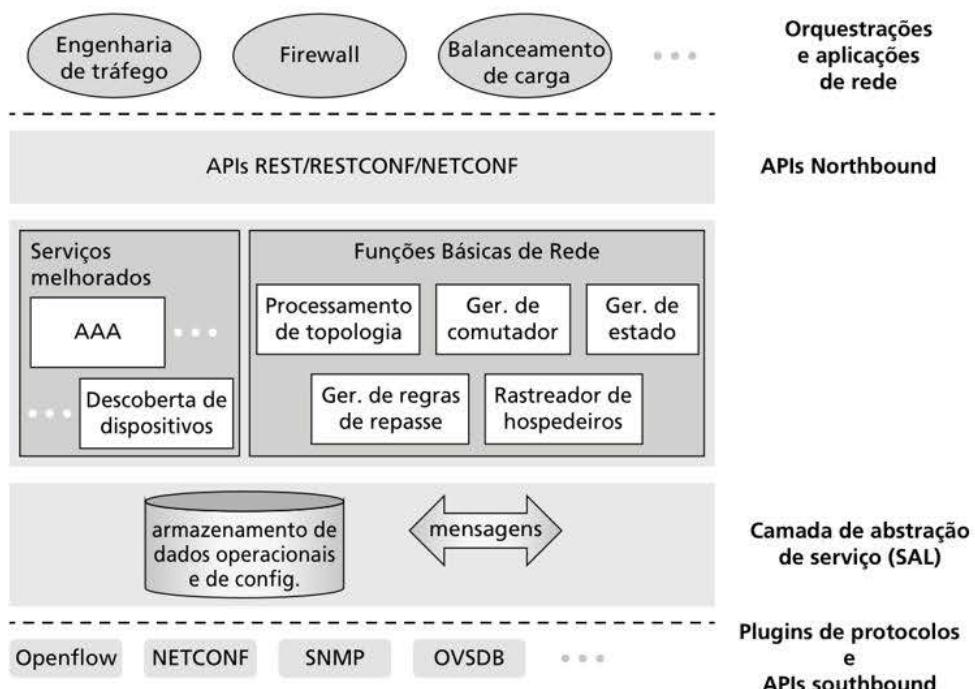


Figura 5.17 Representação simplificada do controlador OpenDaylight.

HTTP. As versões iniciais do controlador OpenDaylight permitiam apenas a AD-SAL. À medida que o ODL foi sendo mais usado na configuração e no gerenciamento de rede, versões posteriores introduziram a abordagem orientada por modelo (MD-SAL – *Model-Driven*). Nela, a linguagem de modelagem de dados YANG (RFC 6020) define modelos de dados de estado operacional e de configuração de redes, protocolos e dispositivos. Os dispositivos são então configurados e gerenciados pela manipulação desses dados usando o protocolo NETCONF.

O controlador ONOS

A Figura 5.18 apresenta uma visão simplificada do controlador ONOS (ONOS, 2020). Semelhante ao controlador canônico da Figura 5.15, podemos identificar três camadas no controlador ONOS:

- *Abstrações e protocolos northbound*. Uma característica exclusiva do ONOS é a sua estrutura de intenção, que permite que uma aplicação requisite um serviço de alto nível (p. ex., configurar uma conexão entre os hospedeiros A e B, ou, ao contrário, não permitir que os hospedeiros A e B se comuniquem) sem precisar conhecer os detalhes de como esse serviço é realizado. As informações de estado são dadas às aplicações de controle de

rede em toda a API northbound de forma síncrona (por consulta) ou assíncrona (por callbacks [notificações] de escuta, p. ex., quando o estado da rede muda).

- *Núcleo distribuído*. O estado dos enlaces, hospedeiros e dispositivos da rede é mantido no núcleo distribuído do ONOS. O ONOS é implementado como um serviço em um conjunto de servidores interconectados, cada um dos quais executa uma cópia idêntica do software ONOS; um número maior de servidores oferece uma capacidade de serviço maior. O núcleo ONOS oferece os mecanismos para replicação de serviços e coordenação entre instâncias, fornecendo às aplicações acima e aos dispositivos de rede abaixo a abstração de serviços de núcleo logicamente centralizados.
- *Protocolos e abstrações southbound*. As abstrações southbound disfarçam a heterogeneidade dos hospedeiros, enlaces, comutadores e protocolos subjacentes, permitindo que o núcleo distribuído seja agnóstico em termos de dispositivo e de protocolo. Devido a essa abstração, a interface southbound abaixo do núcleo distribuído está logicamente mais alta do que no nosso controlador canônico da Figura 5.14 ou o controlador ODL na Figura 5.17.

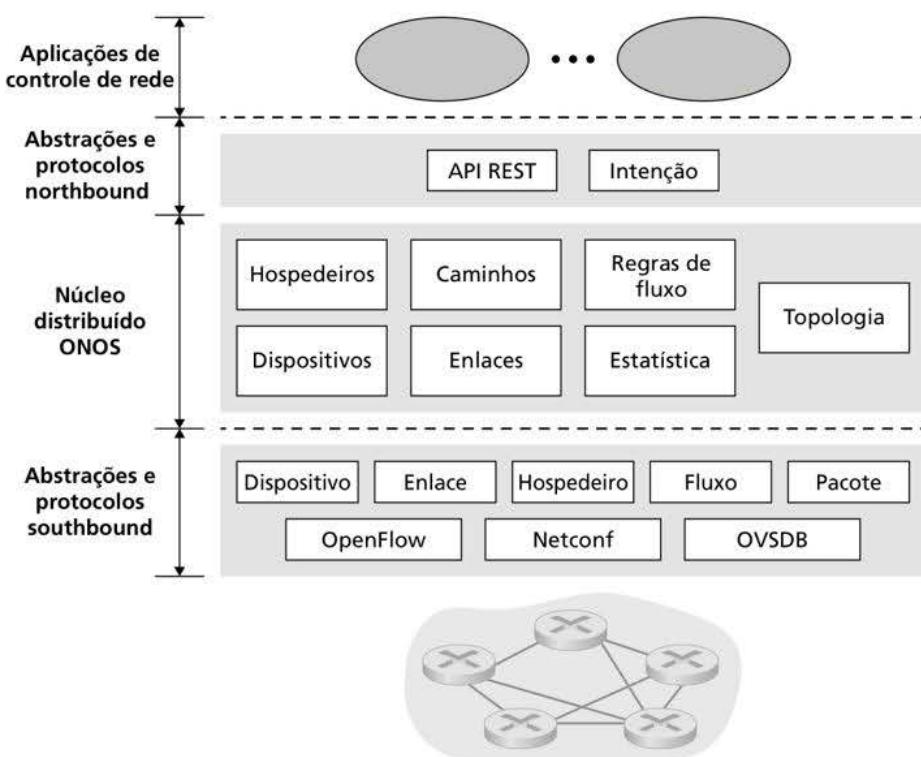


Figura 5.18 Arquitetura do controlador ONOS.

5.6 ICMP: PROTOCOLO DE MENSAGENS DE CONTROLE DA INTERNET

O ICMP, especificado no (RFC 792), é usado por hospedeiros e roteadores para comunicar informações de controle da camada de rede entre si. A utilização mais comum do ICMP é para comunicação de erros. Por exemplo, ao rodar uma sessão HTTP, é possível que você já tenha encontrado uma mensagem de erro como “Rede de destino inalcançável”. Essa mensagem teve sua origem no ICMP. Em algum ponto, um roteador IP não conseguiu descobrir um caminho para o hospedeiro especificado em sua requisição HTTP. O roteador criou e enviou uma mensagem ICMP tipo 3 a seu hospedeiro indicando o erro.

O ICMP é com frequência considerado parte do IP, mas, em termos de arquitetura, está logo acima, pois mensagens ICMP são carregadas dentro de datagramas IP. Isto é, mensagens ICMP são carregadas como carga útil IP, exatamente como segmentos TCP ou Protocolo de Datagrama de Usuário (UDP, do inglês *User Datagram Protocol*), que também o são. De maneira semelhante, quando um hospedeiro recebe um datagrama IP com ICMP especificado como protocolo de camada superior (um número do protocolo da camada superior de 1), ele demultiplexa o conteúdo do datagrama para ICMP, exatamente como demultiplexaria o conteúdo de um datagrama para TCP ou UDP.

Mensagens ICMP têm um campo de tipo e um campo de código. Além disso, contêm o cabeçalho e os primeiros 8 bytes do datagrama IP que causou a criação da mensagem ICMP em primeiro lugar (de modo que o remetente pode determinar o datagrama que causou o erro). Alguns tipos de mensagens ICMP selecionadas são mostrados na Figura 5.19. Note que mensagens ICMP não são usadas somente para sinalizar condições de erro.

O conhecido programa ping envia uma mensagem ICMP do tipo 8, código 0, para o hospedeiro especificado. O hospedeiro de destino, ao ver a solicitação de eco, devolve uma resposta de eco ICMP do tipo 0, código 0. A maior parte das execuções de TCP/IP suporta o servidor ping diretamente no sistema operacional; isto é, o servidor não é um processo. O Capítulo 11 de Stevens (1990) fornece o código-fonte para o programa ping cliente. Note

Tipo de ICMP	Código	Descrição
0	0	resposta de eco (para ping)
3	0	rede de destino inalcançável
3	1	hospedeiro de destino inalcançável
3	2	protocolo de destino inalcançável
3	3	porta de destino inalcançável
3	6	rede de destino desconhecida
3	7	hospedeiro de destino desconhecido
4	0	repressão da origem (controle de congestionamento)
8	0	solicitação de eco
9	0	anúncio do roteador
10	0	descoberta do roteador
11	0	TTL expirado
12	0	cabeçalho IP inválido

Figura 5.19 Tipos de mensagens ICMP.

que o programa cliente tem de ser capaz de instruir o sistema operacional para que ele gere uma mensagem ICMP do tipo 8, código 0.

Outra mensagem ICMP interessante é a de redução da origem (“source quench”). Essa mensagem é pouco usada na prática. Sua finalidade original era realizar controle de congestionamento – permitir que um roteador congestionado enviasse uma mensagem ICMP de redução da origem a um hospedeiro para obrigar-lo a reduzir sua velocidade de transmissão. Vimos no Capítulo 3 que o TCP tem seu próprio mecanismo de controle de congestionamento, que funciona na camada de transporte, e os bits de Notificação Explícita de Congestionamento podem ser usados por dispositivos da camada de rede para sinalizar congestão.

No Capítulo 1, apresentamos o programa Traceroute, que nos permite acompanhar a rota de um hospedeiro a qualquer outro hospedeiro no mundo. O interessante é que o Traceroute é executado com mensagens ICMP. Para determinar os nomes e endereços de roteadores entre a origem e o destino, o Traceroute da origem envia uma série de datagramas comuns ao destino. O primeiro deles tem um tempo de vida (TTL, do inglês *time-to-live*) de 1, o segundo tem um TTL de 2, o terceiro tem um TTL de 3 e assim por diante. A origem também aciona temporizadores para cada datagrama. Quando o enésimo datagrama chega ao enésimo roteador, este observa que o TTL do datagrama acabou de expirar. Segundo as regras do protocolo IP, o roteador descarta o datagrama e envia uma mensagem ICMP de aviso à origem (tipo 11, código 0). Essa mensagem de aviso inclui o nome do roteador e seu endereço IP. Quando chega à origem, a mensagem obtém, do temporizador, o tempo de viagem de ida e volta e, da mensagem ICMP, o nome e o endereço IP do enésimo roteador.

Como uma origem de Traceroute sabe quando parar de enviar segmentos UDP? Lembre-se de que a origem incrementa o campo do TTL para cada datagrama que envia. Assim, um deles conseguirá enfim chegar ao hospedeiro de destino. Como tal datagrama contém um segmento UDP com um número de porta improvável, o hospedeiro de destino devolve à origem uma mensagem ICMP indicando que a porta não pôde ser alcançada (mensagem tipo 3, código 3). Quando recebe essa mensagem ICMP particular, o hospedeiro de origem sabe que não precisa enviar mais pacotes de sondagem. (Na verdade, o programa Traceroute padrão envia conjuntos de três pacotes com o mesmo TTL; assim, a saída de Traceroute oferece três resultados para cada TTL.)

Desse modo, o hospedeiro de origem fica a par do número e das identidades de roteadores que estão entre ele e o hospedeiro de destino e o tempo de viagem de ida e volta entre os dois. Note que o programa cliente Traceroute tem de ser capaz de instruir o sistema operacional para que este gere datagramas UDP com valores específicos de TTL, assim como tem de poder ser avisado por seu sistema operacional quando chegam mensagens ICMP. Agora que você entende como o Traceroute funciona, é provável que queira voltar e brincar um pouco com ele.

Uma nova versão do ICMP foi definida para o IPv6 no RFC 4443. Além da reorganização das definições existentes de tipos e códigos ICMP, o ICMPv6 adicionou novos tipos e códigos exigidos pela nova funcionalidade do IPv6. Entre eles, estão incluídos o tipo “Pacote muito grande” e um código de erro “opções IPv6 não reconhecidas”.

5.7 GERENCIAMENTO DE REDE E SNMP, NETCONF/YANG

Tendo chegado ao fim do nosso estudo sobre a camada de rede, com apenas a camada de enlace à nossa frente, estamos agora conscientes de que uma rede consiste em muitas peças complexas de hardware e software que interagem umas com as outras – desde os enlaces, comutadores, roteadores, hospedeiros e outros dispositivos, que são os componentes físicos, até os muitos protocolos que controlam e coordenam esses componentes. Quando centenas ou milhares de componentes são montados em conjunto por alguma organização para formar

uma rede, o trabalho do administrador da rede, de mantê-la funcionando, não pode deixar de ser um desafio. Vimos na Seção 5.5 que um controlador logicamente centralizado pode ajudar com esse processo no contexto SDN. Mas o desafio do gerenciamento de rede já existia muito antes da SDN, com um amplo conjunto de abordagens e ferramentas especializadas que ajudam o administrador a monitorar, administrar e controlar a rede. Nesta seção, estudaremos essas técnicas e ferramentas, assim como outras que evoluíram em conjunto com a SDN.

Uma pergunta frequente é: “O que é gerenciamento de rede?”. Saydam (1996) oferece uma definição de gerenciamento de rede em uma única frase (embora um tanto longa):

Gerenciamento de rede inclui a implementação, a integração e a coordenação de elementos de hardware, software e humanos, para monitorar, testar, consultar, configurar, analisar, avaliar e controlar os recursos da rede, e de elementos, para satisfazer às exigências operacionais, de desempenho e de qualidade de serviço em tempo real a um custo razoável.

Dada essa definição ampla, trabalharemos apenas os rudimentos do gerenciamento de rede nesta seção – a arquitetura, os protocolos e os dados usados por um administrador de rede para realizar a sua tarefa. Não analisaremos os processos de tomada de decisão do administrador, por meio dos quais são considerados tópicos como a identificação de falhas (Labovitz, 1997; Steinder, 2002; Feamster, 2005; Wu, 2005; Teixeira, 2006), detecção de anomalias (Lakhina, 2005; Barford, 2009), projeto/engenharia de rede para atender aos Acordos de Nível de Serviços (SLA, do inglês *Service Level Agreements*) (Huston, 1999a), entre outros. Nosso foco é intencionalmente limitado; os leitores interessados devem consultar as referências acima, as excelentes visões gerais em Subramanian (2000), Schonwalder (2010) e Claise (2019) e o tratamento mais detalhado sobre gerenciamento de rede disponível no site do livro.

5.7.1 A estrutura de gerenciamento de rede

A Figura 5.20 mostra os componentes fundamentais do gerenciamento de rede:

- *Servidor gerenciador.* O servidor gerenciador é uma aplicação que em geral tem **gerentes de rede** (humanos) no circuito e que é executada em uma estação central de gerenciamento de rede no centro de operações de rede (NOC, do inglês *network operations center*). O servidor gerenciador é o centro da atividade de gerenciamento de rede; ele controla a coleta, o processamento, a análise e a transmissão de informações e comandos de gerenciamento de rede. É nele que são iniciadas ações para configurar, monitorar e controlar os dispositivos gerenciados da rede. Na prática, uma rede pode ter diversos servidores gerenciadores.
- *Dispositivo gerenciado.* Um dispositivo gerenciado é um equipamento de rede (incluindo o seu software) que reside em uma rede gerenciada. Um dispositivo gerenciado pode ser um hospedeiro, um roteador, comutador, middlebox, modem, termômetro ou outro dispositivo conectado à rede. O dispositivo em si possui muitos componentes gerenciáveis (p. ex., uma interface de rede é apenas um entre vários componentes de um hospedeiro ou roteador) e parâmetros de configuração para esses componentes de hardware e software (p. ex., um protocolo de roteamento intra-AS, tal como OSPF).
- *Dados.* Cada dispositivo gerenciado possui dados, também chamados de “estado”, associados a ele. Existem diversos tipos de dados. Os **dados de configuração** são informações do dispositivo configuradas explicitamente pelo gerenciador da rede, por exemplo, uma velocidade de interface ou endereço IP configurado/designado pelo gerenciador para uma interface do dispositivo. Os **dados operacionais** são informações que o dispositivo adquire à medida que opera; por exemplo, a lista de vizinhos imediatos no protocolo OSPF. As **estatísticas do dispositivo** são contadores e indicadores de estado atualizados à medida que o dispositivo opera (p. ex., o número de pacotes descartados

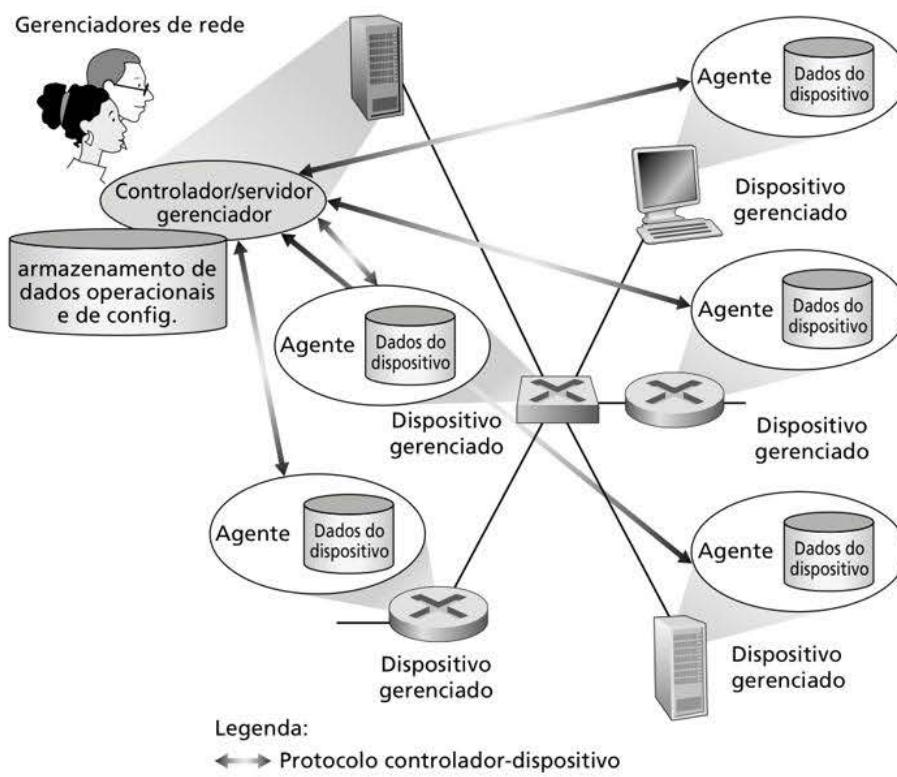


Figura 5.20 Elementos do gerenciamento de rede.

em uma interface ou a velocidade da ventoinha do dispositivo). O gerenciador da rede pode consultar os dados de dispositivos remotos e, em alguns casos, controlá-los com a gravação de valores de dados do dispositivo, como veremos abaixo. Na Figura 5.17, o servidor gerenciador também mantém a sua própria cópia dos dados de configuração, operacionais e estatísticos dos seus dispositivos gerenciados, além de dados em âmbito de rede (p. ex., a topologia da rede).

- *Agente de gerenciamento de rede.* O agente de gerenciamento de rede é um processo de software executado no dispositivo gerenciado que se comunica com o servidor gerenciador, adotando ações locais no dispositivo gerenciado sob o comando e controle do servidor gerenciador. O agente de gerenciamento de rede é semelhante ao agente de roteamento que vimos na Figura 5.2.
- *Protocolo de gerenciamento de rede.* O último componente da estrutura de gerenciamento de rede é o protocolo de gerenciamento de rede. Esse protocolo é executado entre o servidor gerenciador e os dispositivos gerenciados, o que permite que o servidor gerenciador investigue o estado dos dispositivos gerenciados e execute ações sobre eles mediante seus agentes. Estes podem usar o protocolo de gerenciamento de rede para informar ao servidor gerenciador a ocorrência de eventos excepcionais (p. ex., falhas de componentes ou violação de patamares de desempenho). É importante notar que o protocolo de gerenciamento de rede em si não gerencia a rede. Em vez disso, ele fornece uma ferramenta com a qual os administradores podem gerenciar (“monitorar, testar, consultar, configurar, analisar, avaliar e controlar”) a rede. É uma distinção sutil, mas importante.

Na prática, os operadores têm três meios comuns para gerenciar a rede usando os componentes descritos acima:

- *CLI.* Um operador pode emitir comandos da **Interface de Linha de Comando (CLI)**, do inglês **Command Line Interface**) diretos do dispositivo. Esses comandos podem ser

inseridos diretamente no console de um dispositivo gerenciado (se o operador estiver fisicamente presente no dispositivo) ou por uma conexão Telnet ou shell seguro (SSH, do inglês *secure shell*), possivelmente através de scripts, entre o controlador/servidor gerenciador e o dispositivo gerenciado. Os comandos CLI são específicos de cada dispositivo e fornecedor e podem ser obscuros e complexos. Os magos das redes mais experientes podem saber usá-los para configurar perfeitamente os dispositivos, mas a CLI tende a produzir erros e é difícil de automatizar ou ter a escala ampliada para redes maiores. Dispositivos de rede orientados para o público em geral, como o seu roteador sem fio doméstico, podem exportar um menu de gerenciamento que você (o gerenciador da rede!) pode acessar por HTTP para configurá-lo. Essa abordagem pode funcionar bem para dispositivos simples e isolados e ser menos propensa ao erro do que a CLI, mas não pode ser ampliada para redes de maior porte.

- **SNMP/MIB.** Nessa abordagem, o operador de rede pode consultar/configurar os dados contidos nos objetos da **Base de Informações de Gerenciamento (MIB**, do inglês *Management Information Base*) usando o **SNMP**. Algumas MIBs são específicas de dispositivos e fornecedores, enquanto outras (p. ex., o número de datagramas IP descartados em um roteador devido a erros no cabeçalho do datagrama IP ou o número de segmentos UDP recebidos em um hospedeiro) independem do dispositivo, oferecendo abstração e generalidade. Um operador de rede normalmente usaria essa abordagem para consultar e monitorar estatísticas de dispositivos e estado operacional, e então usaria a CLI para controlar/configurar ativamente o dispositivo. É importante observar que ambas as abordagens gerenciam os dispositivos *individualmente*. Analisaremos o SNMP e as MIBs, em uso desde o final da década de 1980, na Seção 5.7.2 a seguir. Um workshop sobre gerenciamento de rede organizado pelo Internet Architecture Board em 2002 (RFC 3535) observou o valor da abordagem SNMP/MIB para o monitoramento de dispositivos e as suas desvantagens, especialmente para a configuração de dispositivos e gerenciamento de rede em maior escala. Isso deu origem à abordagem mais recente para gerenciamento de rede, usando NETCONF e YANG.
- **NETCONF/YANG.** A abordagem NETCONF/YANG adota uma perspectiva mais abstrata, holística e de nível de rede em relação ao gerenciamento de rede, com uma ênfase muito maior no gerenciamento de configuração, incluindo a especificação de limites de precisão e a oferta de operações de gerenciamento atômicas em múltiplos dispositivos controlados. O **YANG** (RFC 6020) é uma linguagem de modelagem de dados usada para modelar dados de configuração e operacionais. O protocolo **NETCONF** (RFC 6241) é usado para comunicar ações e dados compatíveis com YANG de e para dispositivos remotos. Encontramos o NETCONF e o YANG brevemente em nosso estudo de caso sobre o controlador OpenDaylight na Figura 5.17 e o estudaremos na Seção 5.7.3, a seguir.

5.7.2 O Protocolo Simples de Gerenciamento de Rede (SNMP) e a base de informações de gerenciamento (MIB)

O **Protocolo Simples de Gerenciamento de Rede** versão 3 (SNMPv3) (RFC 3410) é um protocolo da camada de aplicação usado para transportar mensagens com informações de controle de gerenciamento de rede entre um servidor gerenciador e um agente que executa em seu nome. A utilização mais comum do SNMP é em um modo comando-resposta, no qual o servidor gerenciador SNMP envia uma requisição a um agente SNMP, que a recebe, realiza alguma ação e envia uma resposta à requisição. Em geral, uma requisição é usada para consultar (recuperar) ou modificar (definir) valores de objetos MIB associados a um dispositivo gerenciado. Um segundo uso comum do SNMP é para um agente enviar uma mensagem não solicitada, conhecida como mensagem trap,* ao servidor gerenciador.

*N. de T.: O nome “trap” foi substituído por “notifications” (notificações).

As mensagens trap são usadas para notificar um servidor gerenciador de uma situação excepcional (p. ex., uma interface de enlace interrompida ou ativada) que resultou em mudanças nos valores dos objetos MIB.

Os objetos MIB são especificados em uma linguagem de descrição de dados chamada SMI (*Structure of Management Information*) (RFC 2578; RFC 2579; RFC 2580), um componente da estrutura de gerenciamento de rede cujo nome esquisito não informa nada sobre a sua funcionalidade. Uma linguagem de definição formal é utilizada para garantir que a sintaxe e a semântica dos dados de gerenciamento de rede sejam claramente definidos e não tenham ambiguidades. Objetos MIB relacionados são reunidos em módulos MIB. Até o final de 2019, havia mais de 400 RFCs relacionados a MIB e um número muito maior de módulos MIB específicos para fornecedores (privados).

O SNMPv3 define sete tipos de mensagens, conhecidas genericamente como PDUs (do inglês *protocol data units* – unidades de dados do protocolo), conforme apresentado na Tabela 5.2 e descrito em seguida. O formato da PDU pode ser visto na Figura 5.21.

- As PDUs GetRequest, GetNextRequest e GetBulkRequest são enviadas de um servidor gerenciador a um agente para requisitar o valor de um ou mais objetos MIB no dispositivo gerenciado do agente. Os objetos MIB cujos valores estão sendo requisitados são especificados na parte de vinculação de variáveis da PDU. GetRequest, GetNextRequest e GetBulkRequest diferem no grau de especificidade de seus pedidos de dados. GetRequest pode requisitar um conjunto arbitrário de valores MIB; múltiplas GetNextRequest podem ser usadas para percorrer a sequência de uma lista ou tabela de objetos MIB, e GetBulkRequest permite que um grande bloco de dados seja devolvido, evitando a sobrecarga incorrida quando tiverem de ser enviadas múltiplas mensagens GetRequest ou GetNextRequest. Em todos os três

TABELA 5.2 Tipos de PDU SNMPv3

Tipo de SNMPv3-PDU	Remetente-receptor	Descrição
GetRequest	gerente a agente	pega valor de uma ou mais instâncias de objetos MIB
GetNextRequest	gerente a agente	pega valor da próxima instância de objeto MIB na lista ou tabela
GetBulkRequest	gerente a agente	pega valores em grandes blocos de dados, por exemplo, valores em uma grande tabela
InformRequest	gerente a gerente	informa à entidade gerenciadora remota valores da MIB que são remotos para seu acesso
SetRequest	gerente a agente	define valores de uma ou mais instâncias de objetos MIB
Response	agente a gerente ou gerente a gerente	gerado em resposta a GetRequest, GetNextRequest, GetBulkRequest, SetRequest e InformRequest
SNMPv2-Trap	agente a gerente	informar gerenciador sobre evento excepcional

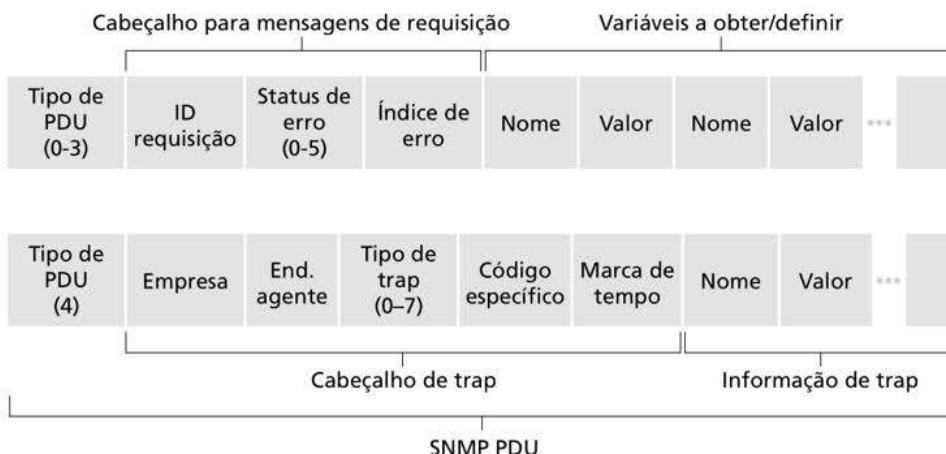


Figura 5.21 Formato da PDU SNMP.

casos, o agente responde com uma PDU Response que contém os identificadores de objetos e seus valores associados.

- A PDU SetRequest é usada por um servidor gerenciador para estabelecer o valor de um ou mais objetos MIB em um dispositivo gerenciado. Um agente responde com uma PDU Response que contém uma mensagem de estado de erro “noError” para confirmar que o valor realmente foi estabelecido.
- A PDU InformRequest é usada por um servidor gerenciador para comunicar a outro servidor gerenciador informações MIB remotas ao servidor receptor.
- A PDU Response normalmente é enviada de um dispositivo gerenciado a um servidor gerenciador em resposta a uma mensagem de requisição desse servidor, retornando as informações requisitadas.
- O tipo final de PDU SNMPv3 é a mensagem trap. Mensagens trap são geradas assincronamente, isto é, não são geradas em resposta a uma requisição recebida, mas em resposta a um evento para o qual o servidor gerenciador requer notificação. O RFC 3418 define tipos conhecidos de trap que incluem uma partida a frio ou a quente realizada por um dispositivo, a ativação ou interrupção de um enlace, a perda de um vizinho ou um evento de falha de autenticação. Uma requisição de trap recebida não exige resposta de um servidor gerenciador.

Sabendo da natureza comando-resposta do SNMP, convém observar que, embora as PDUs SNMP possam ser transportadas por muitos protocolos de transporte diferentes, elas normalmente são transportadas na carga útil de um datagrama UDP. Na verdade, o RFC 3417 estabelece que o UDP é o “mapeamento de transporte preferencial”. Já que o UDP é um protocolo de transporte não confiável, não há garantia de que um comando ou sua resposta será recebido no destino pretendido. O campo ID requisição da PDU (ver Figura 5.21) é usado pelo servidor gerenciador para numerar as requisições que faz a um agente; a resposta de um agente adota a ID requisição copiada do comando recebido. Assim, o campo ID requisição pode ser usado pelo servidor gerenciador para detectar comandos ou respostas perdidos. Cabe ao servidor gerenciador decidir se retransmitirá um comando se nenhuma resposta correspondente for recebida após determinado período. Em particular, o padrão SNMP não impõe nenhum procedimento específico de retransmissão, nem mesmo diz que o comando deve ser enviado em primeiro lugar. Ele requer apenas que o servidor gerenciador “aja com responsabilidade em relação à frequência e à duração das retransmissões”. Isso, é claro, nos leva a pensar como deve agir um protocolo “responsável”!

O SNMP evoluiu em suas três versões. Os projetistas do SNMPv3 têm dito que o “SNMPv3 pode ser considerado um SNMPv2 com capacidades adicionais de segurança

e de administração” (RFC 3410). Certamente, há mudanças no SNMPv3 em relação ao SNMPv2, mas em nenhum lugar elas são mais evidentes do que nas áreas da administração e da segurança. O papel central da segurança no SNMPv3 é de particular importância, já que a falta de segurança adequada resultava no uso do SNMP primordialmente para monitorar, em vez de controlar (p. ex., SetRequest é pouquíssimo usada no SNMPv1). Mais uma vez, vemos que a segurança – tópico que analisaremos em detalhes no Capítulo 8 – é uma preocupação crítica, mas de novo uma preocupação cuja importância só foi percebida tardiamente e apenas “adicionada” ao que já havia sido criado.

Base de Informações de Gerenciamento (MIB)

Vimos anteriormente que os dados de estado operacional de um dispositivo gerenciado (e, em parte, seus dados de configuração) são representados, na abordagem SNMP/MIB ao gerenciamento de rede, como objetos reunidos em um MIB para o dispositivo. Um objeto MIB pode ser um contador, tal como o número de datagramas IP descartados em um roteador em virtude de erros em cabeçalhos de datagramas IP ou o número de erros de detecção de portadora em uma placa de interface Ethernet; um conjunto de informações descritivas, como a versão do software que está sendo executado em um servidor DNS; informações de estado, como se um determinado dispositivo está funcionando corretamente; ou informações específicas sobre protocolos, como um caminho de roteamento até um destino. Os diversos RFCs da Força de Trabalho de Engenharia da Internet (IETF, do inglês *Internet Engineering Task Force*) definem mais de 400 módulos MIB, e há muitos mais específicos para determinados dispositivos e fornecedores. O RFC 4293 especifica o módulo MIB que define objetos gerenciados (incluindo ipSystemStatsInDelivers) para gerenciar implementações do Protocolo da Internet (IP) e o seu ICMP associado. O RFC 4022 especifica o módulo MIB para TCP, enquanto o RFC 4113 especifica o módulo MIB para UDP.

Os RFCs relativos a MIB são uma leitura tediosa, mas ainda seria instrutivo (i.e., como comer a salada, “faz bem para você”) considerar um exemplo de um objeto MIB. A definição do tipo de objeto ipSystem-StatsInDelivers do RFC 4293 define um contador somente para a leitura de 32 bits que registra o número de datagramas IP recebidos pelo dispositivo gerenciado e entregues com sucesso a um protocolo da camada superior. No exemplo abaixo, o Counter32 é um dos tipos de dados básicos definidos no SMI.

```

ipSystemStatsInDelivers OBJECT-TYPE
  SYNTAX Counter32
  MAX-ACCESS read-only
  STATUS current
  DESCRIPTION
    "The total number of datagrams successfully
     delivered to IPuser-protocols (including
     ICMP).
  When tracking interface statistics, the
  counter of the interface to which these
  datagrams were addressed is incremented. This
  interface might not be the same as the input
  interface for some of the datagrams.
  Discontinuities in the value of this
  counter can occur at re-initialization
  of the management system, and at other
  times as indicated by the value of
  ipSystemStatsDiscontinuityTime."
 ::= { ipSystemStatsEntry 18 }

```

5.7.3 O Protocolo de Configuração de Rede (NETCONF) e YANG

O protocolo NETCONF opera entre o servidor gerenciador e os dispositivos de rede gerenciados, oferecendo um serviço de mensagens para (i) recuperar, definir e modificar dados de configuração em dispositivos gerenciados; (ii) consultar estatísticas e dados operacionais em dispositivos gerenciados; e (iii) inscrever para receber notificações geradas pelos dispositivos gerenciados. Para controlar ativamente o dispositivo gerenciado, o servidor gerenciador envia configurações especificadas em um documento XML estruturado e ativa configurações no dispositivo gerenciado. O NETCONF usa uma chamada de procedimento remoto (RPC, do inglês *remote procedure call*), na qual mensagens de protocolo também são codificadas em formato XML e trocadas entre o servidor gerenciador e um dispositivo gerenciado usando uma sessão orientada para conexão segura, como o protocolo TLS (do inglês *Transport Layer Security – Segurança na Camada de Transporte*) (discutido no Capítulo 8) em TCP.

A Figura 5.22 mostra um exemplo de sessão NETCONF. Primeiro, o servidor gerenciador estabelece uma conexão segura com o dispositivo gerenciado (no NETCONF, o servidor gerenciador é chamado de “cliente”, e o dispositivo gerenciado, de “servidor”, já que o servidor gerenciador estabelece a conexão com o dispositivo gerenciado; contudo, vamos ignorar esses termos para fins de consistência e dar preferência à terminologia tradicional de cliente/servidor do gerenciamento de redes mostrada na Figura 5.20). Após o estabelecimento de uma conexão segura, o servidor gerenciador e o dispositivo gerenciado trocam mensagens <hello>, declarando suas “capacidades” – a funcionalidade NETCONF que complementa a especificação NETCONF de base no RFC 6241. As interações entre o servidor gerenciador e o dispositivo gerenciado assumem a forma de uma chamada de procedimento remoto, usando as mensagens <rpc> e <rpc-response>. Essas mensagens são utilizadas para recuperar, definir, consultar e modificar configurações de dispositivos, estatísticas e dados

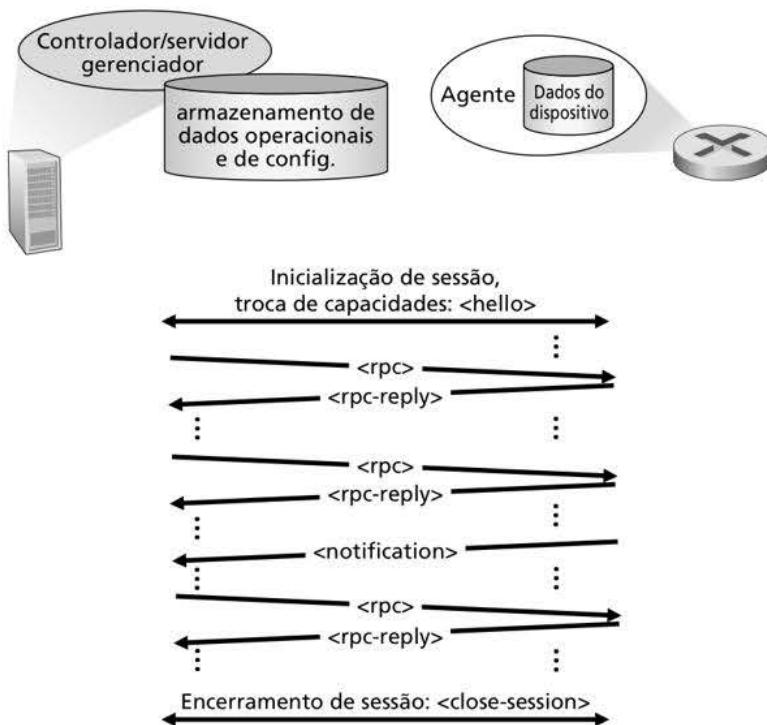


Figura 5.22 Sessão NETCONF entre controlador/servidor gerenciador e dispositivo gerenciado.

operacionais e para se inscrever para receber notificações do dispositivo. As notificações do dispositivo em si são enviadas proativamente do dispositivo gerenciado para o servidor gerenciador usando mensagens <notification> do NETCONF. Uma sessão é encerrada com a mensagem <close-session>.

A Tabela 5.3 mostra o número de operações NETCONF importantes que um servidor gerenciador pode realizar em um dispositivo gerenciado. Assim como no caso do SNMP, vemos operações para recuperar os dados de estado operacional (<get>) e para notificação de eventos. Contudo, as operações <get-config>, <edit-config>, <lock> e <unlock> demonstram a ênfase especial do NETCONF em configuração de dispositivos. Usando as operações básicas mostradas na Tabela 5.3, também é possível criar um *conjunto* de transações de gerenciamento de rede mais sofisticadas completadas atomicamente (i.e., em grupo) e de forma bem-sucedida em um *conjunto* de dispositivos, ou são completamente revertidas e deixam os dispositivos em seus estados pré-transação. Essas transações em múltiplos dispositivos, “permitindo que os operadores se concentrem na *configuração* da rede como um todo em vez de dispositivos individuais”, foi um requisito importante dos operadores estabelecido no RFC 3535.

Uma descrição completa do NETCONF estaria além do nosso escopo; os RFCs 6241 e 5277, Claise (2019) e Schonwalder (2010) oferecem uma análise mais aprofundada do tema.

Mas como esta é a primeira vez que vemos mensagens de protocolo no formato de um documento XML (e não na mensagem tradicional com campos de cabeçalho e corpo de mensagem, como mostrado na Figura 5.21 para o PDU SNMP), vamos concluir nosso breve exemplo do NETCONF com dois exemplos.

No primeiro exemplo, o documento XML enviado do servidor gerenciador para o dispositivo gerenciado é um comando NETCONF <get> requisitando todos os dados operacionais e de configuração do dispositivo. Com esse comando, o servidor pode descobrir a configuração do dispositivo.

```

01 <?xml version="1.0" encoding="UTF-8"?>
02 <rpc message-id="101"
03   xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
04     <get/>
05   </rpc>
```

TABELA 5.3 Operações NETCONF selecionadas

Operação NETCONF	Descrição
<get-config>	Recuperar toda ou parte de uma determinada configuração. Um dispositivo pode possuir múltiplas configurações. Há sempre uma configuração <i>running</i> / que descreve a configuração atual (em execução, ou <i>running</i>) do dispositivo.
<get>	Recuperar todos ou parte dos dados de estado de configuração e estado operacional.
<edit-config>	Muda toda ou parte de uma configuração especificada no dispositivo gerenciado. Se a configuração <i>running</i> / é especificada, a configuração atual (em execução) do dispositivo será alterada. Se o dispositivo gerenciado é capaz de atender à requisição, um <rpc-reply> é enviado, contendo um elemento <ok>; se não, o sistema retorna <rpccerror>. Em caso de erro, o estado de configuração do dispositivo pode ser retornado ao seu estado anterior.
<lock>, <unlock>	A operação <lock> (<unlock>) permite que o servidor gerenciador trave (ou destrave) todo o sistema de banco de dados de configuração de um dispositivo gerenciado. A ideia é que as travas sejam de curta duração e permitam que o cliente faça alterações sem medo de interações com outros comandos NETCONF, SNMP ou CLIs de outras fontes.
<create-subscription> , <notification>	Esta operação inicia uma assinatura de notificação de eventos que envia uma <notification> de evento assíncrono referente aos eventos especificados, do dispositivo gerenciado para o servidor gerenciador, até a assinatura ser encerrada.

Poucas pessoas são capazes de interpretar XML diretamente, mas vemos que o comando NETCONF é relativamente legível para um ser humano, lembrando muito mais o HTTP e o HTML do que os formatos de mensagem de protocolo que vimos para o formato de PDU SNMP na Figura 5.21. A mensagem de RPC em si abrange as linhas 02 a 05 (adicionamos os números de linha para fins pedagógicos). A RPC tem um valor de ID de mensagem de 101, declarado na linha 02, e contém um único comando NETCONF <get>. A resposta do dispositivo contém um número de ID correspondente (101) e todos os dados de configuração do dispositivo (em formato XML, é claro), iniciando na linha 04 e se encerrando com um </rpc-reply> final.

```
01 <?xml version="1.0" encoding="UTF-8"?>
02 <rpc-reply message-id="101"
03   xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
04     <!-- . . . todos os dados de configurações
05     retornados... --> . . .
06   </rpc-reply>
```

No segundo exemplo abaixo, adaptado do RFC 6241, o documento XML enviado do servidor gerenciador para o dispositivo gerenciado define a unidade máxima de transmissão (MTU, do inglês *maximum transmission unit*) de uma interface chamada “Ethernet0/0” como 1.500 bytes:

```
01 <?xml version="1.0" encoding="UTF-8"?>
02 <rpc message-id="101"
03   xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
04     <edit-config>
05       <target>
06         <running/>
07       </target>
08       <config>
09         <top xmlns="http://example.com/schema/
10           1.2/config">
11             <interface>
12               <name>Ethernet0/0</name>
13               <mtu>1500</mtu>
14             </interface>
15         </top>
16       </config>
17     </edit-config>
18   </rpc>
```

A mensagem RPC em si abrange as linhas 02 a 17, tem um valor de ID da mensagem de 101, e contém um único comando NETCONF <edit-config>, abrangendo as linhas 04 a 15. A linha 06 indica que a configuração em execução no dispositivo gerenciado será alterada. As linhas 11 e 12 especificam o tamanho da MTU a ser configurada para a interface de Ethernet0/0.

Após ter alterado o tamanho da MTU da interface na configuração, o dispositivo gerenciado responde para o servidor gerenciador com uma resposta OK (linha 04, abaixo), mais uma vez dentro de um documento XML:

```
01 <?xml version="1.0" encoding="UTF-8"?>
02 <rpc-reply message-id="101"
03   xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
04     <ok/>
05   </rpc-reply>
```

YANG

O YANG é uma linguagem de modelagem de dados usada para especificar precisamente a estrutura, sintaxe e semântica dos dados de gerenciamento de rede usados pelo NETCONF, da mesma forma como o SMI é usado para especificar MIBs no SNMP. Todas as definições YANG estão contidas em módulos, e um documento XML que descreve um dispositivo e suas capacidades pode ser gerado a partir de um módulo YANG.

O YANG inclui um pequeno conjunto de tipos de dados integrados (como no caso do SMI), e permite que modeladores de dados expressem restrições que devem ser atendidas por uma configuração NETCONF válida – um recurso muito útil para ajudar a garantir que as configurações NETCONF atendem a limitações específicas de precisão e consistência. O YANG também é usado para especificar notificações NETCONF.

Uma discussão mais completa sobre o YANG estaria além do nosso escopo. Para mais informações, o leitor interessado deve consultar o livro excelente de Claise (2019).

5.8 RESUMO

Agora completamos nossa viagem de dois capítulos ao mundo do núcleo da rede, que começou com o nosso estudo sobre o plano de dados da camada de rede no Capítulo 4 e termina aqui com o nosso estudo sobre o plano de controle. Aprendemos que o plano de controle é a lógica em nível de rede que controla como um datagrama é repassado entre os roteadores no caminho fim a fim, do hospedeiro de origem ao de destino, e como os serviços e componentes da camada de rede são configurados e gerenciados.

Aprendemos que há duas abordagens amplas para a construção de um plano de controle: o *controle por roteador tradicional* (no qual um algoritmo de roteamento roda em todos os roteadores e o componente de roteamento no roteador se comunica com os componentes de roteamento nos outros roteadores) e o *controle por rede definida por software* (SDN) (no qual um controlador logicamente centralizado calcula e distribui as tabelas de repasse que serão usadas por todos os roteadores). Estudamos dois algoritmos de roteamento fundamentais para o cálculo dos caminhos de menor custo em um grafo (roteamento de estado de enlace e roteamento por vetor de distâncias) na Seção 5.2. Os algoritmos têm aplicação tanto no controle por roteador quanto no controle SDN e servem de base para dois protocolos de roteamento da Internet amplamente utilizados, o OSPF e o BGP, que trabalhamos nas Seções 5.3 e 5.4. Analisamos a abordagem SDN ao plano de controle da camada de rede na Seção 5.5, investigando as aplicações de controle de rede SDN, o controlador SDN e o protocolo OpenFlow para a comunicação entre o controlador e os dispositivos controlados por SDN. Nas Seções 5.6 e 5.7, examinamos alguns dos elementos fundamentais do gerenciamento de uma rede IP: ICMP (o Protocolo de Mensagens de Controle da Internet) e gerenciamento de rede usando SNMP e NETCONF/YANG.

Agora que concluímos nosso estudo da camada de rede, nossa jornada segue rumo a um nível mais baixo da pilha de protocolos, isto é, à camada de enlace. Como a camada de rede, a camada de enlace é parte de todos os dispositivos conectados à rede, sem exceção. Mas veremos no próximo capítulo que a camada de enlace tem a tarefa muito mais localizada de movimentar pacotes entre nós no mesmo enlace ou rede local. Embora essa tarefa possa, à primeira vista, parecer simples quando comparada às tarefas da camada de rede, veremos que a camada de enlace envolve uma série de questões importantes e fascinantes que podem nos manter ocupados por muito tempo.

Exercícios de fixação e perguntas

Questões de revisão do Capítulo 5

SEÇÃO 5.1

- R1. O que significa dizer que um plano de controle se baseia em controle por roteador? Nesses casos, o que significa quando afirmamos que os planos de controle e de dados da rede são implementados “monoliticamente”?
- R2. O que significa dizer que um plano de controle se baseia em controle logicamente centralizado? Nesses casos, o plano de dados e o plano de controle são implementados no mesmo dispositivo ou em dispositivos separados? Explique.

SEÇÃO 5.2

- R3. Compare e aponte as diferenças entre as propriedades de um algoritmo de roteamento centralizado e um distribuído. Dê um exemplo de protocolo de roteamento que adote uma abordagem centralizada e de outro que adote uma abordagem descentralizada.
- R4. Compare e aponte as diferenças entre os algoritmos de roteamento de estado de enlace e por vetor de distâncias.
- R5. O que é o problema de “contagem ao infinito” no roteamento de vetor de distâncias?
- R6. É necessário que todo sistema autônomo use o mesmo algoritmo de roteamento intra-AS? Justifique sua resposta.

SEÇÕES 5.3–5.4

- R7. Por que são usados protocolos inter-AS e intra-AS diferentes na Internet?
- R8. Verdadeiro ou falso: quando uma rota OSPF envia a sua informação de estado de enlace, esta é enviada apenas aos nós vizinhos ligados diretamente. Explique a sua resposta.
- R9. O que significa falar da *área* em um sistema autônomo OSPF? Por que o conceito de área foi introduzido?
- R10. Defina e aponte as diferenças entre os seguintes termos: *sub-rede*, *prefixo* e *rota BGP*.
- R11. Como o BGP usa o atributo NEXT-HOP? Como ele usa o atributo AS-PATH?
- R12. Descreva como um administrador de rede de um ISP de nível superior pode executar uma política ao configurar o BGP.
- R13. Verdadeiro ou falso: quando um roteador BGP recebe um caminho anunciado do seu vizinho, este deve acrescentar a sua própria identidade ao caminho recebido e então enviar esse novo caminho para todos os seus vizinhos. Explique a sua resposta.

SEÇÃO 5.5

- R14. Descreva o papel principal da camada de comunicação, da camada de gerenciamento de estado em âmbito de rede e da camada de aplicação de controle de rede em um controlador SDN.
- R15. Suponha que você deseja implementar um novo protocolo de roteamento no plano de controle SDN. Em qual camada implementaria o protocolo? Explique a sua resposta.
- R16. Quais tipos de mensagens fluem através das APIs northbound e southbound de um controlador SDN? Qual é o destinatário dessas mensagens enviadas do controlador através da interface southbound, e quem envia as mensagens para o controlador através da interface northbound?

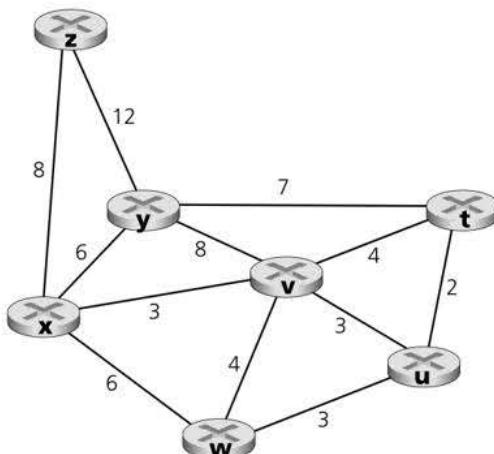
- R17. Descreva o propósito de dois tipos de mensagem OpenFlow (à sua escolha) enviadas de um dispositivo controlado para o controlador. Descreva o propósito de dois tipos de mensagem OpenFlow (à sua escolha) enviadas do controlador para um dispositivo controlado.
- R18. Qual é o propósito da camada de abstração de serviço no controlador SDN OpenDaylight?

SEÇÕES 5.6–5.7

- R19. Liste quatro tipos diferentes de mensagens ICMP.
- R20. Quais os dois tipos de mensagem ICMP recebidas no hospedeiro remetente que executa o programa *Traceroute*?
- R21. Defina os seguintes termos no contexto do SNMP: *servidor gerenciador*, *dispositivo gerenciado*, *agente de gerenciamento* e *MIB*.
- R22. Qual é o propósito das mensagens SNMP *GetRequest* e *SetRequest*?
- R23. Qual é o propósito da mensagem SNMP trap?

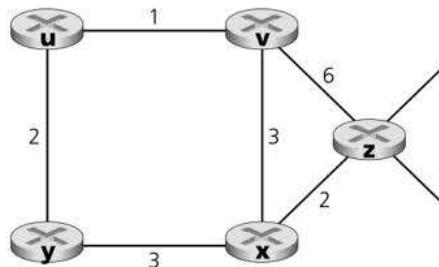
Problemas

- P1. Considerando a Figura 5.3, enumere os caminhos de *y* a *u* que não tenham laços.
- P2. Repita o Problema P1 considerando os caminhos de *x* a *z*, *z* a *u* e *z* a *w*.
- P3. Considere a seguinte rede. Com os custos de enlace indicados, use o algoritmo do caminho mais curto de Dijkstra para calcular o caminho mais curto de *x* até todos os nós da rede. Mostre como o algoritmo funciona calculando uma tabela semelhante à Tabela 5.1.

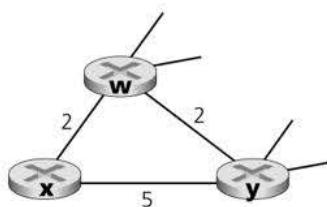


- P4. Considere a rede mostrada no Problema P3. Usando o algoritmo de Dijkstra e mostrando seu trabalho usando uma tabela semelhante à Tabela 5.1, faça o seguinte:
- Calcule o caminho mais curto de *t* até todos os nós da rede.
 - Calcule o caminho mais curto de *u* até todos os nós da rede.
 - Calcule o caminho mais curto de *v* até todos os nós da rede.
 - Calcule o caminho mais curto de *w* até todos os nós da rede.
 - Calcule o caminho mais curto de *y* até todos os nós da rede.
 - Calcule o caminho mais curto de *z* até todos os nós da rede.

- P5. Considere a rede mostrada a seguir e admita que cada nó inicialmente conheça os custos até cada um de seus vizinhos. Considere o algoritmo de vetor de distâncias e mostre os registros na tabela de distâncias para o nó z .



- P6. Considere uma topologia geral (i.e., não a rede específica mostrada antes) e uma versão síncrona do algoritmo de vetor de distâncias. Suponha que, a cada iteração, um nó troque seus vetores de distâncias com seus vizinhos e receba os vetores de distâncias deles. Supondo que o algoritmo comece com cada nó conhecendo apenas os custos até seus vizinhos imediatos, qual é o número máximo de iterações exigidas até que o algoritmo distribuído converja? Justifique sua resposta.
- P7. Considere o fragmento de rede mostrado a seguir. x tem apenas dois vizinhos ligados a ele: w e y . w tem um caminho de custo mínimo até o destino u (não mostrado) de 5, e y tem um caminho de custo mínimo u de 6. Os caminhos completos de w e de y até u (e entre w e y) não são mostrados. Todos os valores dos custos de enlace na rede são números inteiros estritamente positivos.



- Dê os vetores de distâncias de x para os destinos w , y e u .
 - Dê uma mudança de custo de enlace para $c(x, w)$ ou para $c(x, y)$ tal que x informará a seus vizinhos um novo caminho de custo mínimo até u como resultado da execução do algoritmo de vetor de distâncias.
 - Dê uma mudança de custo de enlace para $c(x, w)$ ou para $c(x, y)$ tal que x não informará a seus vizinhos um novo caminho de custo mínimo até u como resultado da execução do algoritmo de vetor de distâncias.
- P8. Considere a topologia de três nós mostrada na Figura 5.6. Em vez de ter os custos de enlace da Figura 5.6, os custos de enlace são: $c(x, y) = 3$, $c(y, z) = 6$, $c(z, x) = 4$. Calcule as tabelas de distâncias após a etapa de inicialização e após cada iteração de uma versão síncrona do algoritmo de vetor de distâncias (como fizemos em nossa discussão anterior da Figura 5.6).
- P9. Considere o problema da contagem até o infinito no roteamento de vetor de distâncias. Esse problema ocorrerá se reduzirmos o custo de um enlace? Por quê? E se conectarmos dois nós que não possuem enlace?
- P10. Demonstre que, para o algoritmo de vetor de distâncias na Figura 5.6, cada valor no vetor de distâncias $D(x)$ é não crescente e, consequentemente, se estabilizará em um número finito de etapas.
- P11. Considere a Figura 5.7. Suponha que haja outro roteador, w , conectado aos roteadores y e z . Os custos de todos os enlaces são: $c(x,y) = 4$, $c(x,z) = 50$, $c(y,w) = 1$, $c(z,w) = 1$,

$c(y,z) = 3$. Suponha que a reversão envenenada seja utilizada no algoritmo de roteamento de vetor de distâncias.

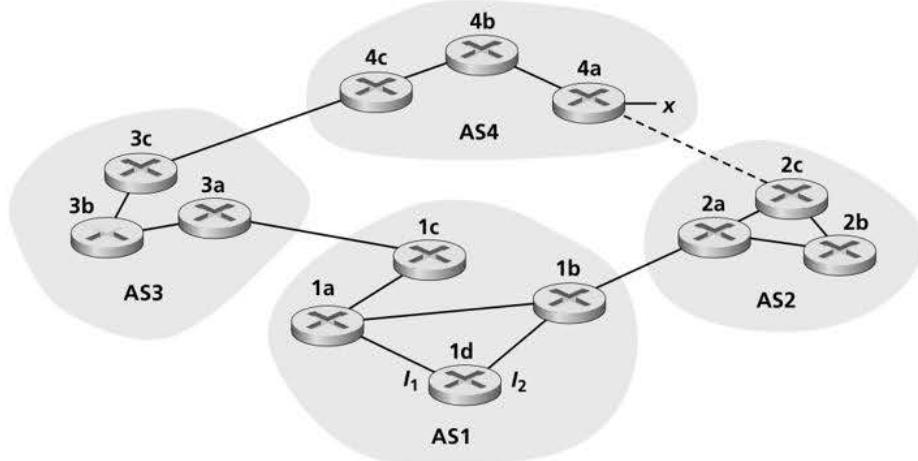
- Quando o roteamento de vetor de distâncias é estabilizado, os roteadores w , y e z informam uns aos outros sobre suas distâncias para x . Quais valores de distância eles podem informar uns aos outros?
- Agora suponha que o custo do enlace entre x e y aumente para 60. Haverá um problema de contagem até o infinito mesmo se a reversão envenenada for utilizada? Por quê? Se houver um problema de contagem até o infinito, então quantas iterações são necessárias para o roteamento de vetor de distâncias alcançar um estágio estável novamente? Justifique sua resposta.
- Como você modifica $c(y,z)$ de modo que não haja qualquer problema de contagem até o infinito se $c(y,x)$ mudar de 4 para 60?

P12. Descreva como laços nos caminhos podem ser detectados com BGP.

P13. Um roteador BGP sempre escolherá uma rota sem laços com o menor comprimento de AS-PATH? Justifique sua resposta.

P14. Considere a rede a seguir. Suponha que AS3 e AS2 estejam rodando o OSPF para seu protocolo de roteamento intra-AS. Suponha que AS1 e AS4 estejam rodando o RIP para seu protocolo de roteamento intra-AS. Suponha que o eBGP e o iBGP sejam usados para o protocolo de roteamento inter-AS. Inicialmente, suponha que não haja enlace físico entre AS2 e AS4.

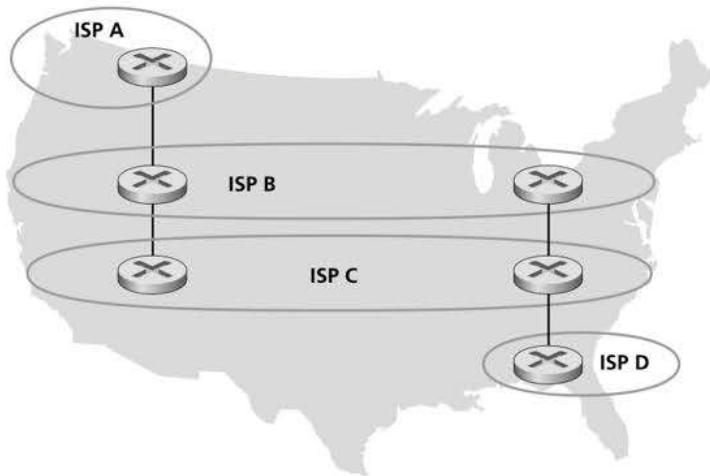
- O roteador 3c sabe sobre o prefixo x por qual protocolo de roteamento: OSPF, RIP, eBGP ou iBGP?
- O roteador 3a sabe sobre o prefixo x por qual protocolo de roteamento?
- O roteador 1c sabe sobre o prefixo x por qual protocolo de roteamento?
- O roteador 1d sabe sobre o prefixo x por qual protocolo de roteamento?



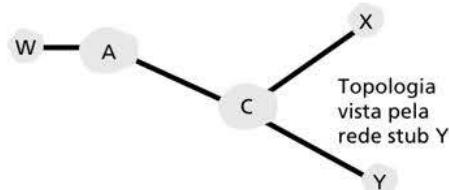
P15. Referindo-se ao problema anterior, uma vez que o roteador 1d sabe sobre x , ele inserirá uma entrada (x, I) em sua tabela de repasse.

- I será igual a I_1 ou I_2 para essa entrada? Justifique a resposta em uma frase.
- Agora suponha que haja um enlace físico entre AS2 e AS4, ilustrado pela linha pontilhada. Suponha que o roteador 1d saiba que x é acessível por meio de AS2 e de AS3. I será definido para I_1 ou I_2 ? Justifique a resposta em uma frase.
- Agora suponha que haja outro AS, denominado AS5, que fica no caminho entre AS2 e AS4 (não ilustrado no diagrama). Suponha que o roteador 1d saiba que x é acessível por meio de AS2, AS5 e AS4, bem como de AS3 e AS4. I será definido para I_1 ou I_2 ? Em uma frase, explique o motivo.

- P16. Considere a rede a seguir. O ISP B provê serviço nacional de backbone ao ISP regional A. O ISP C provê serviço nacional de backbone ao ISP regional D. Cada ISP consiste em um AS. Usando BGP, B e C se emparelham em dois lugares. Considere o tráfego na direção de A a D. B preferiria passar esse tráfego para C na Costa Oeste (de modo que C teria de absorver o custo de transportar o tráfego através do país), enquanto C preferiria receber o tráfego via seu ponto de emparelhamento com B na Costa Leste (de modo que B transportaria o tráfego através do país). Qual mecanismo BGP poderia ser usado por C de modo que B entregasse o tráfego de A a D em seu ponto de emparelhamento na Costa Leste? Para responder a essa pergunta, você precisará estudar muito bem a especificação do BGP.



- P17. Na Figura 5.13, considere a informação de caminho que chega às sub-redes* stub W, X e Y. Com base na informação disponível em W e X, quais são as respectivas visões da topologia da rede? Justifique sua resposta. A topologia vista de Y é mostrada a seguir.



- P18. Considere a Figura 5.13. B nunca encaminharia tráfego destinado a Y via X com base no roteamento BGP. Mas existem muitas aplicações conhecidas para as quais os pacotes de dados vão primeiro para X e depois fluem para Y. Identifique tal aplicação e descreva como os pacotes de dados percorrem um caminho não determinado pelo roteamento BGP.
- P19. Na Figura 5.13, suponha que haja outra rede stub V que seja cliente do ISP A. Suponha que B e C tenham uma relação de emparelhamento, e que A seja cliente de B e de C. Suponha, ainda, que A gostaria de ter o tráfego destinado para W vindo apenas de B, e o tráfego destinado para V vindo de B ou C. Como A deveria anunciar suas rotas para B e C? Quais rotas AS são recebidas por C?
- P20. Suponha que os ASs X e Z não estejam conectados diretamente, mas estejam conectados pelo AS Y. Suponha ainda que X tenha um acordo de emparelhamento com Y, e que Y tenha um acordo de emparelhamento com Z. Por fim, suponha que Z queira transitar todo o tráfego de Y, mas não queira transitar o tráfego de X. O BGP permite que Z execute essa política?

*N. de T.: Sub-rede “stub” é aquela que se liga em um único ponto ao restante da rede e, portanto, não pode ter tráfego de passagem. A palavra “stub” significa um ramo de uma árvore.

- P21. Considere as duas maneiras pelas quais ocorrem as comunicações entre uma entidade gerenciadora e um dispositivo gerenciado: modo comando-resposta e modo assíncrono com traps. Quais são os prós e os contras dessas duas técnicas, em termos de (1) sobrecarga, (2) tempo de notificação quando ocorrem eventos excepcionais, e (3) robustez quanto às mensagens perdidas entre a entidade gerenciadora e o dispositivo gerenciado?
- P22. Na Seção 5.7, vimos que era preferível transportar mensagens SNMP em datagramas UDP não confiáveis. Em sua opinião, por que os projetistas do SNMP preferiram o UDP ao TCP como protocolo de transporte para o SNMP?

Tarefas de programação de *sockets* 5: ICMP Ping

Ao final do Capítulo 2, existem quatro tarefas de programação de *sockets*. A seguir, você verá uma quinta tarefa que emprega ICMP, um protocolo discutido neste capítulo.

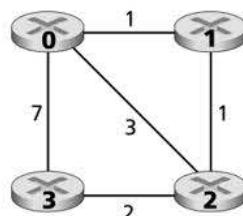
Ping é uma aplicação popular para redes, usada para testar, de um local remoto, se determinado hospedeiro está ativo e pode ser alcançado. Em geral, é usada para medir a latência entre o hospedeiro cliente e o hospedeiro de destino. Ele funciona enviando pacotes ICMP de “requisição de eco” (i.e., pacotes ping) ao hospedeiro de destino e escutando as “respostas de eco” ICMP (i.e., pacotes pong). Ping mede o RTT, registra perda de pacote e calcula um resumo estatístico de diversas trocas ping-pong (o mínimo, média, máximo e desvio-padrão dos tempos de viagem de ida e volta).

Neste laboratório, você escreverá sua própria aplicação Ping em Python. Sua aplicação usará ICMP. Mas, para simplificar seu programa, você não seguirá exatamente a especificação oficial no RFC 1739. Observe que só precisará escrever o lado cliente do programa, pois a funcionalidade necessária no lado servidor já está embutida em todos os sistemas operacionais. Você poderá achar todos os detalhes desta tarefa, bem como trechos importantes do código em Python, no site de apoio do livro.

Tarefas de programação: roteamento

Nesta tarefa de programação, você escreverá um conjunto “distribuído” de procedimentos que executam um roteamento de vetor de distâncias assíncrono distribuído para a rede mostrada a seguir.

Você deve escrever as seguintes rotinas que “rodarão” assincronamente dentro do ambiente emulado fornecido para essa tarefa. Para o nó 0, escreverá estas rotinas:



- *rtinit0()*. Essa rotina será chamada uma vez no início da emulação. *rtinit0()* não tem argumentos. Você deve inicializar sua tabela de distâncias no nó 0 para refletir os custos diretos de 1, 3 e 7 até os nós 1, 2 e 3, respectivamente. Na figura anterior, todos os enlaces são bidirecionais, e os custos em ambas as direções são idênticos. Após inicializar a tabela de distâncias e quaisquer outras estruturas de dados necessárias às rotinas de

seu nó 0, este deve então enviar a seus vizinhos diretamente ligados (nesse caso, 1, 2 e 3) o custo de seus caminhos de custo mínimo para todos os outros nós da rede. Essa informação do custo mínimo é enviada aos nós vizinhos em um pacote de atualização de roteamento chamando a rotina *tolayer2()*, conforme descrito na tarefa completa. O formato do pacote de atualização de roteamento também está descrito na tarefa completa.

- *rtupdate0(struct rtpkt *rcvdpkt)*. Essa rotina será chamada quando o nó 0 receber um pacote de roteamento que foi enviado a ele por um de seus vizinhos diretamente conectados. O parâmetro **rcvdpkt* é um ponteiro para o pacote que foi recebido. *rtupdate0()* é o “coração” do algoritmo de vetor de distâncias. Os valores que ele recebe em um pacote de atualização de roteamento de algum outro nó *i* contêm os custos correntes do caminho mais curto de *i* para todos os outros nós da rede. *rtupdate0()* usa esses valores recebidos para atualizar sua própria tabela de distâncias (conforme especificado pelo algoritmo de vetor de distâncias). Se seu próprio custo mínimo até outro nó mudar como resultado da atualização, o nó 0 informará essa mudança no custo mínimo a seus vizinhos diretamente conectados enviando-lhes um pacote de roteamento. Lembre-se de que no algoritmo de vetor de distâncias apenas os nós conectados diretamente trocarão pacotes de roteamento. Assim, os nós 1 e 2 vão se comunicar, mas os nós 1 e 3, não.

Rotinas semelhantes são definidas para os nós 1, 2 e 3. Assim, você escreverá oito procedimentos ao todo: *rtinit0()*, *rtinit1()*, *rtinit2()*, *rtinit3()*, *rtupdate0()*, *rtupdate1()*, *rtupdate2()* e *rtupdate3()*. Juntas, essas rotinas executarão um cálculo assíncrono, distribuído, das tabelas de distâncias para a topologia e os custos mostrados na figura apresentada anteriormente.

No site de apoio do livro você poderá encontrar todos os detalhes da tarefa de programação, bem como o código em C de que precisará para criar o ambiente simulado de hardware/software. Também está disponível uma versão da tarefa em Java.

Wireshark Lab: ICMP

No site deste livro, você encontrará uma tarefa de laboratório Wireshark que examina o uso do protocolo ICMP nos comandos ping e Traceroute.

ENTREVISTA

Jennifer Rexford

Jennifer Rexford é professora no departamento de Ciência da Computação da Princeton University. Sua pesquisa tem o objetivo geral de tornar as redes de computadores mais fáceis de projetar e administrar, com ênfase particular em redes programáveis. De 1996 a 2004, foi membro do departamento de Gerenciamento e Desempenho de Redes da AT&T Labs-Research. Enquanto estava na AT&T, projetou técnicas e ferramentas para medição de rede, engenharia de tráfego e configuração de roteadores, que foram implementadas na rede de backbone da AT&T. Jennifer é coautora do livro *Web Protocols and Practice: Networking Protocols, Caching, and Traffic Measurement*, publicado pela Addison-Wesley em maio de 2001. Foi presidente da ACM SIGCOMM de 2003 a 2007. Graduou-se como bacharel em engenharia elétrica pela Princeton University em 1991 e obteve doutorado em engenharia elétrica e ciência da computação pela Universidade de Michigan em 1996. Em 2004, Jennifer foi vencedora do Grace Murray Hopper Award da ACM como jovem profissional de destaque em computação, recebendo subsequentemente os prêmios ACM Athena Lecturer Award (2016), NCWIT Harrold and Notkin Research and Graduate Mentoring Award (2017), ACM SIGCOMM por contribuições na carreira (2018) e IEEE Internet Award (2019). É ACM Fellow (2008), IEEE Fellow (2018) e membro da National Academy of Engineering (2014).



Imagem cortesia de Jennifer Rexford

Por favor, descreva um ou dois dos projetos mais interessantes em que você já trabalhou durante sua carreira. Quais foram os maiores desafios?

Quando eu era pesquisadora na AT&T, um grupo nosso projetou uma nova forma de gerenciar o roteamento nas redes de backbone dos ISPs. Em geral, os operadores de rede configuram cada roteador individualmente, e esses roteadores executam protocolos distribuídos para calcular caminhos através da rede. Acreditamos que o gerenciamento de rede seria mais simples e mais flexível se os operadores da rede pudesse exercer controle direto sobre o modo como os roteadores repassam o tráfego com base em uma visão em nível de rede da topologia e do tráfego. A plataforma de controle de roteamento (RCP, do inglês *Routing Control Platform*) que projetamos e montamos poderia calcular as rotas para todo o backbone da AT&T em um único computador comercial, e poderia controlar roteadores legados sem modificação. Para mim, esse projeto foi interessante porque tivemos uma ideia provocadora, um sistema funcional e, por fim, uma execução real em uma rede operacional. Avançando alguns anos, as redes definidas por software (SDN) se tornaram uma tecnologia amplamente difundida, e protocolos (como o OpenFlow) e linguagens (como P4) padrões facilitaram muito o processo de dizer aos comutadores o que fazer.

Como você acha que as redes definidas por software deveriam evoluir no futuro?

Em uma forte ruptura com o passado, o software que controla os dispositivos de rede pode ser criado por muitos programadores diferentes, não apenas nas empresas que vendem equipamentos de rede. Mas ao contrário das aplicações que rodam em um servidor ou em um smartphone, as de SDN precisam trabalhar juntas para lidar com o mesmo tráfego. Os operadores de rede não querem fazer balanceamento de carga para parte do tráfego e rotear o resto; em vez disso, querem balanceamento de carga e roteamento juntos, no mesmo tráfego. As plataformas de SDN futuras deveriam oferecer boas abstrações de programação para a composição de múltiplas aplicações produzidas independentemente juntas. De forma mais ampla, boas abstrações de programações podem facilitar o processo de criar aplicações sem que precisemos nos preocupar com detalhes de baixo nível, como linhas em tabelas

de fluxo, contadores de tráfego, padrões de bits em cabeçalhos de pacotes e assim por diante. Além disso, enquanto um controlador SDN é logicamente centralizado, a rede ainda é composta por um conjunto distribuído de dispositivos. Redes programáveis futuras deveriam oferecer boas abstrações para a atualização de um conjunto distribuído de dispositivos para que os administradores possam entender e raciocinar sobre o que acontece com os pacotes em trânsito enquanto os dispositivos são atualizados. A programação de abstrações para redes programáveis é uma área muito interessante para as pesquisas interdisciplinares entre redes de computadores, sistemas distribuídos e linguagens de programação, com uma chance real de terem um impacto prático nos próximos anos.

Qual futuro você vê para as redes e a Internet?

As redes compõem um campo muito interessante, pois as aplicações e as tecnologias utilizadas mudam o tempo todo. Estamos sempre reinventando! Quem teria previsto, há apenas dez anos, o domínio dos smartphones, permitindo que usuários móveis acessem aplicações existentes e novos serviços baseados em sua localização? O surgimento da computação em nuvem está fundamentalmente mudando a relação entre os usuários e as aplicações que eles executam, e os sensores e atuadores em rede (a “Internet das Coisas”) estão permitindo diversas aplicações novas (e novas vulnerabilidades de segurança!). O ritmo da inovação é mesmo inspirador.

A rede de apoio é um componente decisivo em todas essas inovações. Mesmo assim, a rede está notoriamente “no caminho” – limitando o desempenho, comprometendo a confiabilidade, restringindo aplicações e complicando a implementação e o gerenciamento de serviços. Devemos lutar para tornar a rede do futuro tão invisível quanto o ar que respiramos, de modo que nunca fique no caminho de novas ideias e serviços valiosos. Para isso, precisamos elevar o nível de abstração acima dos dispositivos de rede e protocolos individuais (e seus respectivos acrônimos!), de modo que possamos raciocinar sobre a rede e sobre os objetivos de alto nível do usuário como um todo.

Quais pessoas o inspiraram profissionalmente?

Há muito tempo tenho me inspirado em Sally Floyd, do International Computer Science Institute. Sua pesquisa foi sempre significativa, focalizando os desafios importantes enfrentados pela Internet. Ela mergulhava a fundo em questões difíceis, até que entendia completamente o problema e o espaço das soluções, e dedicava muita energia para “fazer as coisas acontecerem”, como empurrar suas ideias para padrões de protocolos e equipamentos de rede. Além disso, ela oferecia retorno à comunidade, através de serviços profissionais em diversas organizações de padrões e pesquisa, também criando ferramentas (como os simuladores ns-2 e ns-3 bastante utilizados) que permitem que outros pesquisadores tenham sucesso. Ela se aposentou em 2009 e faleceu em 2019, mas sua influência no campo será sentida durante anos.

Quais são suas recomendações para estudantes que desejam seguir carreira em computação e tecnologia da informação?

Redes é um campo inherentemente interdisciplinar. As técnicas aplicadas às redes são oriundas dos avanços de diversas disciplinas, tais como teoria de filas, teorias de jogos, teoria de controle, sistemas distribuídos, otimização de redes, linguagens de programação, aprendizado de máquina, algoritmos, estruturas de dados e assim por diante. Creio que familiarizar-se com um campo relacionado, ou colaborar de perto com especialistas nessas áreas, seja um modo maravilhoso de preparar um alicerce mais forte para as redes, de modo que possamos aprender como montar redes que sejam dignas de confiança da sociedade. Além das disciplinas teóricas, o campo das redes é interessante, porque criamos artefatos reais, que pessoas reais utilizam. Dominar o modo como projetamos e montamos sistemas – ganhando experiência em sistemas operacionais, arquitetura de computador etc. – é outro modo fantástico de ampliar seus conhecimentos em redes, ajudando a mudar o mundo.

Esta página foi deixada em branco intencionalmente.