Capítulo 3: Interconexão de redes

Problema: Nem todas as redes estão conectadas diretamente

Como vimos, existem muitas tecnologias que podem ser usadas para construir links de última milha ou conectar um número modesto de nós, mas como construímos redes em escala global? Uma única Ethernet pode interconectar no máximo 1.024 hosts; um link ponto a ponto conecta apenas dois. Redes sem fio são limitadas pelo alcance de seus rádios. Para construir uma rede global, precisamos de uma maneira de interconectar esses diferentes tipos de links e redes de multiacesso. O conceito de interconectar diferentes tipos de redes para construir uma grande rede global é a ideia central da Internet e é frequentemente chamado de *interconexão de redes* .

Podemos dividir o problema de interconexão de redes em alguns subproblemas. Primeiramente, precisamos de uma maneira de interconectar links. Dispositivos que interconectam links do mesmo tipo são frequentemente chamados de *switches* ou, às vezes, switches *da Camada 2* (L2). Esses dispositivos são o primeiro tópico deste capítulo. Uma classe particularmente importante de switches L2 em uso hoje são aqueles usados para interconectar segmentos Ethernet. Esses switches também são, às vezes, chamados de *pontes* .

A principal função de um switch é receber pacotes que chegam em uma entrada e *encaminhá-los* (ou *comutá-los*) para a saída correta, para que cheguem ao seu destino apropriado. Há diversas maneiras pelas quais o switch pode determinar a saída "correta" para um pacote, que podem ser amplamente categorizadas como abordagens sem conexão e orientadas à conexão. Essas duas abordagens encontraram importantes áreas de aplicação ao longo dos anos.

Dada a enorme diversidade de tipos de rede, também precisamos de uma maneira de interconectar redes e links díspares (ou seja, lidar com *a heterogeneidade*). Os dispositivos que realizam essa tarefa, antes chamados de *gateways* , agora são conhecidos principalmente como *roteadores* ou, alternativamente, switches *de Camada 3* (L3). O protocolo inventado para lidar com a interconexão de tipos de rede díspares, o Protocolo de Internet (IP), é o tema da nossa segunda seção.

Ao interconectarmos uma grande quantidade de links e redes com switches e roteadores, é provável que existam muitas maneiras diferentes de ir de um ponto a outro. Encontrar um caminho ou *rota* adequada em uma rede é um dos problemas fundamentais das redes. Esses caminhos devem ser eficientes (por exemplo, não mais longos do que o necessário), livres de loops e capazes de responder ao fato de que as redes não são estáticas — nós podem falhar ou reinicializar, links podem quebrar e novos nós ou links podem ser adicionados. Nossa terceira seção analisa alguns dos algoritmos e protocolos que foram desenvolvidos para abordar essas questões.

Uma vez compreendidos os problemas de comutação e roteamento, precisamos de alguns dispositivos para executar essas funções. Este capítulo conclui com uma discussão sobre as maneiras como switches e roteadores são implementados. Embora muitos switches e roteadores de pacotes sejam bastante semelhantes a um computador de uso geral, existem muitas situações em que projetos mais especializados são utilizados. Isso é particularmente verdadeiro no segmento de ponta, onde parece haver uma necessidade constante de maior capacidade de comutação que possa lidar com a carga de tráfego cada vez maior no núcleo da internet.

3.1 Noções básicas de comutação

Em termos mais simples, um switch é um mecanismo que nos permite interconectar links para formar uma rede maior. Um switch é um dispositivo com múltiplas entradas e

múltiplas saídas que transfere pacotes de uma entrada para uma ou mais saídas. Assim, um switch adiciona a topologia em estrela (veja a Figura 56) ao conjunto de estruturas de rede possíveis. Uma topologia em estrela possui várias propriedades atraentes:

- Embora um switch tenha um número fixo de entradas e saídas, o que limita o número de hosts que podem ser conectados a um único switch, grandes redes podem ser construídas interconectando vários switches.
- Podemos conectar switches entre si e a hosts usando links ponto a ponto, o que normalmente significa que podemos construir redes de grande escopo geográfico.
- Adicionar um novo host à rede conectando-o a um switch não reduz necessariamente o desempenho da rede para outros hosts já conectados.

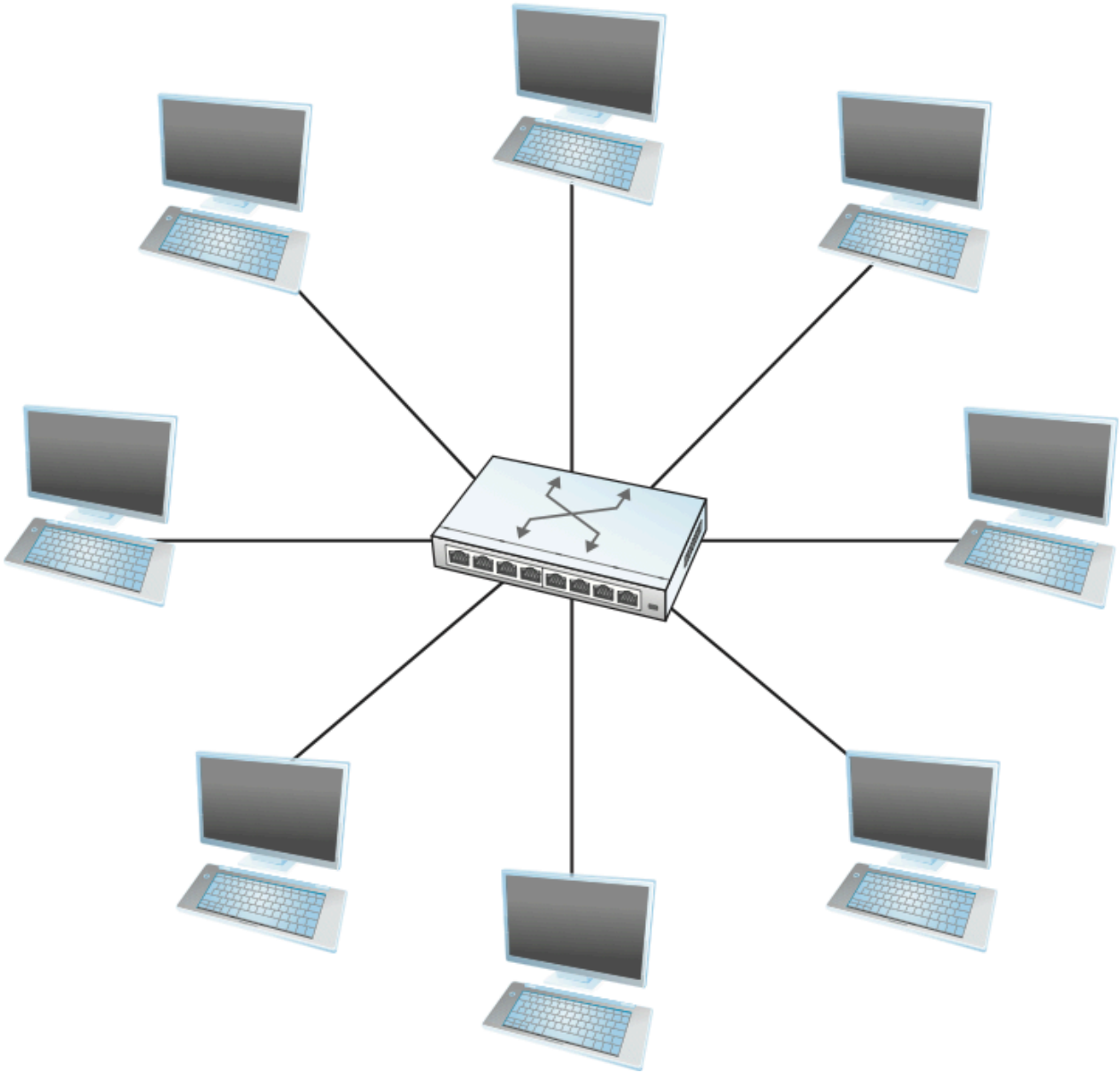


Figura 56. *Um switch fornece uma topologia em estrela.*

Esta última afirmação não pode ser feita para as redes de mídia compartilhada discutidas no último capítulo. Por exemplo, é impossível para dois hosts no mesmo segmento Ethernet de 10 Mbps transmitirem continuamente a 10 Mbps porque compartilham o mesmo meio de transmissão. Cada host em uma rede comutada tem seu próprio link para o switch, portanto, pode ser perfeitamente possível para muitos hosts transmitirem na velocidade máxima do link (largura de banda), desde que o switch seja projetado com capacidade agregada suficiente. Fornecer alta taxa de

transferência agregada é um dos objetivos do projeto de um switch; retornaremos a este tópico mais tarde. Em geral, as redes comutadas são consideradas mais *escaláveis* (ou seja, mais capazes de crescer para um grande número de nós) do que as redes de mídia compartilhada devido a essa capacidade de suportar muitos hosts em velocidade máxima.

Multiplexação por Divisão de Comprimento de Onda Densa

> Nosso foco em redes comutadas por pacotes obscurece o fato de que, especialmente em redes de longa distância, o transporte físico subjacente é totalmente óptico: não há pacotes. Nessa camada, equipamentos DWDM (*Multiplexação por Divisão de Comprimento de Onda Densa*) disponíveis comercialmente são capazes de transmitir um grande número de comprimentos de onda ópticos (cores) por uma única fibra. Por exemplo, pode-se enviar dados em 100 ou mais comprimentos de onda diferentes, e cada comprimento de onda pode transportar até 100 Gbps de dados.
>
> Conectando essas fibras está um dispositivo óptico chamado ROADM (*Reconfigurable Optical Add/Drop Multiplexers*). Uma coleção de ROADMs (nós) e fibras (links) formam uma rede de transporte óptico, onde cada ROADM é capaz de encaminhar comprimentos de onda individuais ao longo de um caminho multi-hop, criando um circuito lógico de ponta a ponta. Da perspectiva de uma rede de comutação de pacotes que pode ser construída sobre esse transporte óptico, um comprimento de onda, mesmo que cruze vários ROADMs, parece ser um único link ponto a ponto entre dois switches, sobre o qual se pode optar por executar SONET ou Ethernet de 100 Gbps como o protocolo de enquadramento. O recurso de reconfigurabilidade dos ROADMs significa que é possível alterar esses comprimentos de onda ponta a ponta subjacentes, criando efetivamente uma nova topologia na camada de comutação de pacotes.

Um switch é conectado a um conjunto de enlaces e, para cada um deles, executa o protocolo de enlace de dados apropriado para se comunicar com o nó na outra

extremidade do enlace. A principal função de um switch é receber pacotes de entrada em um de seus enlaces e transmiti-los em algum outro enlace. Essa função às vezes é chamada de *comutação* ou *encaminhamento* e, em termos da arquitetura OSI (Open Systems Interconnection), é considerada uma função da camada de rede. (Este é um caso em que a estrutura em camadas OSI não reflete perfeitamente o mundo real, como veremos mais adiante.)

A questão, então, é como o switch decide em qual link de saída colocar cada pacote? A resposta geral é que ele procura no cabeçalho do pacote um identificador que usa para tomar a decisão. Os detalhes de como ele usa esse identificador variam, mas existem duas abordagens comuns. A primeira é a abordagem *do datagrama* ou *sem conexão . A segunda é a abordagem do circuito virtual* ou *orientada à conexão* . Uma terceira abordagem, *o roteamento de origem* , é menos comum que as outras duas, mas tem algumas aplicações úteis.

Um aspecto comum a todas as redes é a necessidade de uma maneira de identificar os nós finais. Esses identificadores geralmente são chamados de *endereços* . Já vimos exemplos de endereços, como o endereço de 48 bits usado para Ethernet. O único requisito para endereços Ethernet é que nenhum dos dois nós em uma rede tenha o mesmo endereço. Isso é feito garantindo que todas as placas Ethernet recebam um identificador *globalmente exclusivo* . Para a discussão a seguir, presumimos que cada host tenha um endereço globalmente exclusivo. Posteriormente, consideraremos outras propriedades úteis que um endereço pode ter, mas a exclusividade global é suficiente para começar.

Outra suposição que precisamos fazer é que existe uma maneira de identificar as portas de entrada e saída de cada switch. Há pelo menos duas maneiras sensatas de identificar portas: uma é numerar cada porta e a outra é identificar a porta pelo nome

do nó (switch ou host) ao qual ela se conecta. Por enquanto, usaremos a numeração das portas.

3.1.1 Datagramas

A ideia por trás dos datagramas é incrivelmente simples: você simplesmente inclui em cada pacote informações suficientes para permitir que qualquer switch decida como levá-lo ao seu destino. Ou seja, cada pacote contém o endereço de destino completo. Considere a rede de exemplo ilustrada na Figura 57 , na qual os hosts têm os endereços A, B, C e assim por diante. Para decidir como encaminhar um pacote, um switch consulta uma *tabela de encaminhamento* (às vezes chamada de *tabela de roteamento*), um exemplo da qual é retratado na Tabela 5. Esta tabela em particular mostra as informações de encaminhamento que o switch 2 precisa para encaminhar datagramas na rede de exemplo. É muito fácil descobrir tal tabela quando você tem um mapa completo de uma rede simples como a retratada aqui; poderíamos imaginar um operador de rede configurando as tabelas estaticamente. É muito mais difícil criar as tabelas de encaminhamento em redes grandes e complexas com topologias que mudam dinamicamente e múltiplos caminhos entre destinos. Esse problema mais difícil é conhecido como *roteamento* e é o tópico de uma seção posterior. Podemos pensar no roteamento como um processo que ocorre em segundo plano para que, quando um pacote de dados aparecer, tenhamos as informações corretas na tabela de encaminhamento para poder encaminhar ou comutar o pacote.

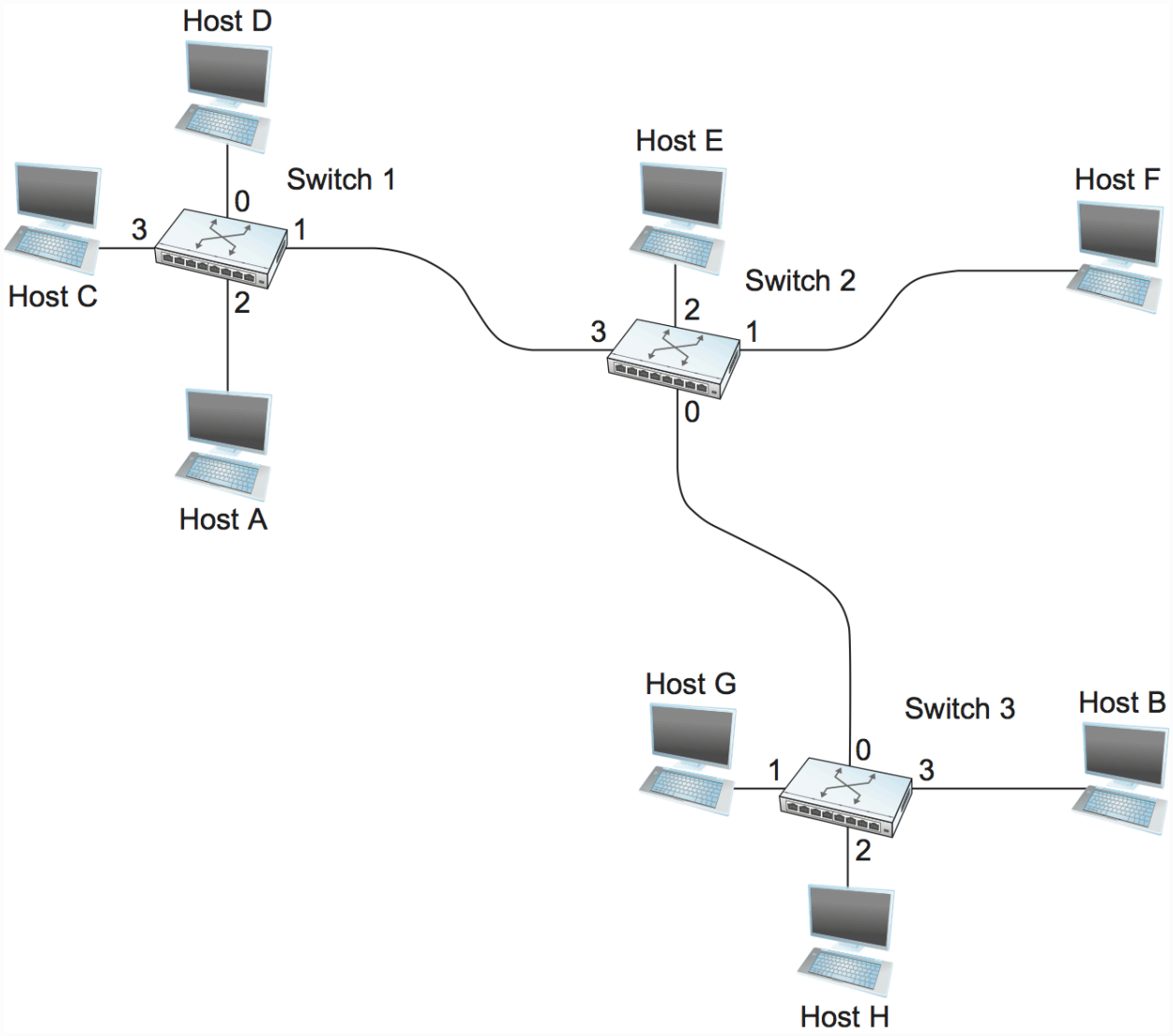


Figura 57. *Encaminhamento de datagramas: um exemplo de rede.*

Destino	Porta
UM	3
B	0

C	3
D	3
E	2
F	1
G	0
H	0

As redes de datagramas têm as seguintes características:

- Um host pode enviar um pacote para qualquer lugar e a qualquer momento, já que qualquer pacote que chegue a um switch pode ser encaminhado imediatamente (assumindo uma tabela de encaminhamento corretamente preenchida). Por esse motivo, as redes de datagramas são frequentemente chamadas de *redes sem conexão* ; isso contrasta com as redes *orientadas à conexão* descritas abaixo, nas quais algum *estado de conexão* precisa ser estabelecido antes do envio do primeiro pacote de dados.
- Quando um host envia um pacote, ele não tem como saber se a rede é capaz de entregá-lo ou se o host de destino está ativo e funcionando.
- Cada pacote é encaminhado independentemente de pacotes anteriores que possam ter sido enviados para o mesmo destino. Assim, dois pacotes

sucessivos do host A para o host B podem seguir caminhos completamente diferentes (talvez devido a uma alteração na tabela de encaminhamento em algum switch da rede).

- Uma falha de switch ou link pode não ter nenhum efeito sério na comunicação se for possível encontrar uma rota alternativa para contornar a falha e atualizar a tabela de encaminhamento adequadamente.

Este último fato é particularmente importante para a história das redes de datagramas. Um dos objetivos importantes do projeto da internet é a robustez a falhas, e a história tem demonstrado sua eficácia em atingir esse objetivo. Como as redes baseadas em datagramas são a tecnologia dominante discutida neste livro, adiamos exemplos ilustrativos para as seções seguintes e passamos para as duas principais alternativas.

3.1.2 Comutação de circuitos virtuais

Uma segunda técnica para comutação de pacotes utiliza o conceito de *circuito virtual* (CV). Essa abordagem, também conhecida como *modelo orientado à conexão* , requer a configuração de uma conexão virtual do host de origem para o host de destino antes do envio de quaisquer dados. Para entender como isso funciona, considere a Figura 58 , onde o host A deseja novamente enviar pacotes para o host B. Podemos pensar nisso como um processo de duas etapas. A primeira etapa é a "configuração da conexão". A segunda é a transferência de dados. Consideraremos cada etapa separadamente.

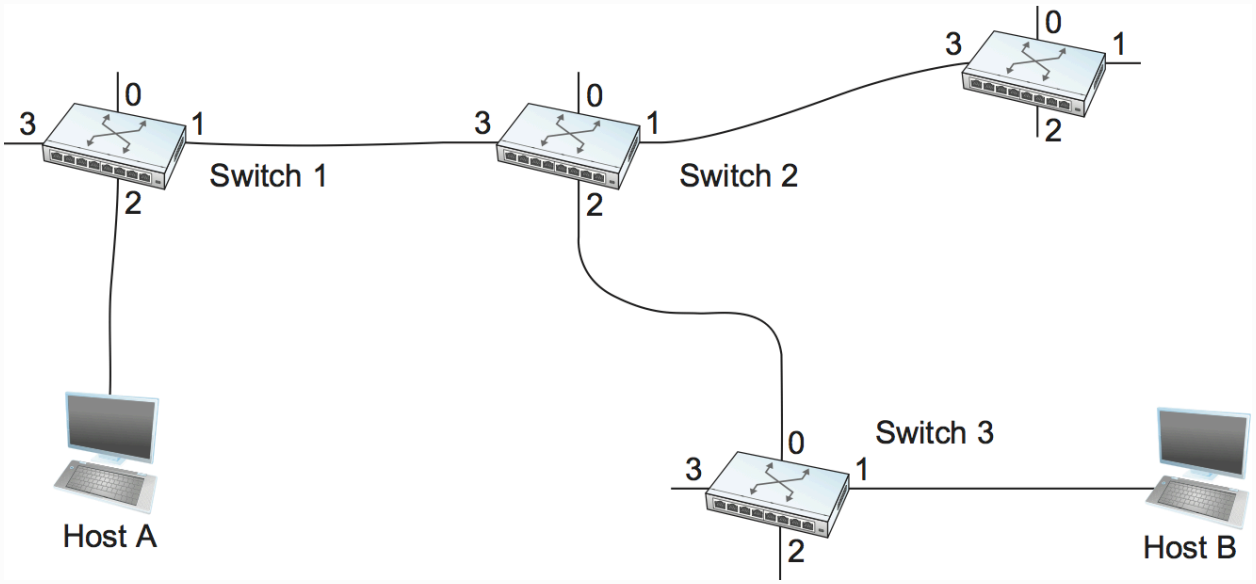


Figura 58. *Um exemplo de uma rede de circuitos virtuais.*

Na fase de configuração da conexão, é necessário estabelecer um "estado de conexão" em cada um dos switches entre os hosts de origem e destino. O estado de conexão para uma única conexão consiste em uma entrada em uma "tabela VC" em cada switch por onde a conexão passa. Uma entrada na tabela VC em um único switch contém:

- Um *identificador de circuito virtual* (VCI) que identifica exclusivamente a conexão neste switch e que será transportado dentro do cabeçalho dos pacotes que pertencem a esta conexão
- Uma interface de entrada na qual os pacotes para este VC chegam ao switch
- Uma interface de saída na qual os pacotes para este VC saem do switch
- Um VCI potencialmente diferente que será usado para pacotes de saída

A semântica de uma dessas entradas é a seguinte: se um pacote chega na interface de entrada designada e esse pacote contém o valor VCI designado em seu cabeçalho, então esse pacote deve ser enviado pela interface de saída especificada com o valor VCI de saída especificado tendo sido colocado primeiro em seu cabeçalho.

Observe que a combinação do VCI dos pacotes recebidos no switch *e* a interface na qual são recebidos identificam exclusivamente a conexão virtual. É claro que pode haver muitas conexões virtuais estabelecidas no switch simultaneamente. Além disso, observamos que os valores de VCI de entrada e saída geralmente não são os mesmos. Portanto, o VCI não é um identificador globalmente significativo para a conexão; em vez disso, ele tem significado apenas em um determinado link (ou seja, tem *escopo link-local*).

Sempre que uma nova conexão é criada, precisamos atribuir um novo VCI para essa conexão em cada link que ela percorrer. Também precisamos garantir que o VCI escolhido em um determinado link não esteja em uso naquele link por alguma conexão existente.

Existem duas abordagens amplas para estabelecer o estado de conexão. Uma é ter um administrador de rede configurando o estado, nesse caso o circuito virtual é "permanente". Claro, ele também pode ser excluído pelo administrador, então um circuito virtual permanente (PVC) pode ser melhor pensado como um VC de longa duração ou configurado administrativamente. Alternativamente, um host pode enviar mensagens para a rede para fazer com que o estado seja estabelecido. Isso é chamado de *sinalização* , e os circuitos virtuais resultantes são chamados de *comutados* . A característica saliente de um circuito virtual comutado (SVC) é que um host pode configurar e excluir tal VC dinamicamente sem o envolvimento de um administrador de rede. Observe que um SVC deveria ser mais precisamente chamado de VC *sinalizado* , uma vez que é o uso de sinalização (não comutação) que distingue um SVC de um PVC.

Vamos supor que um administrador de rede queira criar manualmente uma nova conexão virtual do host A para o host B. Primeiro, o administrador precisa identificar um caminho pela rede de A para B. Na rede de exemplo da Figura 58 , há apenas um caminho desse tipo, mas, em geral, esse pode não ser o caso. O administrador então escolhe um valor de VCI que não está sendo usado atualmente em cada link para a conexão. Para os propósitos do nosso exemplo, vamos supor que o valor de VCI 5 seja

escolhido para o link do host A para o switch 1, e que 11 seja escolhido para o link do switch 1 para o switch 2. Nesse caso, o switch 1 precisa ter uma entrada em sua tabela VC configurada conforme mostrado na Tabela 6 .

Interface de entrada	VCI de entrada	Interface de saída	VCI de saída
2	5	1	11

Da mesma forma, suponha que o VCI de 7 seja escolhido para identificar esta conexão no link do switch 2 para o switch 3 e que um VCI de 4 seja escolhido para o link do switch 3 para o host B. Nesse caso, os switches 2 e 3 precisam ser configurados com entradas na tabela VC, conforme mostrado na Tabela 7 e na Tabela 8 , respectivamente. Observe que o valor do VCI de "saída" em um switch é o valor do VCI de "entrada" no switch seguinte.

Interface de entrada	VCI de entrada	Interface de saída	VCI de saída
3	11	2	7

Interface de entrada	VCI de entrada	Interface de saída	VCI de saída

0	7	1	4

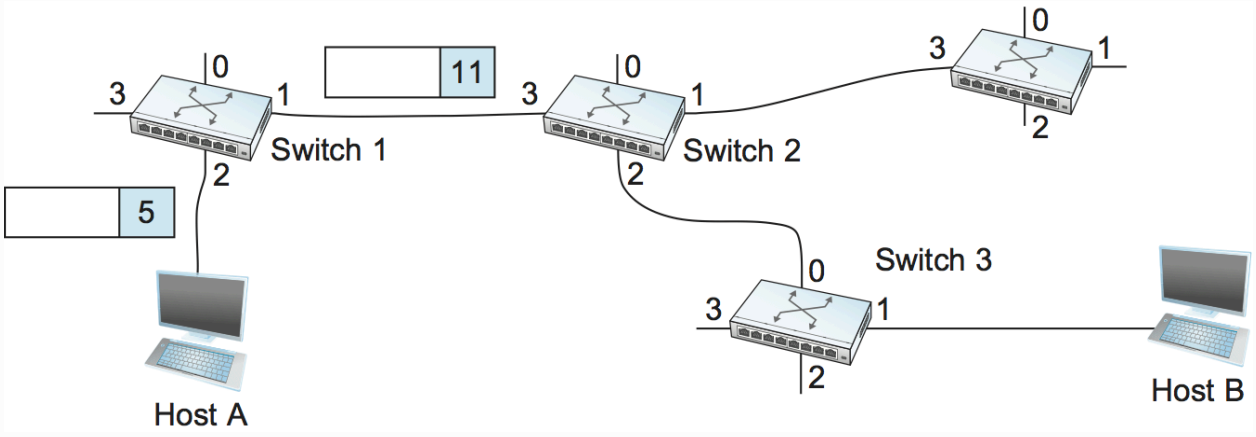


Figura 59. *Um pacote é enviado para uma rede de circuitos virtuais.*

Uma vez que as tabelas VC foram configuradas, a fase de transferência de dados pode prosseguir, conforme ilustrado na Figura 59. Para qualquer pacote que ele queira enviar ao host B, A coloca o valor VCI de 5 no cabeçalho do pacote e o envia ao switch 1. O switch 1 recebe qualquer pacote desse tipo na interface 2 e usa a combinação da interface e do VCI no cabeçalho do pacote para encontrar a entrada apropriada da tabela VC. Conforme mostrado na Tabela 6 , a entrada da tabela neste caso diz ao switch 1 para encaminhar o pacote para fora da interface 1 e colocar o valor VCI 11 no cabeçalho quando o pacote for enviado. Assim, o pacote chegará ao switch 2 na interface 3 com o VCI 11. O switch 2 procura a interface 3 e o VCI 11 em sua tabela VC (conforme mostrado na Tabela 7) e envia o pacote para o switch 3 após atualizar o valor VCI no cabeçalho do pacote adequadamente, conforme mostrado na Figura 60 . Esse processo continua até chegar ao host B com o valor VCI de 4 no pacote. Para o host B, isso identifica o pacote como vindo do host A.

Em redes reais de tamanho razoável, a carga de configurar tabelas de VC corretamente em um grande número de switches se tornaria rapidamente excessiva usando os procedimentos acima. Assim, uma ferramenta de gerenciamento de rede ou algum tipo de sinalização (ou ambos) é quase sempre utilizada, mesmo ao configurar VCs "permanentes". No caso de PVCs, a sinalização é iniciada pelo administrador da

rede, enquanto os SVCs geralmente são configurados usando a sinalização de um dos hosts. Consideraremos agora como o mesmo VC descrito acima poderia ser configurado por meio da sinalização do host.

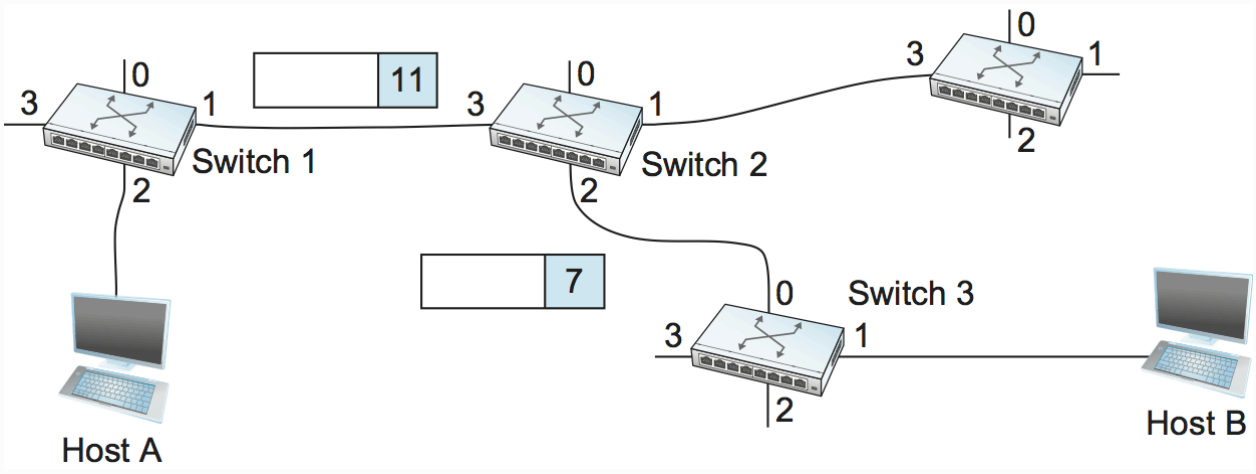


Figura 60. *Um pacote percorre uma rede de circuitos virtuais.*

Para iniciar o processo de sinalização, o host A envia uma mensagem de configuração para a rede, ou seja, para o switch 1. A mensagem de configuração contém, entre outras coisas, o endereço de destino completo do host B. A mensagem de configuração precisa chegar até B para criar o estado de conexão necessário em cada switch ao longo do caminho. Podemos ver que enviar a mensagem de configuração para B é muito parecido com enviar um datagrama para B, no sentido de que os switches precisam saber para qual saída enviar a mensagem de configuração para que ela finalmente chegue a B. Por enquanto, vamos supor que os switches conheçam a topologia da rede o suficiente para descobrir como fazer isso, de modo que a mensagem de configuração flua para os switches 2 e 3 antes de finalmente chegar ao host B.

Quando o switch 1 recebe a solicitação de conexão, além de enviá-la ao switch 2, ele cria uma nova entrada em sua tabela de circuitos virtuais para essa nova conexão. Essa entrada é exatamente a mesma mostrada anteriormente na Tabela 6. A principal diferença é que agora a tarefa de atribuir um valor VCI não utilizado na interface é realizada pelo switch para essa porta. Neste exemplo, o switch escolhe o valor 5. A

tabela de circuitos virtuais agora tem a seguinte informação: "Quando os pacotes chegarem na porta 2 com o identificador 5, envie-os pela porta 1". Outro problema é que, de alguma forma, o host A precisará aprender que deve colocar o valor VCI 5 nos pacotes que deseja enviar para B; veremos como isso acontece a seguir.

Quando o switch 2 recebe a mensagem de configuração, ele executa um processo semelhante; neste exemplo, ele escolhe o valor 11 como o valor do VCI de entrada. Da mesma forma, o switch 3 escolhe 7 como o valor para seu VCI de entrada. Cada switch pode escolher qualquer número que desejar, desde que esse número não esteja em uso para alguma outra conexão naquela porta do switch. Como observado acima, os VCIs têm escopo link-local; ou seja, não têm significado global.

Por fim, a mensagem de configuração chega ao host B. Supondo que B esteja saudável e disposto a aceitar uma conexão do host A, ele também aloca um valor VCI de entrada, neste caso 4. Esse valor VCI pode ser usado por B para identificar todos os pacotes provenientes do host A.

Agora, para completar a conexão, todos precisam ser informados sobre qual VCI seu vizinho a jusante está usando para esta conexão. O Host B envia uma confirmação da configuração da conexão para o switch 3 e inclui nessa mensagem o VCI que escolheu (4). Agora, o switch 3 pode completar a entrada da tabela de circuitos virtuais para esta conexão, pois sabe que o valor de saída deve ser 4. O Switch 3 envia a confirmação para o Switch 2, especificando um VCI de 7. O Switch 2 envia a mensagem para o Switch 1, especificando um VCI de 11. Finalmente, o Switch 1 passa a confirmação para o Host A, instruindo-o a usar o VCI de 5 para esta conexão.

Neste ponto, todos sabem tudo o que é necessário para permitir o fluxo de tráfego do host A para o host B. Cada switch possui uma entrada completa na tabela de circuitos virtuais para a conexão. Além disso, o host A tem um reconhecimento firme de que tudo está pronto até o host B. Neste ponto, as entradas na tabela de conexão estão prontas em todos os três switches, assim como no exemplo configurado administrativamente acima, mas todo o processo ocorreu automaticamente em

resposta à mensagem de sinalização enviada por A. A fase de transferência de dados pode agora começar e é idêntica à usada no caso do PVC.

Quando o host A não deseja mais enviar dados para o host B, ele interrompe a conexão enviando uma mensagem de interrupção ao switch 1. O switch remove a entrada relevante de sua tabela e encaminha a mensagem para os outros switches no caminho, que, da mesma forma, excluem as entradas apropriadas da tabela. Nesse ponto, se o host A enviasse um pacote com VCI 5 para o switch 1, ele seria descartado como se a conexão nunca tivesse existido.

Há várias coisas a serem observadas sobre a comutação de circuitos virtuais:

- Como o host A precisa esperar a solicitação de conexão chegar ao outro lado da rede e retornar antes de poder enviar seu primeiro pacote de dados, há pelo menos um tempo de ida e volta (RTT) de atraso antes que os dados sejam enviados.
- Embora a solicitação de conexão contenha o endereço completo do host B (que pode ser bastante grande, sendo um identificador global na rede), cada pacote de dados contém apenas um pequeno identificador, único em apenas um link. Assim, a sobrecarga por pacote causada pelo cabeçalho é reduzida em relação ao modelo de datagrama. Mais importante ainda, a consulta é rápida porque o número do circuito virtual pode ser tratado como um índice em uma tabela, em vez de uma chave que precisa ser consultada.
- Se um switch ou um link em uma conexão falhar, a conexão será interrompida e uma nova precisará ser estabelecida. Além disso, a antiga precisará ser desmontada para liberar espaço de armazenamento de tabela nos switches.
- A questão de como um switch decide para qual link encaminhar a solicitação de conexão foi esclarecida. Em essência, este é o mesmo problema que a construção da tabela de encaminhamento para o encaminhamento de datagramas, o que requer algum tipo de *algoritmo de roteamento* . O roteamento é descrito em uma seção posterior, e os algoritmos descritos ali

são geralmente aplicáveis a solicitações de configuração de roteamento, bem como a datagramas.

Um dos aspectos positivos dos circuitos virtuais é que, quando o host recebe autorização para enviar dados, ele já sabe bastante sobre a rede — por exemplo, que realmente existe uma rota para o receptor e que este está disposto e apto a receber dados. Também é possível alocar recursos ao circuito virtual no momento em que ele é estabelecido. Por exemplo, o X.25 (uma tecnologia de rede baseada em circuitos virtuais antiga e agora amplamente obsoleta) empregava a seguinte estratégia de três partes:

1. Buffers são alocados para cada circuito virtual quando o circuito é inicializado.
2. O protocolo de janela deslizante é executado entre cada par de nós ao longo do circuito virtual, e esse protocolo é complementado com controle de fluxo para impedir que o nó emissor ultrapasse os buffers alocados no nó receptor.
3. O circuito é rejeitado por um determinado nó se não houver buffers suficientes disponíveis naquele nó quando a mensagem de solicitação de conexão for processada.

Ao realizar essas três ações, cada nó garante os buffers necessários para enfileirar os pacotes que chegam naquele circuito. Essa estratégia básica é geralmente chamada de *controle de fluxo salto a salto.*

Em comparação, uma rede de datagramas não possui fase de estabelecimento de conexão, e cada comutador processa cada pacote independentemente, tornando menos óbvio como uma rede de datagramas alocaria recursos de forma significativa. Em vez disso, cada pacote que chega compete com todos os outros pacotes por espaço no buffer. Se não houver buffers livres, o pacote recebido deve ser descartado. Observamos, no entanto, que mesmo em uma rede baseada em datagramas, um host de origem frequentemente envia uma sequência de pacotes para o mesmo host de destino. É possível para cada comutador distinguir entre o conjunto de pacotes que ele

atualmente tem na fila, com base no par origem/destino, e, assim, para o comutador garantir que os pacotes pertencentes a cada par origem/destino estejam recebendo uma parcela justa dos buffers do comutador.

No modelo de circuito virtual, poderíamos imaginar que cada circuito tenha uma *qualidade de serviço* (QoS) diferente. Nesse cenário, o termo *qualidade de serviço* geralmente significa que a rede oferece ao usuário algum tipo de garantia relacionada ao desempenho, o que, por sua vez, implica que os switches reservem os recursos necessários para atender a essa garantia. Por exemplo, os switches ao longo de um determinado circuito virtual podem alocar uma porcentagem da largura de banda de cada link de saída para esse circuito. Como outro exemplo, uma sequência de switches pode garantir que os pacotes pertencentes a um determinado circuito não sejam atrasados (enfileirados) por mais de um determinado período de tempo.

Houve vários exemplos bem-sucedidos de tecnologias de circuitos virtuais ao longo dos anos, notadamente X.25, Frame Relay e Modo de Transferência Assíncrona (ATM). Com o sucesso do modelo sem conexão da internet, no entanto, nenhuma delas desfruta de grande popularidade atualmente. Uma das aplicações mais comuns de circuitos virtuais por muitos anos foi a construção de *redes privadas virtuais* (VPNs), um assunto discutido em uma seção posterior. Mesmo essa aplicação agora é amplamente suportada por tecnologias baseadas na internet.

Modo de Transferência Assíncrona (ATM)

O Modo de Transferência Assíncrona (ATM) é provavelmente a tecnologia de rede baseada em circuitos virtuais mais conhecida, embora já tenha passado do seu auge em termos de implantação. O ATM tornou-se uma tecnologia importante nas décadas de 1980 e início de 1990 por uma série de razões, entre elas a sua adoção pela indústria telefônica, que naquela época era menos ativa em redes de computadores (exceto como fornecedora de links a partir dos quais outras pessoas construíam redes). O ATM também estava no lugar certo na hora certa, como uma tecnologia de comutação de alta velocidade que surgiu justamente quando mídias compartilhadas,

como Ethernet e token rings, começavam a parecer lentas demais para muitos usuários de redes de computadores. De certa forma, o ATM era uma tecnologia concorrente da comutação Ethernet e era visto por muitos como um concorrente do IP também.

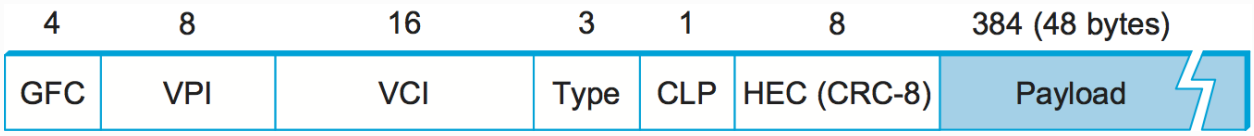


Figura 61. *Formato de célula ATM na UNI.*

A abordagem ATM possui algumas propriedades interessantes, o que a torna digna de um exame mais aprofundado. A imagem do formato de pacote ATM — mais comumente chamado de *célula* ATM — na Figura 61 ilustrará os pontos principais. Iremos pular os bits de controle de fluxo genérico (GFC), que nunca foram muito utilizados, e começaremos com os 24 bits rotulados como VPI (identificador de caminho virtual — 8 bits) e VCI (identificador de circuito virtual — 16 bits). Se considerarmos esses bits juntos como um único campo de 24 bits, eles correspondem ao identificador de circuito virtual apresentado acima. A razão para dividir o campo em duas partes foi permitir um nível de hierarquia: todos os circuitos com o mesmo VPI poderiam, em alguns casos, ser tratados como um grupo (um caminho virtual) e poderiam ser comutados juntos observando apenas o VPI, simplificando o trabalho de um comutador que poderia ignorar todos os bits VCI e reduzindo consideravelmente o tamanho da tabela VC.

Pulando para o último byte do cabeçalho, encontramos uma verificação de redundância cíclica (CRC) de 8 bits, conhecida como *verificação de erro de cabeçalho* (`HEC`). Ela utiliza CRC-8 e fornece detecção de erros e capacidade de correção de erros de bit único apenas no cabeçalho da célula. Proteger o cabeçalho da célula é particularmente importante, pois um erro no `VCI`cabeçalho causará a entrega incorreta da célula.

Provavelmente, o aspecto mais significativo a se notar sobre a célula ATM, e a razão pela qual ela é chamada de célula e não de pacote, é que ela tem apenas um tamanho: 53 bytes. Qual foi a razão para isso? Uma das principais razões foi facilitar a

implementação de switches de hardware. Quando o ATM estava sendo criado, em meados e no final da década de 1980, a Ethernet de 10 Mbps era a tecnologia de ponta em termos de velocidade. Para ir muito mais rápido, a maioria das pessoas pensava em termos de hardware. Além disso, no mundo da telefonia, as pessoas pensam grande quando pensam em switches — switches telefônicos geralmente atendem a dezenas de milhares de clientes. Pacotes de comprimento fixo acabam sendo muito úteis se você deseja construir switches rápidos e altamente escaláveis. Há duas razões principais para isso:

1. É mais fácil construir hardware para executar tarefas simples, e a tarefa de processar pacotes é mais simples quando você já sabe o tamanho de cada um.
2. Se todos os pacotes tiverem o mesmo comprimento, você poderá ter vários elementos de comutação, todos fazendo praticamente a mesma coisa em paralelo, e cada um deles levando o mesmo tempo para fazer seu trabalho.

Este segundo motivo, a habilitação do paralelismo, melhora significativamente a escalabilidade dos projetos de switches. Seria um exagero dizer que switches de hardware paralelos rápidos só podem ser construídos usando células de comprimento fixo. No entanto, é certamente verdade que as células facilitam a tarefa de construir esse hardware e que havia muito conhecimento disponível sobre como construir switches de célula em hardware na época em que os padrões ATM estavam sendo definidos. Acontece que esse mesmo princípio ainda é aplicado em muitos switches e roteadores hoje, mesmo que lidem com pacotes de comprimento variável — eles cortam esses pacotes em algum tipo de célula para encaminhá-los da porta de entrada para a porta de saída, mas tudo isso é interno ao switch.

Há outro bom argumento a favor de pequenas células ATM, relacionado à latência de ponta a ponta. O ATM foi projetado para transportar tanto chamadas telefônicas de voz (o caso de uso dominante na época) quanto dados. Como a voz tem baixa largura de banda, mas requisitos rigorosos de atraso, a última coisa que você quer é um pequeno pacote de voz enfileirado atrás de um grande pacote de dados em um switch. Se você

forçar todos os pacotes a serem pequenos (ou seja, do tamanho de uma célula), grandes pacotes de dados ainda poderão ser suportados pela remontagem de um conjunto de células em um pacote, e você terá o benefício de poder intercalar o encaminhamento de células de voz e células de dados em cada switch ao longo do caminho da origem ao destino. Essa ideia de usar pequenas células para melhorar a latência de ponta a ponta está viva e bem hoje em redes de acesso celular.

Tendo decidido usar pacotes pequenos e de comprimento fixo, a próxima pergunta era: qual o comprimento correto para fixá-los? Se você os tornar muito curtos, a quantidade de informações de cabeçalho que precisa ser transportada em relação à quantidade de dados que cabe em uma célula aumenta, de modo que a porcentagem de largura de banda do link que é realmente usada para transportar dados diminui. Ainda mais sério, se você construir um dispositivo que processa células em um número máximo de células por segundo, à medida que as células ficam mais curtas, a taxa total de dados cai em proporção direta ao tamanho da célula. Um exemplo desse dispositivo pode ser um adaptador de rede que remonta as células em unidades maiores antes de entregá-las ao host. O desempenho desse dispositivo depende diretamente do tamanho da célula. Por outro lado, se você tornar as células muito grandes, haverá um problema de desperdício de largura de banda causado pela necessidade de preencher os dados transmitidos para preencher uma célula completa. Se o tamanho da carga útil da célula for 48 bytes e você quiser enviar 1 byte, precisará enviar 47 bytes de preenchimento. Se isso acontecer com frequência, a utilização do link será muito baixa. A combinação da relação cabeçalho-carga relativamente alta, somada à frequência de envio de células parcialmente preenchidas, levou a uma ineficiência perceptível nas redes ATM, que alguns críticos chamaram de " *imposto de células"* .

Acontece que 48 bytes foram escolhidos para a carga útil da célula ATM como um meio-termo. Havia bons argumentos tanto para células maiores quanto para menores, e 48 bytes praticamente não agradou a ninguém — uma potência de dois certamente teria sido melhor para os computadores processarem.

3.1.3 Roteamento de origem

Uma terceira abordagem de comutação que não utiliza circuitos virtuais nem datagramas convencionais é conhecida como *roteamento de origem* . O nome deriva do fato de que todas as informações sobre a topologia da rede necessárias para comutar um pacote pela rede são fornecidas pelo host de origem.

Existem várias maneiras de implementar o roteamento de origem. Uma delas seria atribuir um número a cada saída de cada switch e colocar esse número no cabeçalho do pacote. A função de comutação é então muito simples: para cada pacote que chega em uma entrada, o switch lê o número da porta no cabeçalho e transmite o pacote nessa saída. No entanto, como geralmente haverá mais de um switch no caminho entre o host remetente e o receptor, o cabeçalho do pacote precisa conter informações suficientes para permitir que cada switch no caminho determine em qual saída o pacote precisa ser colocado. Uma maneira de fazer isso seria colocar uma lista ordenada de portas de switch no cabeçalho e rotacionar a lista para que o próximo switch no caminho esteja sempre no início da lista. A Figura 62 ilustra essa ideia.

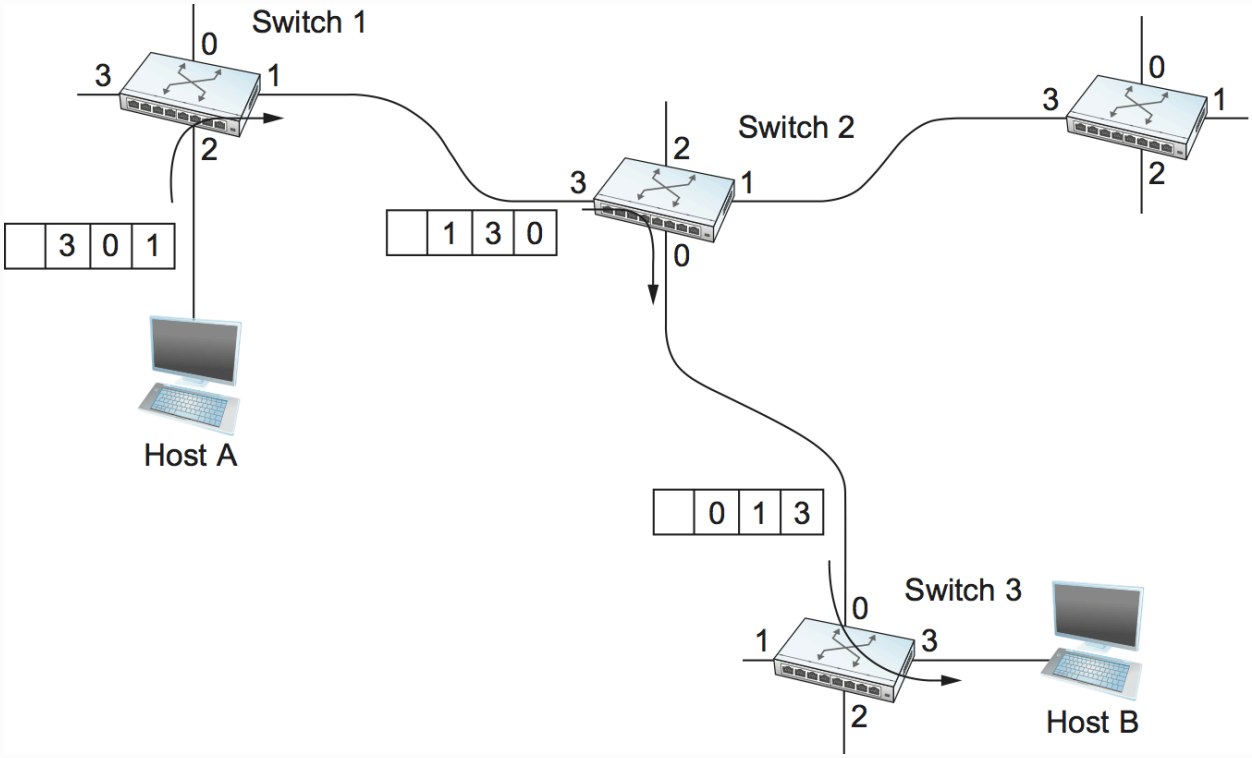


Figura 62. *Roteamento de origem em uma rede comutada (onde o switch lê o número mais à direita).*

Neste exemplo, o pacote precisa atravessar três switches para ir do host A ao host B. No switch 1, ele precisa sair na porta 1, no próximo switch ele precisa sair na porta 0 e no terceiro switch ele precisa sair na porta 3. Assim, o cabeçalho original quando o pacote sai do host A contém a lista de portas (3, 0, 1), onde assumimos que cada switch lê o elemento mais à direita da lista. Para garantir que o próximo switch receba as informações apropriadas, cada switch rotaciona a lista após ler sua própria entrada. Assim, o cabeçalho do pacote quando ele sai do switch 1 a caminho do switch 2 agora é (1, 3, 0); o switch 2 executa outra rotação e envia um pacote com (0, 1, 3) no cabeçalho. Embora não mostrado, o switch 3 executa ainda outra rotação, restaurando o cabeçalho para o que era quando o host A o enviou.

Há vários pontos a serem observados sobre essa abordagem. Primeiro, ela pressupõe que o host A conhece a topologia da rede o suficiente para formar um cabeçalho com todas as direções corretas para cada switch no caminho. Isso é um pouco análogo ao problema de construir as tabelas de encaminhamento em uma rede de datagramas ou

descobrir para onde enviar um pacote de configuração em uma rede de circuitos virtuais. Na prática, porém, é o primeiro switch na entrada da rede (em oposição ao host final conectado a esse switch) que anexa a rota de origem.

Em segundo lugar, observe que não podemos prever o tamanho necessário do cabeçalho, pois ele deve ser capaz de conter uma palavra de informação para cada switch no caminho. Isso implica que os cabeçalhos provavelmente têm comprimento variável, sem limite superior, a menos que possamos prever com absoluta certeza o número máximo de switches pelos quais um pacote precisará passar.

Terceiro, existem algumas variações dessa abordagem. Por exemplo, em vez de rotacionar o cabeçalho, cada switch poderia simplesmente remover o primeiro elemento conforme o utiliza. A rotação tem uma vantagem sobre a remoção, no entanto: o host B obtém uma cópia do cabeçalho completo, o que pode ajudá-lo a descobrir como retornar ao host A. Outra alternativa é fazer com que o cabeçalho carregue um ponteiro para a entrada atual da "próxima porta", de modo que cada switch apenas atualize o ponteiro em vez de rotacionar o cabeçalho; isso pode ser mais eficiente de implementar. Mostramos essas três abordagens na Figura 63. Em cada caso, a entrada que esse switch precisa ler é `A`, e a entrada que o próximo switch precisa ler é `B`.

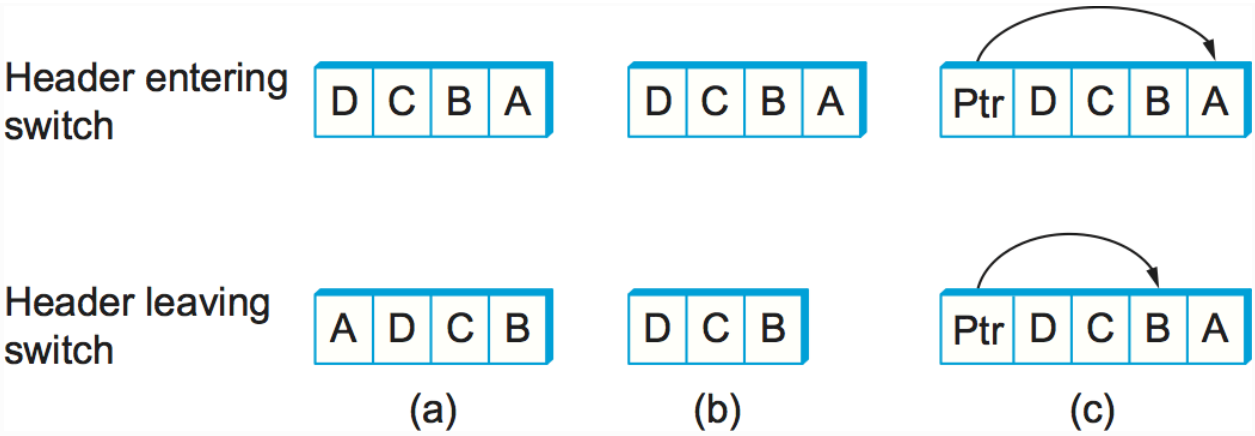


Figura 63. *Três maneiras de manipular cabeçalhos para roteamento de origem: (a) rotação; (b) remoção; (c) ponteiro. Os rótulos são lidos da direita para a esquerda.*

O roteamento de origem pode ser usado tanto em redes de datagramas quanto em redes de circuitos virtuais. Por exemplo, o Protocolo de Internet (IP), que é um protocolo de datagramas, inclui uma opção de roteamento de origem que permite que pacotes selecionados sejam roteados na origem, enquanto a maioria é comutada como datagramas convencionais. O roteamento de origem também é usado em algumas redes de circuitos virtuais como meio de encaminhar a solicitação de configuração inicial ao longo do caminho da origem ao destino.

As rotas de origem são, às vezes, categorizadas como *estritas* ou *flexíveis* . Em uma rota de origem rígida, cada nó ao longo do caminho deve ser especificado, enquanto uma rota de origem flexível especifica apenas um conjunto de nós a serem percorridos, sem dizer exatamente como ir de um nó ao próximo. Uma rota de origem flexível pode ser considerada um conjunto de pontos de referência em vez de uma rota completamente especificada. A opção flexível pode ser útil para limitar a quantidade de informações que uma origem deve obter para criar uma rota de origem. Em qualquer rede razoavelmente grande, é provável que seja difícil para um host obter as informações completas do caminho necessárias para construir corretamente uma rota de origem rígida para qualquer destino. Mas ambos os tipos de rotas de origem encontram aplicação em certos cenários, como veremos em capítulos posteriores.

3.2 Ethernet comutada

Tendo discutido algumas das ideias básicas por trás da comutação, agora nos concentramos mais em uma tecnologia de comutação específica: *Ethernet Comutada* . Os switches usados para construir essas redes, frequentemente chamados de *switches L2* , são amplamente utilizados em redes de campus e corporativas. Historicamente, eles eram mais comumente chamados de *pontes* , pois eram usados para "conectar" segmentos Ethernet para construir uma *LAN estendida* . Mas hoje, a maioria das redes implementa Ethernet em uma configuração ponto a ponto, com esses links interconectados por switches L2 para formar uma Ethernet comutada.

O texto a seguir começa com a perspectiva histórica (usando pontes para conectar um conjunto de segmentos Ethernet) e, em seguida, passa para a perspectiva amplamente utilizada hoje (usando switches L2 para conectar um conjunto de links ponto a ponto). Mas, independentemente de chamarmos o dispositivo de ponte ou switch — e a rede que você constrói de LAN estendida ou Ethernet comutada —, os dois se comportam *exatamente* da mesma maneira.

Para começar, suponha que você tenha um par de Ethernets que deseja interconectar. Uma abordagem que você pode tentar é colocar um repetidor entre elas. No entanto, essa não seria uma solução viável se isso excedesse as limitações físicas da Ethernet. (Lembre-se de que não são permitidos mais de quatro repetidores entre qualquer par de hosts e um comprimento total de no máximo 2.500 m.) Uma alternativa seria colocar um nó com um par de adaptadores Ethernet entre as duas Ethernets e fazer com que o nó encaminhasse quadros de uma Ethernet para a outra. Esse nó seria diferente de um repetidor, que opera em bits, não em quadros, e apenas copia cegamente os bits recebidos de uma interface para outra. Em vez disso, esse nó implementaria completamente os protocolos de detecção de colisões e acesso à mídia da Ethernet em cada interface. Portanto, as restrições de comprimento e número de hosts da Ethernet, que se referem ao gerenciamento de colisões, não se aplicariam ao par combinado de Ethernets conectadas dessa maneira. Este dispositivo opera em modo promíscuo, aceitando todos os quadros transmitidos em qualquer uma das Ethernets e encaminhando-os para a outra.

Em suas variantes mais simples, as pontes simplesmente aceitam quadros LAN em suas entradas e os encaminham para todas as outras saídas. Essa estratégia simples foi usada pelas primeiras pontes, mas apresenta algumas limitações bastante sérias, como veremos a seguir. Diversos refinamentos foram adicionados ao longo dos anos para tornar as pontes um mecanismo eficaz para interconectar um conjunto de LANs. O restante desta seção aborda os detalhes mais interessantes.

3.2.1 Pontes de Aprendizagem

A primeira otimização que podemos fazer em uma ponte é observar que ela não precisa encaminhar todos os quadros que recebe. Considere a ponte na Figura 64. Sempre que um quadro do host A endereçado ao host B chega na porta 1, não há necessidade de a ponte encaminhá-lo pela porta 2. A questão, então, é como uma ponte descobre em qual porta os vários hosts residem?

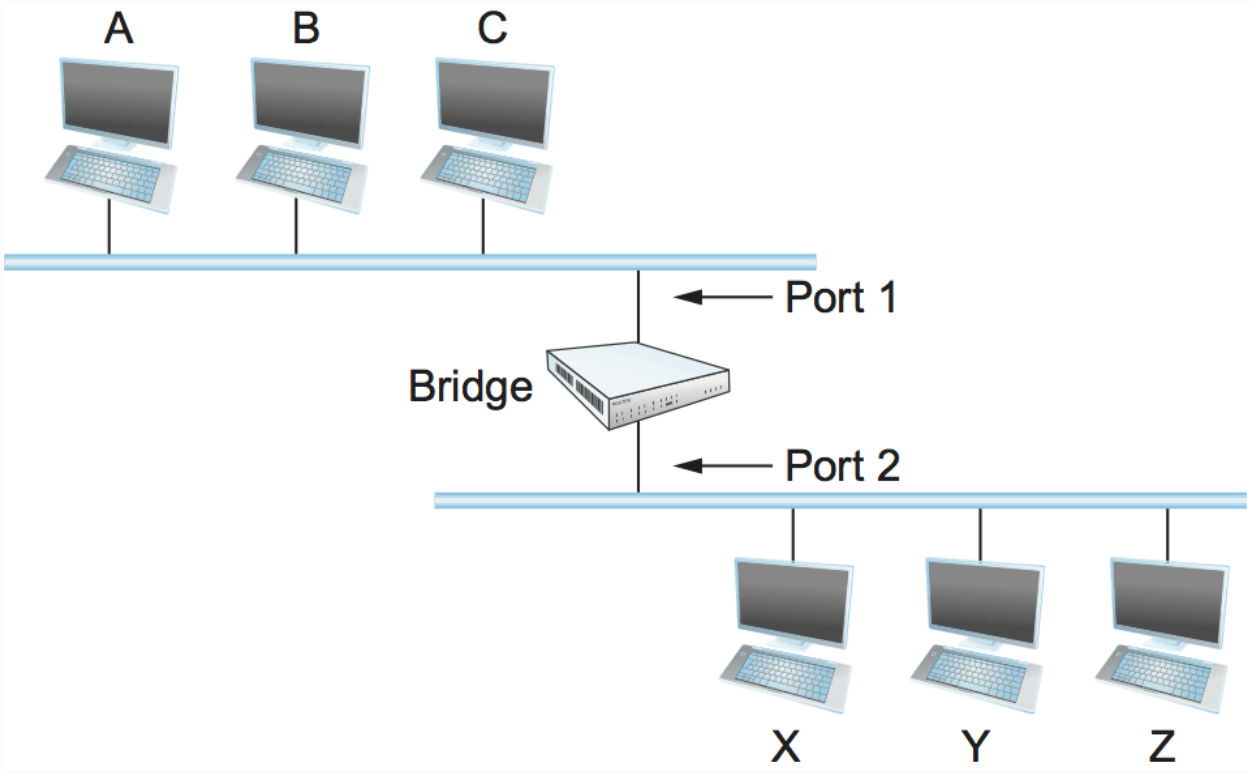


Figura 64. *Ilustração de uma ponte de aprendizagem.*

Uma opção seria fazer com que um humano baixasse uma tabela na ponte, semelhante à apresentada na Tabela 9. Assim, sempre que a ponte recebesse um quadro na porta 1 endereçado ao host A, ela não encaminharia o quadro na porta 2; não haveria necessidade, pois o host A já teria recebido o quadro diretamente na LAN conectada à porta 1. Sempre que um quadro endereçado ao host A fosse recebido na porta 2, a ponte encaminharia o quadro na porta 1.

Hospedar	Porta
UM	1
B	1
C	1
X	2
E	2
Z	2

Ter um humano mantendo essa tabela é muito trabalhoso, e existe um truque simples pelo qual uma ponte pode aprender essas informações por si mesma. A ideia é que cada ponte inspecione o endereço *de origem* em todos os quadros que recebe. Assim, quando o host A envia um quadro para um host em qualquer um dos lados da ponte, a ponte recebe esse quadro e registra o fato de que um quadro do host A acabou de ser recebido na porta 1. Dessa forma, a ponte pode construir uma tabela semelhante à Tabela 9 .

Observe que uma ponte que utiliza essa tabela implementa uma versão do modelo de encaminhamento por datagrama (ou sem conexão) descrito anteriormente. Cada pacote carrega um endereço global, e a ponte decide para qual saída enviar o pacote, consultando esse endereço em uma tabela.

Quando uma ponte inicializa pela primeira vez, esta tabela está vazia; entradas são adicionadas ao longo do tempo. Além disso, um tempo limite é associado a cada entrada, e a ponte descarta a entrada após um período de tempo especificado. Isso serve para proteger contra a situação em que um host — e, consequentemente, seu endereço LAN — é movido de uma rede para outra. Portanto, esta tabela não está necessariamente completa. Caso a ponte receba um quadro endereçado a um host que não está atualmente na tabela, ela prossegue e encaminha o quadro para todas as outras portas. Em outras palavras, esta tabela é simplesmente uma otimização que filtra alguns quadros; não é necessária para a correção.

3.2.2 Implementação

O código que implementa o algoritmo da ponte de aprendizagem é bastante simples, e o esboçamos aqui. A estrutura `BridgeEntry`define uma única entrada na tabela de encaminhamento da ponte; estas são armazenadas em uma `Map`estrutura (que suporta as operações `mapCreate`, `mapBind`e `mapResolve`) para permitir que as entradas sejam localizadas de forma eficiente quando os pacotes chegam de fontes que já estão na tabela. A constante `MAX_TTL`especifica por quanto tempo uma entrada é mantida na tabela antes de ser descartada.

```
#define BRIDGE_TAB_SIZE   1024  /* max size of bridging table */
#define MAX_TTL           120   /* time (in seconds) before an entry is flushed */

typedef struct {
    MacAddr     destination;    /* MAC address of a node */
    int         ifnumber;       /* interface to reach it */
    u_short     TTL;            /* time to live */
    Binding     binding;        /* binding in the Map */
} BridgeEntry;

int     numEntries = 0;
Map     bridgeMap = mapCreate(BRIDGE_TAB_SIZE, sizeof(BridgeEntry));
```

A rotina que atualiza a tabela de encaminhamento quando um novo pacote chega é dada por `updateTable`. Os argumentos passados são o endereço de controle de acesso à mídia (MAC) de origem contido no pacote e o número da interface na qual ele foi recebido. Outra rotina, não mostrada aqui, é invocada em intervalos regulares, verifica as entradas na tabela de encaminhamento e decrementa o `TTL`campo (tempo de vida) de cada entrada, descartando quaisquer entradas que `TTL`tenham atingido 0. Observe que o `TTL`é redefinido para `MAX_TTL`sempre que um pacote chega para atualizar uma entrada existente na tabela e que a interface na qual o destino pode ser alcançado é atualizada para refletir o pacote recebido mais recentemente.

```
void
updateTable (MacAddr src, int inif)
{
    BridgeEntry       *b;

    if (mapResolve(bridgeMap, &src, (void **)&b) == FALSE )
    {
        /* this address is not in the table, so try to add it */
        if (numEntries < BRIDGE_TAB_SIZE)
        {
            b = NEW(BridgeEntry);
            b->binding = mapBind( bridgeMap, &src, b);
            /* use source address of packet as dest. address in table */
            b->destination = src;
            numEntries++;
        }
        else
        {
            /* can't fit this address in the table now, so give up */
            return;
        }
    }
    /* reset TTL and use most recent input interface */
    b->TTL = MAX_TTL;
    b->ifnumber = inif;
}
```

Observe que esta implementação adota uma estratégia simples no caso em que a tabela de ponte está lotada — ela simplesmente não consegue adicionar o novo endereço. Lembre-se de que a integridade da tabela de ponte não é necessária para o

encaminhamento correto; apenas otimiza o desempenho. Se houver alguma entrada na tabela que não esteja sendo usada no momento, ela eventualmente expirará e será removida, criando espaço para uma nova entrada. Uma abordagem alternativa seria invocar algum tipo de algoritmo de substituição de cache ao encontrar a tabela cheia; por exemplo, poderíamos localizar e remover a entrada com o menor TTL para acomodar a nova entrada.

3.2.3 Algoritmo de Árvore de Extensão

A estratégia anterior funciona perfeitamente até que a rede entre em um loop, e nesse caso ela falha de forma terrível — os quadros podem ser encaminhados para sempre. Isso é fácil de ver no exemplo da Figura 65 , onde os switches S1, S4 e S6 formam um loop.

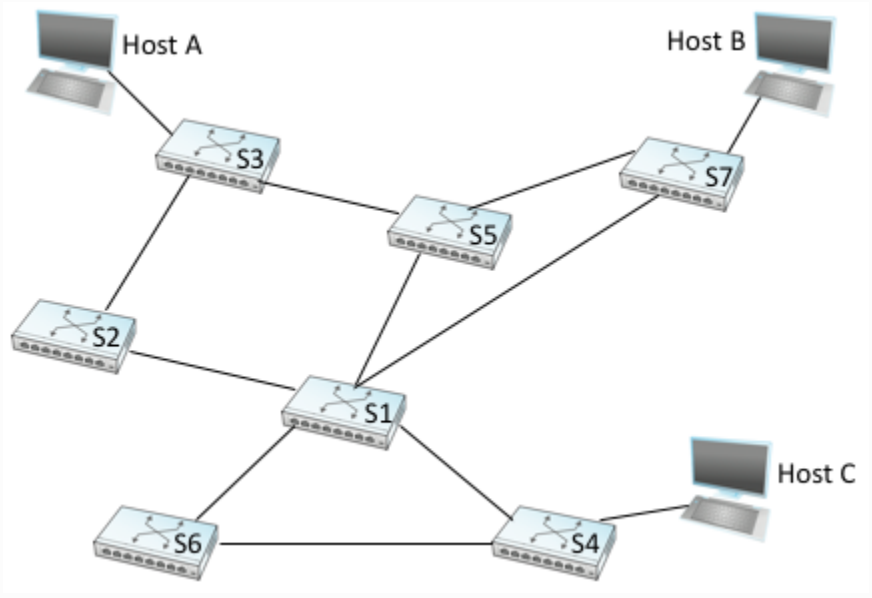


Figura 65. *Ethernet comutada com loops.*

Observe que agora estamos migrando de chamar cada dispositivo de encaminhamento de ponte (conectando segmentos que podem alcançar vários outros dispositivos) para switches L2 (conectando links ponto a ponto que alcançam apenas um outro dispositivo). Para manter o exemplo gerenciável, incluímos apenas três hosts. Na

prática, os switches normalmente têm 16, 24 ou 48 portas, o que significa que podem se conectar a essa quantidade de hosts (e outros switches).

Em nosso exemplo de rede comutada, suponha que um pacote entre no switch S4 vindo do Host C e que o endereço de destino ainda não esteja na tabela de encaminhamento de nenhum switch: o S4 envia uma cópia do pacote por suas outras duas portas: para os switches S1 e S6. O switch S6 encaminha o pacote para o S1 (e, enquanto isso, o S1 encaminha o pacote para o S6), que, por sua vez, encaminham seus pacotes de volta para o S4. O switch S4 ainda não possui esse destino em sua tabela, então encaminha o pacote por suas outras duas portas. Não há nada que impeça esse ciclo de se repetir infinitamente, com pacotes circulando em ambas as direções entre S1, S4 e S6.

Por que uma Ethernet comutada (ou LAN estendida) apresentaria um loop? Uma possibilidade é que a rede seja gerenciada por mais de um administrador, por exemplo, porque abrange vários departamentos de uma organização. Nesse cenário, é possível que nenhuma pessoa conheça toda a configuração da rede, o que significa que um switch que fecha um loop pode ser adicionado sem que ninguém saiba. Um segundo cenário, mais provável, é que os loops sejam incorporados à rede propositalmente — para fornecer redundância em caso de falha. Afinal, uma rede sem loops precisa apenas de uma falha de link para se dividir em duas partições separadas.

Seja qual for a causa, os switches devem ser capazes de lidar corretamente com loops. Esse problema é resolvido fazendo com que os switches executem um algoritmo *de árvore de extensão* distribuída . Se você pensar na rede como sendo representada por um grafo que possivelmente possui loops (ciclos), então uma árvore de extensão é um subgrafo desse grafo que cobre (abrange) todos os vértices, mas não contém ciclos. Ou seja, uma árvore de extensão mantém todos os vértices do grafo original, mas descarta algumas das arestas. Por exemplo, a Figura 66 mostra um grafo cíclico à esquerda e uma das possíveis muitas árvores de extensão à direita.

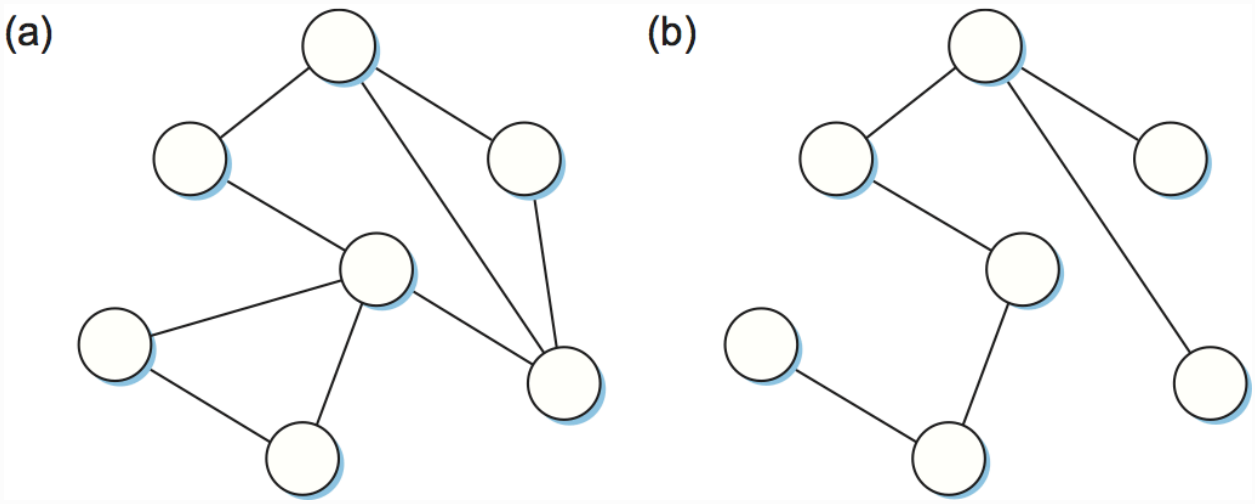


Figura 66. *Exemplo de (a) um gráfico cíclico; (b) uma árvore de extensão correspondente.*

A ideia de uma árvore de abrangência é bastante simples: é um subconjunto da topologia de rede real, sem loops, que alcança todos os dispositivos da rede. A parte difícil é como todos os switches coordenam suas decisões para chegar a uma visão única da árvore de abrangência. Afinal, uma topologia normalmente pode ser coberta por várias árvores de abrangência. A resposta está no protocolo da árvore de abrangência, que descreveremos a seguir.

O algoritmo de árvore de abrangência, desenvolvido por Radia Perlman, então na Digital Equipment Corporation, é um protocolo usado por um conjunto de switches para concordar com uma árvore de abrangência para uma rede específica. (A especificação IEEE 802.1 é baseada neste algoritmo.) Na prática, isso significa que cada switch decide as portas pelas quais está ou não disposto a encaminhar quadros. De certa forma, é removendo portas da topologia que a rede é reduzida a uma árvore acíclica. É até possível que um switch inteiro não participe do encaminhamento de quadros, o que parece um tanto estranho à primeira vista. O algoritmo é dinâmico, no entanto, o que significa que os switches estão sempre preparados para se reconfigurar em uma nova árvore de abrangência caso algum switch falhe, e assim essas portas e switches não utilizados fornecem a capacidade redundante necessária para a recuperação de falhas.

A ideia principal da árvore de abrangência é que os switches selecionem as portas pelas quais encaminharão os quadros. O algoritmo seleciona as portas da seguinte forma. Cada switch possui um identificador único; para nossos propósitos, usamos os rótulos S1, S2, S3 e assim por diante. O algoritmo primeiro elege o switch com o menor ID como a raiz da árvore de abrangência; exatamente como essa eleição ocorre é descrito abaixo. O switch raiz sempre encaminha os quadros por todas as suas portas. Em seguida, cada switch calcula o caminho mais curto para a raiz e observa qual de suas portas está nesse caminho. Essa porta também é selecionada como o caminho preferencial do switch para a raiz. Finalmente, para levar em conta a possibilidade de haver outro switch conectado às suas portas, o switch elege um único switch *designado* que será responsável por encaminhar os quadros para a raiz. Cada switch designado é aquele que está mais próximo da raiz. Se dois ou mais switches estiverem igualmente próximos da raiz, os identificadores dos switches são usados para desempatar, e o menor ID vence. É claro que cada switch pode estar conectado a mais de um switch, portanto, ele participa da eleição de um switch designado para cada porta. Na prática, isso significa que cada switch decide se é o switch designado em relação a cada uma de suas portas. O switch encaminha quadros pelas portas para as quais é o switch designado.

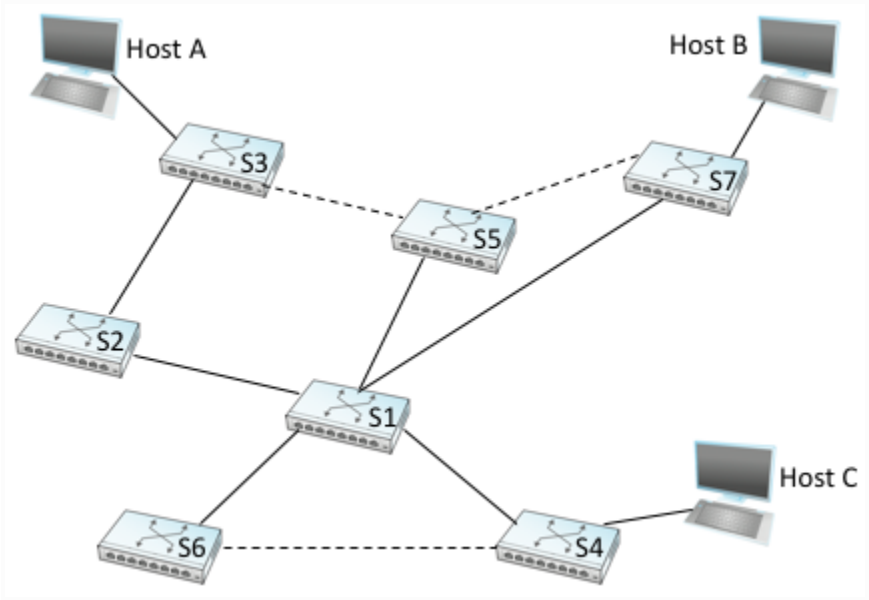


Figura 67. *Árvore de extensão com algumas portas não selecionadas.*

A Figura 67 mostra a árvore de extensão que corresponde à rede mostrada na Figura 65. Neste exemplo, S1 é a raiz, pois possui o menor ID. Observe que S3 e S5 estão conectados entre si, mas S5 é o switch designado, pois está mais próximo da raiz. Da mesma forma, S5 e S7 estão conectados entre si, mas neste caso S5 é o switch designado, pois possui o menor ID; ambos estão à mesma distância de S1.

Embora seja possível para um ser humano observar a rede apresentada na Figura 65 e calcular a árvore de abrangência apresentada na Figura 67 de acordo com as regras apresentadas acima, os switches não têm o luxo de ver a topologia de toda a rede, muito menos de espiar o interior de outros switches para ver seus IDs. Em vez disso, eles precisam trocar mensagens de configuração entre si e, com base nessas mensagens, decidir se são a raiz ou um switch designado.

Especificamente, as mensagens de configuração contêm três informações:

1. O ID do switch que está enviando a mensagem.
2. O ID do que o switch de envio acredita ser o switch raiz.
3. A distância, medida em saltos, do switch de envio até o switch raiz.

Cada switch registra a melhor mensagem de configuração atual que viu em cada uma de suas portas (a “melhor” é definida abaixo), incluindo mensagens que recebeu de outros switches e mensagens que ele próprio transmitiu.

Inicialmente, cada switch pensa que é a raiz e, portanto, envia uma mensagem de configuração em cada uma de suas portas, identificando-se como raiz e fornecendo uma distância de 0 até a raiz. Ao receber uma mensagem de configuração em uma porta específica, o switch verifica se a nova mensagem é melhor do que a melhor mensagem de configuração registrada para aquela porta. A nova mensagem de configuração é considerada *melhor* do que as informações registradas atualmente se qualquer uma das seguintes condições for verdadeira:

- Ele identifica uma raiz com um ID menor.

- Ele identifica uma raiz com um ID igual, mas com uma distância menor.
- O ID da raiz e a distância são iguais, mas o switch de envio tem um ID menor

Se a nova mensagem for melhor do que as informações registradas atualmente, o switch descarta as informações antigas e salva as novas. No entanto, ele primeiro adiciona 1 ao campo de distância até a raiz, já que o switch está um salto mais distante da raiz do que o switch que enviou a mensagem.

Quando um switch recebe uma mensagem de configuração indicando que não é a raiz — ou seja, uma mensagem de um switch com um ID menor — ele para de gerar mensagens de configuração por conta própria e, em vez disso, encaminha apenas mensagens de configuração de outros switches, após adicionar primeiro 1 ao campo de distância. Da mesma forma, quando um switch recebe uma mensagem de configuração indicando que não é o switch designado para aquela porta — ou seja, uma mensagem de um switch que está mais próximo da raiz ou igualmente distante da raiz, mas com um ID menor — o switch para de enviar mensagens de configuração por aquela porta. Assim, quando o sistema se estabiliza, apenas o switch raiz ainda está gerando mensagens de configuração, e os outros switches estão encaminhando essas mensagens apenas pelas portas para as quais são os switches designados. Nesse ponto, uma árvore de abrangência foi construída e todos os switches concordam sobre quais portas estão em uso para a árvore de abrangência. Somente essas portas podem ser usadas para encaminhar pacotes de dados.

Vejamos como isso funciona com um exemplo. Considere o que aconteceria na Figura 67 se a energia elétrica tivesse acabado de ser restaurada em um campus, de modo que todos os switches inicializassem aproximadamente ao mesmo tempo. Todos os switches começariam afirmando ser a raiz. Denotamos uma mensagem de configuração do nó X, na qual ele afirma estar a uma distância d do nó raiz Y, como (Y,d,X). Concentrando-nos na atividade em S3, uma sequência de eventos se desenrolaria da seguinte forma:

1. S3 recebe (S2, 0, S2).

2. Como 2 < 3, S3 aceita S2 como raiz.
3. S3 adiciona um à distância anunciada por S2 (0) e, portanto, envia (S2, 1, S3) em direção a S5.
4. Enquanto isso, S2 aceita S1 como raiz porque tem o ID mais baixo e envia (S1, 1, S2) para S3.
5. S5 aceita S1 como raiz e envia (S1, 1, S5) para S3.
6. O S3 aceita S1 como raiz e observa que tanto S2 quanto S5 estão mais próximos da raiz do que ele, mas S2 tem o id menor, então ele permanece no caminho do S3 para a raiz.

Isso deixa o S3 com portas ativas, conforme mostrado na Figura 67. Observe que os hosts A e B não conseguem se comunicar pelo caminho mais curto (via S5) porque os quadros precisam "subir na árvore e descer", mas esse é o preço que você paga para evitar loops.

Mesmo após a estabilização do sistema, o switch raiz continua a enviar mensagens de configuração periodicamente, e os outros switches continuam a encaminhá-las, conforme descrito anteriormente. Caso um switch específico falhe, os switches a jusante não receberão essas mensagens de configuração e, após aguardar um período de tempo especificado, voltarão a se identificar como raiz, e o algoritmo entrará em ação novamente para eleger uma nova raiz e novos switches designados.

Um ponto importante a ser observado é que, embora o algoritmo seja capaz de reconfigurar a árvore de extensão sempre que um switch falha, ele não é capaz de encaminhar quadros por caminhos alternativos para fins de roteamento em torno de um switch congestionado.

3.2.4 Transmissão e Multidifusão

A discussão anterior se concentra em como os switches encaminham quadros unicast de uma porta para outra. Como o objetivo de um switch é estender uma LAN de forma transparente por várias redes, e como a maioria das LANs suporta broadcast e

multicast, os switches também devem suportar esses dois recursos. O broadcast é simples — cada switch encaminha um quadro com um endereço de broadcast de destino em cada porta ativa (selecionada) diferente daquela em que o quadro foi recebido.

O multicast pode ser implementado exatamente da mesma maneira, com cada host decidindo por si mesmo se aceita ou não a mensagem. É exatamente isso que acontece na prática. Observe, no entanto, que, como nem todos os hosts são membros de um grupo multicast específico, é possível fazer melhor. Especificamente, o algoritmo de árvore de abrangência pode ser estendido para podar redes sobre as quais quadros multicast não precisam ser encaminhados. Considere um quadro enviado ao grupo M por um host A na Figura 67. Se o host C não pertencer ao grupo M, não há necessidade de o switch S4 encaminhar os quadros por essa rede.

Como um determinado switch aprenderia se deve encaminhar um quadro multicast por uma determinada porta? Ele aprende exatamente da mesma forma que um switch aprende se deve encaminhar um quadro unicast por uma porta específica — observando os endereços *de origem* que recebe por essa porta. É claro que grupos normalmente não são a origem dos quadros, então precisamos trapacear um pouco. Em particular, cada host membro do grupo M deve enviar periodicamente um quadro com o endereço do grupo M no campo de origem do cabeçalho do quadro. Esse quadro teria como endereço de destino o endereço multicast dos switches.

Embora a extensão multicast descrita anteriormente tenha sido proposta, ela não foi amplamente adotada. Em vez disso, o multicast é implementado exatamente da mesma forma que a transmissão.

3.2.5 LANs virtuais (VLANs)

Uma limitação dos switches é que eles não são escaláveis. Não é realista conectar mais do que alguns switches, onde na prática *poucos* normalmente significam "dezenas de". Um motivo para isso é que o algoritmo de árvore de abrangência é escalável

linearmente; ou seja, não há nenhuma disposição para impor uma hierarquia no conjunto de switches. Um segundo motivo é que os switches encaminham todos os quadros de broadcast. Embora seja razoável que todos os hosts dentro de um ambiente limitado (por exemplo, um departamento) vejam as mensagens de broadcast uns dos outros, é improvável que todos os hosts em um ambiente maior (por exemplo, uma grande empresa ou universidade) queiram ser incomodados pelas mensagens de broadcast uns dos outros. Em outras palavras, o broadcast não é escalável e, como consequência, as redes baseadas em L2 não são escaláveis.

Uma abordagem para aumentar a escalabilidade é a *LAN virtual* (VLAN). As VLANs permitem que uma única LAN estendida seja particionada em várias LANs aparentemente separadas. Cada LAN virtual recebe um identificador (às vezes chamado de *cor*), e os pacotes só podem trafegar de um segmento para outro se ambos os segmentos tiverem o mesmo identificador. Isso limita o número de segmentos em uma LAN estendida que receberão qualquer pacote de transmissão.

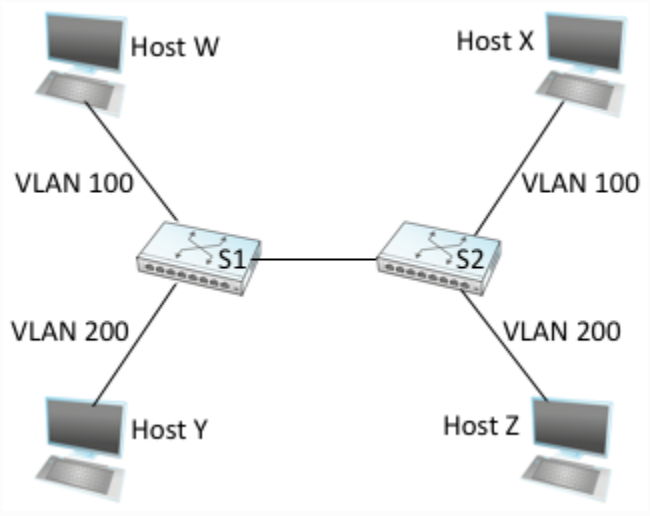


Figura 68. *Duas LANs virtuais compartilham um backbone comum.*

Podemos ver como as VLANs funcionam com um exemplo. A Figura 68 mostra quatro hosts e dois switches. Na ausência de VLANs, qualquer pacote de broadcast de qualquer host alcançará todos os outros hosts. Agora, vamos supor que definimos os segmentos conectados aos hosts W e X como estando em uma VLAN, que

chamaremos de VLAN 100. Também definimos os segmentos que se conectam aos hosts Y e Z como estando na VLAN 200. Para isso, precisamos configurar um ID de VLAN em cada porta dos switches S1 e S2. O link entre S1 e S2 é considerado como estando em ambas as VLANs.

Quando um pacote enviado pelo host X chega ao switch S2, o switch observa que ele veio por uma porta configurada como VLAN 100. Ele insere um cabeçalho de VLAN entre o cabeçalho Ethernet e seu payload. A parte interessante do cabeçalho de VLAN é o ID da VLAN; neste caso, esse ID é definido como 100. O switch agora aplica suas regras normais de encaminhamento para o pacote, com a restrição adicional de que o pacote não pode ser enviado por uma interface que não faça parte da VLAN 100. Portanto, sob nenhuma circunstância o pacote — mesmo um pacote de broadcast — será enviado pela interface para o host Z, que está na VLAN 200. O pacote, no entanto, é encaminhado para o switch S1, que segue as mesmas regras e, portanto, pode encaminhá-lo para o host W, mas não para o host Y.

Um recurso interessante das VLANs é a possibilidade de alterar a topologia lógica sem mover nenhum fio ou alterar endereços. Por exemplo, se quiséssemos que o link que se conecta ao host Z fizesse parte da VLAN 100 e, assim, permitisse que X, W e Z estivessem na mesma LAN virtual, precisaríamos alterar apenas uma parte da configuração no switch S2.

O suporte a VLANs requer uma extensão bastante simples da especificação original do cabeçalho 802.1, inserindo um `VID`campo VLAN ID ( ) de 12 bits entre os campos `SrcAddr`e `Type`, conforme mostrado na Figura 69. (Este VID é normalmente chamado de *VLAN Tag* .) Na verdade, há 32 bits inseridos no meio do cabeçalho, mas os primeiros 16 bits são usados para preservar a compatibilidade com a especificação original (eles são usados para indicar que este quadro inclui a extensão VLAN); os outros quatro bits contêm informações de controle usadas para priorizar quadros. Isso significa que é possível mapear`Type = 0x8100`

212=4096

redes virtuais em uma única LAN física.

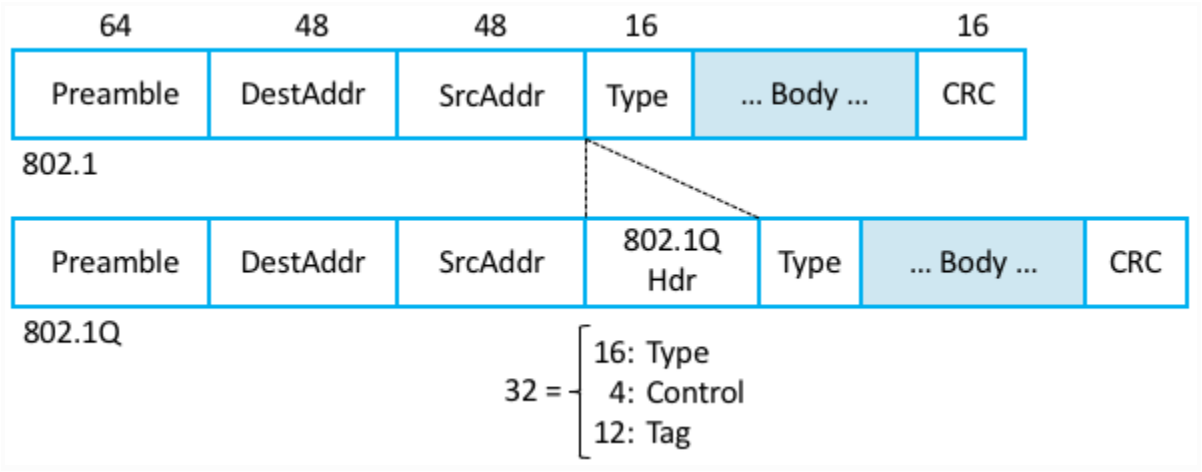


Figura 69. *Etiqueta VLAN 802.1Q incorporada em um cabeçalho Ethernet (802.1).*

Concluímos esta discussão observando que há outra limitação nas redes construídas pela interconexão de switches L2: a falta de suporte à heterogeneidade. Ou seja, os switches são limitados quanto aos tipos de redes que podem interconectar. Em particular, os switches utilizam o cabeçalho de quadro da rede e, portanto, podem suportar apenas redes que tenham exatamente o mesmo formato de endereço. Por exemplo, switches podem ser usados para conectar redes Ethernet e baseadas em 802.11 entre si, uma vez que compartilham um formato de cabeçalho comum, mas switches não se generalizam facilmente para outros tipos de redes com diferentes formatos de endereçamento, como ATM, SONET, PON ou a rede celular. A próxima seção explica como lidar com essa limitação, bem como escalar redes comutadas para tamanhos ainda maiores.

3.3 Internet (IP)

Na seção anterior, vimos que era possível construir LANs razoavelmente grandes usando pontes e switches LAN, mas que tais abordagens eram limitadas em sua capacidade de escalabilidade e de lidar com a heterogeneidade. Nesta seção, exploramos algumas maneiras de ir além das limitações das redes em ponte, permitindo-nos construir redes grandes e altamente heterogêneas com roteamento

razoavelmente eficiente. Chamamos essas redes de *"internetworks".* Continuaremos a discussão sobre como construir uma internetwork verdadeiramente global no próximo capítulo, mas, por enquanto, exploraremos o básico. Começaremos considerando com mais cuidado o significado da palavra " *internetwork"* .

3.3.1 O que é uma inter-rede?

Usamos o termo *interconexão de redes* , ou às vezes apenas *internet* com i minúsculo *,* para nos referirmos a uma coleção arbitrária de redes interconectadas para fornecer algum tipo de serviço de entrega de pacotes de host para host. Por exemplo, uma corporação com muitos sites pode construir uma interconexão de redes privadas interconectando as LANs em seus diferentes sites com links ponto a ponto alugados da companhia telefônica. Quando falamos sobre a interconexão de redes globais amplamente utilizada, à qual uma grande porcentagem de redes está conectada atualmente, a chamamos de *Internet* com I maiúsculo *.* Mantendo a abordagem dos primeiros princípios deste livro, queremos principalmente que você aprenda sobre os princípios da interconexão de redes com " *i* minúsculo ", mas ilustramos essas ideias com exemplos do mundo real da Internet com " *I maiúsculo".*

Outra terminologia que pode ser confusa é a diferença entre redes, sub-redes e inter-redes. Evitaremos sub-redes (ou sub-redes) completamente até uma seção posterior. Por enquanto, usamos *"rede"* para significar uma rede conectada diretamente ou comutada, do tipo descrito na seção anterior e no capítulo anterior. Tal rede usa uma tecnologia, como 802.11 ou Ethernet. Uma *inter-rede* é um conjunto interconectado dessas redes. Às vezes, para evitar ambiguidade, nos referimos às redes subjacentes que estamos interconectando como redes *físicas* . Uma internet é uma rede *lógica* construída a partir de um conjunto de redes físicas. Nesse contexto, um conjunto de segmentos Ethernet conectados por pontes ou switches ainda seria visto como uma única rede.

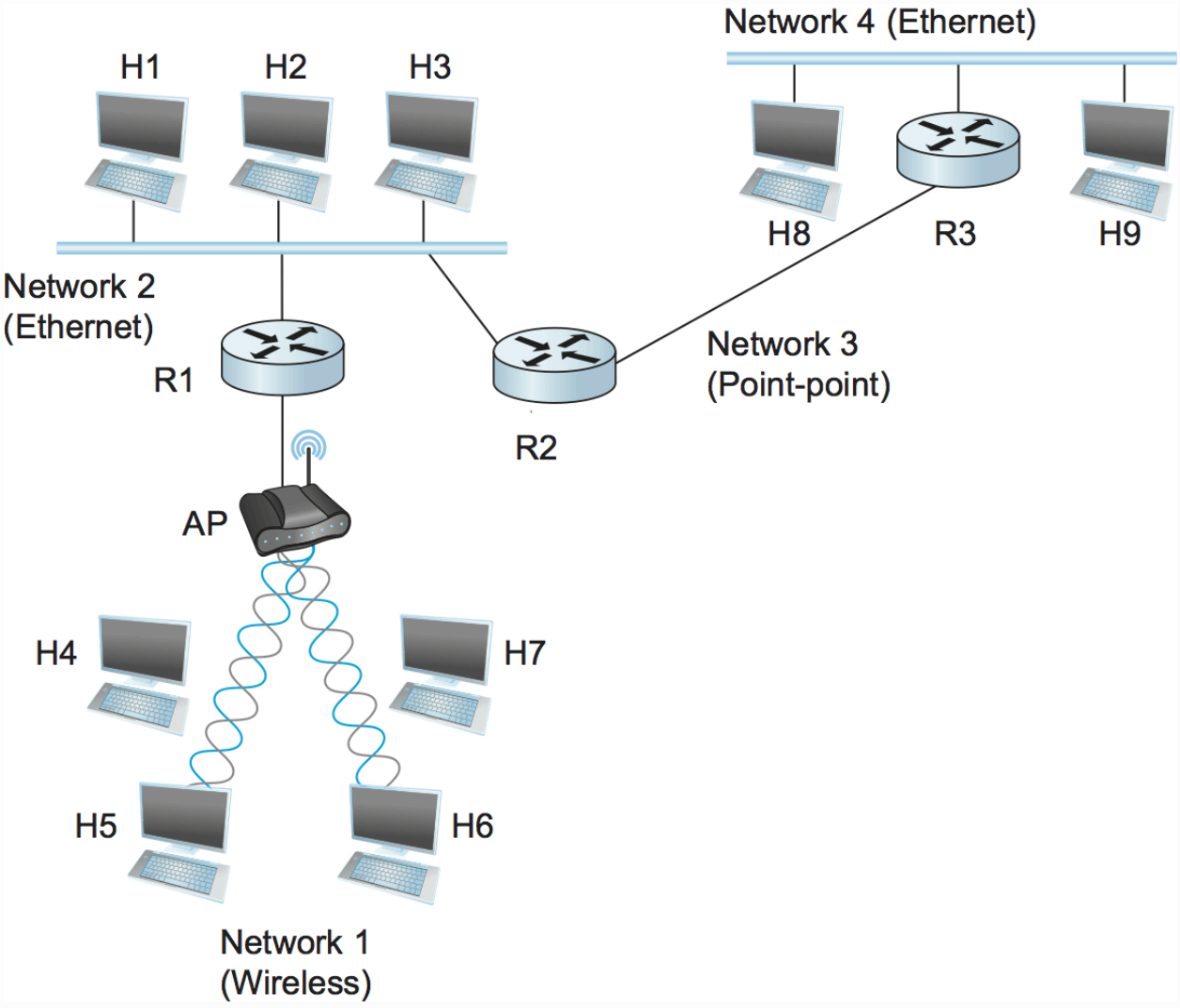


Figura 70. *Uma interconexão de redes simples. H denota um host e R denota um roteador.*

A Figura 70 mostra um exemplo de interconexão de redes. Uma interconexão de redes é frequentemente chamada de "rede de redes" porque é composta por várias redes menores. Nesta figura, vemos Ethernets, uma rede sem fio e um enlace ponto a ponto. Cada um deles é uma rede de tecnologia única. Os nós que interconectam as redes são chamados de *roteadores* . Às vezes, também são chamados de *gateways* , mas como esse termo tem várias outras conotações, restringimos nosso uso a roteador.

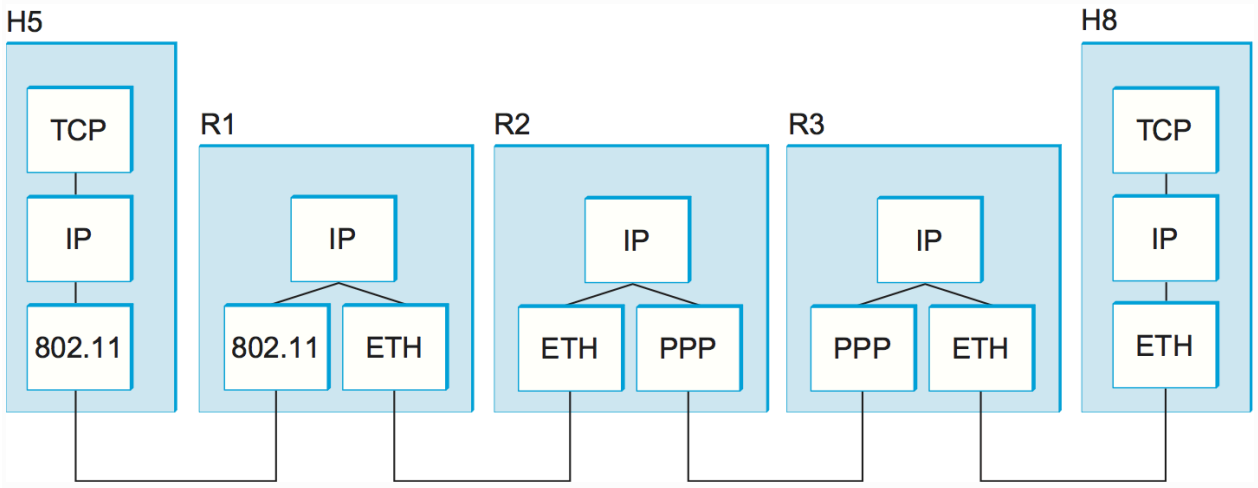


Figura 71. *Uma interconexão simples de redes, mostrando as camadas de protocolo usadas para conectar H5 a H8 na figura acima. ETH é o protocolo que roda na Ethernet.*

O *Protocolo de Internet (IP)* é a principal ferramenta usada hoje para construir redes escaláveis e heterogêneas. Originalmente, era conhecido como protocolo Kahn-Cerf, em homenagem aos seus inventores. Uma maneira de pensar em IP é que ele roda em todos os nós (hosts e roteadores) em um conjunto de redes e define a infraestrutura que permite que esses nós e redes funcionem como uma única rede lógica. Por exemplo, a Figura 71 mostra como os hosts H5 e H8 são logicamente conectados pela internet na Figura 70 , incluindo o gráfico de protocolos em execução em cada nó. Observe que protocolos de nível superior, como TCP e UDP, normalmente rodam sobre o IP nos hosts.

O restante deste e do próximo capítulo abordam vários aspectos do IP. Embora seja certamente possível construir uma interconexão de redes que não utilize IP — e, de fato, nos primórdios da internet havia soluções alternativas — o IP é o caso mais interessante para estudar simplesmente devido ao seu tamanho. Em outras palavras, apenas a internet IP realmente enfrentou o problema da escala. Portanto, ela fornece o melhor estudo de caso de um protocolo de interconexão de redes escalável.

Redes L2 vs L3

Como visto na seção anterior, uma Ethernet pode ser tratada como um *link* ponto a ponto que interconecta um par de switches, com uma malha de switches interconectados formando uma *Ethernet Comutada* . Essa configuração também é conhecida como *Rede L2* .

Mas, como descobriremos nesta seção, uma Ethernet (mesmo quando organizada em uma configuração ponto a ponto em vez de uma rede CSMA/CD compartilhada) pode ser tratada como uma *rede* que interconecta um par de roteadores, com uma malha desses roteadores formando uma internet. Essa configuração também é conhecida como *Rede L3* .

Confusamente, isso ocorre porque uma Ethernet ponto a ponto é tanto um link quanto uma rede (embora uma rede trivial de dois nós no segundo caso), dependendo se está conectada a um par de switches L2 executando o algoritmo de árvore de abrangência ou a um par de roteadores L3 executando IP (além dos protocolos de roteamento descritos posteriormente neste capítulo). Por que escolher uma configuração em vez da outra? Depende, em parte, se você deseja que a rede seja um único domínio de broadcast (em caso afirmativo, escolha L2) e se você deseja que os hosts conectados à rede estejam em redes diferentes (em caso afirmativo, selecione L3).

A boa notícia é que, quando você entender completamente as implicações dessa dualidade, terá superado um grande obstáculo no domínio das modernas redes de comutação de pacotes.

3.3.2 Modelo de Serviço

Um bom ponto de partida ao construir uma inter-rede é definir seu *modelo de serviço* , ou seja, os serviços host-to-host que você deseja fornecer. A principal preocupação ao definir um modelo de serviço para uma inter-rede é que podemos fornecer um serviço host-to-host somente se esse serviço puder, de alguma forma, ser fornecido por cada uma das redes físicas subjacentes. Por exemplo, não seria bom decidir que nosso modelo de serviço de inter-rede forneceria entrega garantida de cada pacote em 1 ms

ou menos se houvesse tecnologias de rede subjacentes que pudessem atrasar os pacotes arbitrariamente. A filosofia usada na definição do modelo de serviço IP, portanto, era torná-lo pouco exigente o suficiente para que praticamente qualquer tecnologia de rede que pudesse aparecer em uma inter-rede fosse capaz de fornecer o serviço necessário.

O modelo de serviço IP pode ser considerado como tendo duas partes: um esquema de endereçamento, que fornece uma maneira de identificar todos os hosts na inter-rede, e um modelo de entrega de dados por datagrama (sem conexão). Esse modelo de serviço às vezes é chamado de *"melhor esforço"* porque, embora o IP faça todos os esforços para entregar datagramas, não oferece garantias. Adiaremos a discussão sobre o esquema de endereçamento por enquanto e examinaremos primeiro o modelo de entrega de dados.

Entrega de datagramas

O datagrama IP é fundamental para o Protocolo de Internet. Lembre-se de uma seção anterior que um datagrama é um pacote enviado sem conexão por uma rede. Cada datagrama carrega informações suficientes para permitir que a rede encaminhe o pacote ao seu destino correto; não há necessidade de nenhum mecanismo de configuração antecipada para informar à rede o que fazer quando o pacote chega. Você simplesmente o envia, e a rede faz o possível para levá-lo ao destino desejado. A parte do "melhor esforço" significa que, se algo der errado e o pacote for perdido, corrompido, entregue incorretamente ou de alguma forma não chegar ao destino pretendido, a rede não faz nada — ela fez o possível, e isso é tudo o que precisa fazer. Ela não faz nenhuma tentativa de se recuperar da falha. Isso às vezes é chamado de serviço *não confiável* .

O serviço de melhor esforço e sem conexão é o serviço mais simples que você poderia esperar de uma interconexão de redes, e esse é seu grande ponto forte. Por exemplo, se você fornece o serviço de melhor esforço em uma rede que oferece um serviço confiável, tudo bem — você acaba com um serviço de melhor esforço que, por acaso,

sempre entrega os pacotes. Se, por outro lado, você tivesse um modelo de serviço confiável em uma rede não confiável, teria que adicionar muitas funcionalidades extras aos roteadores para compensar as deficiências da rede subjacente. Manter os roteadores o mais simples possível era um dos objetivos originais do projeto IP.

A capacidade do IP de "romper qualquer coisa" é frequentemente citada como uma de suas características mais importantes. Vale ressaltar que muitas das tecnologias sobre as quais o IP opera hoje não existiam quando o IP foi inventado. Até o momento, nenhuma tecnologia de rede inventada se mostrou tão bizarra para o IP. Em princípio, o IP pode operar em uma rede que transporta mensagens usando pombos-correio.

Entrega de melhor esforço não significa apenas que os pacotes podem se perder. Às vezes, eles podem ser entregues fora de ordem e, às vezes, o mesmo pacote pode ser entregue mais de uma vez. Os protocolos ou aplicativos de nível superior que rodam sobre IP precisam estar cientes de todos esses possíveis modos de falha.

Formato de pacote

Claramente, uma parte fundamental do modelo de serviço IP é o tipo de pacotes que podem ser transportados. O datagrama IP, como a maioria dos pacotes, consiste em um cabeçalho seguido por um número de bytes de dados. O formato do cabeçalho é mostrado na Figura 72. Observe que adotamos um estilo diferente de representação de pacotes do que o usado nos capítulos anteriores. Isso ocorre porque os formatos de pacotes na camada de interconexão de redes e acima, onde concentraremos nossa atenção nos próximos capítulos, são quase invariavelmente projetados para se alinharem em limites de 32 bits para simplificar a tarefa de processá-los em software. Assim, a maneira comum de representá-los (usada em Solicitações de Comentários da Internet, por exemplo) é desenhá-los como uma sucessão de palavras de 32 bits. A palavra do topo é a transmitida primeiro, e o byte mais à esquerda de cada palavra é o transmitido primeiro. Nessa representação, você pode reconhecer facilmente campos que são múltiplos de 8 bits de comprimento. Nas raras ocasiões em que os campos

não são múltiplos pares de 8 bits, você pode determinar os comprimentos dos campos observando as posições dos bits marcadas no topo do pacote.

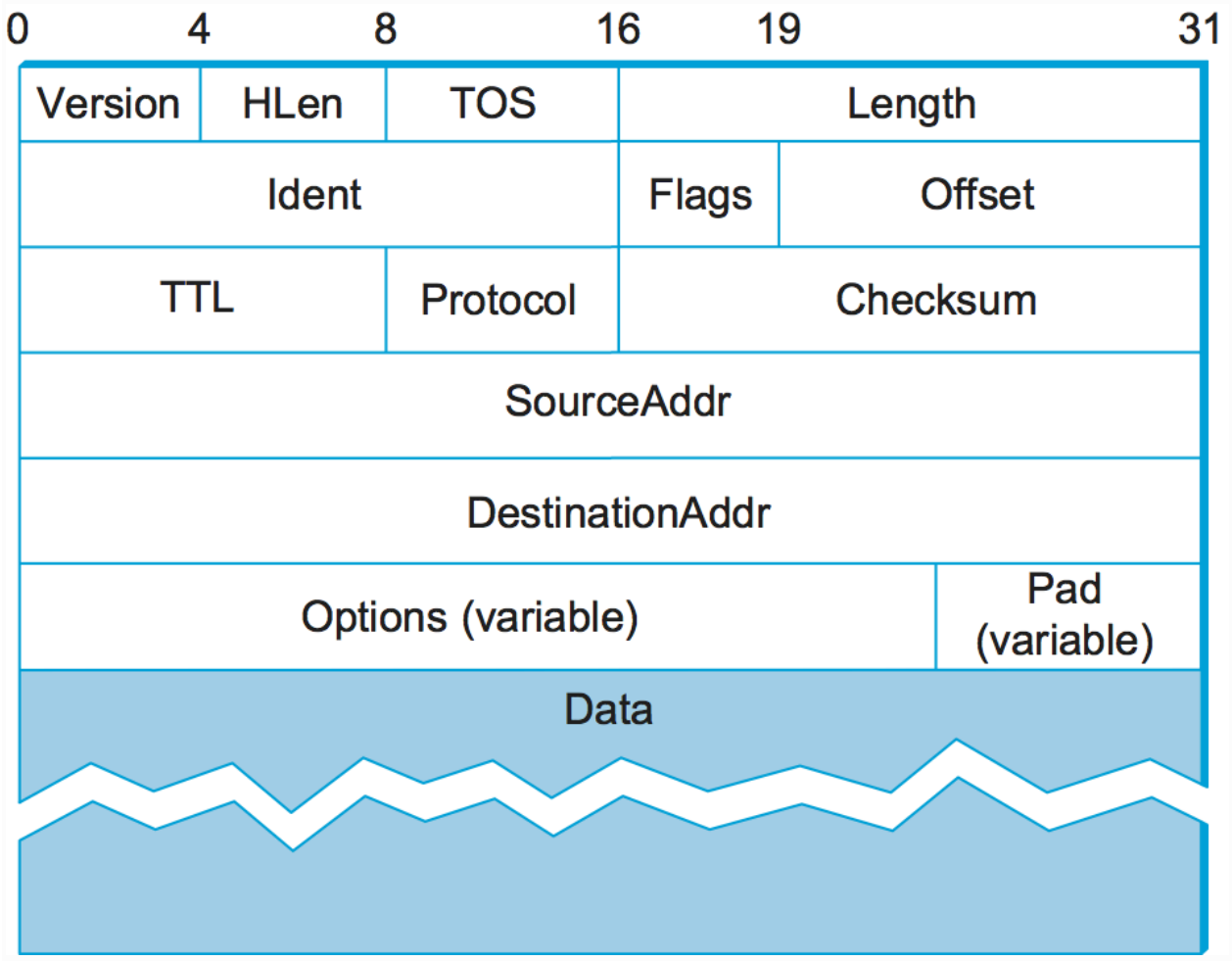


Figura 72. *Cabeçalho do pacote IPv4.*

Observando cada campo no cabeçalho IP, vemos que o modelo "simples" de entrega de datagramas de melhor esforço ainda possui algumas características sutis. O `Version`campo especifica a versão do IP. A versão ainda assumida do IP é a 4, normalmente chamada de *IPv4* . Observe que colocar esse campo logo no início do datagrama facilita a redefinição de todo o restante do formato do pacote em versões subsequentes; o software de processamento do cabeçalho começa analisando a versão e, em seguida, ramifica-se para processar o restante do pacote de acordo com o formato apropriado. O próximo campo, `HLen`, especifica o comprimento do cabeçalho em palavras de 32 bits. Quando não há opções, o que ocorre na maioria das vezes, o

cabeçalho tem 5 palavras (20 bytes). O `TOS`campo de 8 bits (tipo de serviço) teve diversas definições diferentes ao longo dos anos, mas sua função básica é permitir que os pacotes sejam tratados de forma diferente com base nas necessidades da aplicação. Por exemplo, o `TOS`valor pode determinar se um pacote deve ou não ser colocado em uma fila especial que recebe baixo atraso.

Os próximos 16 bits do cabeçalho contêm a estrutura `Length`do datagrama, incluindo o cabeçalho. Ao contrário do `HLen`campo, `Length`este conta bytes em vez de palavras. Portanto, o tamanho máximo de um datagrama IP é de 65.535 bytes. A rede física sobre a qual o IP está sendo executado, no entanto, pode não suportar pacotes tão longos. Por esse motivo, o IP suporta um processo de fragmentação e remontagem. A segunda palavra do cabeçalho contém informações sobre fragmentação, e os detalhes de seu uso são apresentados na seção seguinte, intitulada "Fragmentação e Remontagem".

Passando para a terceira palavra do cabeçalho, o próximo byte é o `TTL` campo (tempo de vida). Seu nome reflete seu significado histórico, e não a forma como é comumente usado hoje. A intenção do campo é capturar pacotes que circulam em loops de roteamento e descartá-los, em vez de deixá-los consumir recursos indefinidamente. Originalmente, "time to live" `TTL`era definido como um número específico de segundos que o pacote teria permissão de vida, e os roteadores ao longo do caminho decrementavam esse campo até que ele atingisse 0. No entanto, como era raro um pacote permanecer por até 1 segundo em um roteador, e os roteadores não tinham acesso a um relógio comum, a maioria dos roteadores apenas decrementava "time to live" `TTL`em 1 ao encaminhar o pacote. Assim, tornou-se mais uma contagem de saltos do que um temporizador, o que ainda é uma maneira perfeitamente adequada de capturar pacotes presos em loops de roteamento. Uma sutileza está na configuração inicial desse campo pelo host remetente: se definido como muito alto, os pacotes poderiam circular bastante antes de serem descartados; se definido como muito baixo, eles poderiam não chegar ao destino. O valor 64 é o padrão atual.

O `Protocol`campo é simplesmente uma chave de demultiplexação que identifica o protocolo de nível superior para o qual este pacote IP deve ser passado. Há valores definidos para TCP (Protocolo de Controle de Transmissão — 6), UDP (Protocolo de Datagrama de Usuário — 17) e muitos outros protocolos que podem estar acima do IP no gráfico de protocolos.

O `Checksum`é calculado considerando todo o cabeçalho IP como uma sequência de palavras de 16 bits, somando-as usando aritmética de complemento para um e tomando o complemento para um do resultado. Assim, se algum bit no cabeçalho for corrompido durante o trânsito, a soma de verificação não conterá o valor correto no recebimento do pacote. Como um cabeçalho corrompido pode conter um erro no endereço de destino — e, como resultado, pode ter sido entregue incorretamente —, faz sentido descartar qualquer pacote que falhe na soma de verificação. Deve-se observar que esse tipo de soma de verificação não possui as mesmas propriedades robustas de detecção de erros que um CRC, mas é muito mais fácil de calcular em software.

Os dois últimos campos obrigatórios no cabeçalho são o `SourceAddr`e o `DestinationAddr`para o pacote. Este último é a chave para a entrega do datagrama: cada pacote contém um endereço completo para o seu destino pretendido, para que as decisões de encaminhamento possam ser tomadas em cada roteador. O endereço de origem é necessário para permitir que os destinatários decidam se desejam aceitar o pacote e para que possam responder. Os endereços IP serão discutidos em uma seção posterior — por enquanto, o importante é saber que o IP define seu próprio espaço de endereçamento global, independentemente de quaisquer redes físicas sobre as quais ele seja executado. Como veremos, esta é uma das chaves para sustentar a heterogeneidade.

Por fim, pode haver uma série de opções no final do cabeçalho. A presença ou ausência de opções pode ser determinada examinando o `HLen`campo de comprimento

do cabeçalho ( ). Embora as opções sejam usadas raramente, uma implementação IP completa deve lidar com todas elas.

Fragmentação e Remontagem

Um dos problemas de fornecer um modelo de serviço host-a-host uniforme em um conjunto heterogêneo de redes é que cada tecnologia de rede tende a ter sua própria ideia do tamanho de um pacote. Por exemplo, a Ethernet clássica pode aceitar pacotes de até 1.500 bytes, mas as variantes modernas podem entregar pacotes maiores (jumbo) que transportam até 9.000 bytes de carga útil. Isso deixa duas opções para o modelo de serviço IP: garantir que todos os datagramas IP sejam pequenos o suficiente para caber em um pacote em qualquer tecnologia de rede ou fornecer um meio pelo qual os pacotes possam ser fragmentados e remontados quando forem grandes demais para passar por uma determinada tecnologia de rede. Esta última opção acaba sendo uma boa opção, especialmente quando se considera o fato de que novas tecnologias de rede estão sempre surgindo e o IP precisa ser executado em todas elas; isso dificultaria a escolha de um limite pequeno adequado para o tamanho do datagrama. Isso também significa que um host não enviará pacotes desnecessariamente pequenos, o que desperdiça largura de banda e consome recursos de processamento, exigindo mais cabeçalhos por byte de dados enviado.

A ideia central aqui é que cada tipo de rede tem uma *unidade máxima de transmissão* (MTU), que é o maior datagrama IP que ela pode transportar em um quadro. 1 Observe que esse valor é menor que o maior tamanho de pacote nessa rede porque o datagrama IP precisa caber na *carga útil* do quadro da camada de enlace.

1

Em redes ATM, a MTU é, felizmente, muito maior do que uma única célula, pois o ATM possui seu próprio mecanismo de fragmentação e remontagem. O quadro da camada de enlace no ATM é chamado de *unidade de dados de protocolo de subcamada de convergência* (CS-PDU).

Portanto, quando um host envia um datagrama IP, ele pode escolher o tamanho que desejar. Uma escolha razoável é a MTU da rede à qual o host está diretamente conectado. Nesse caso, a fragmentação só será necessária se o caminho para o destino incluir uma rede com uma MTU menor. No entanto, se o protocolo de transporte que se baseia no IP fornecer ao IP um pacote maior que a MTU local, o host de origem deverá fragmentá-lo.

A fragmentação normalmente ocorre em um roteador quando ele recebe um datagrama que deseja encaminhar por uma rede com uma MTU menor que a do datagrama recebido. Para permitir que esses fragmentos sejam remontados no host receptor, todos eles carregam o mesmo identificador no `Ident`campo. Esse identificador é escolhido pelo host remetente e deve ser único entre todos os datagramas que podem chegar ao destino a partir dessa origem em um período de tempo razoável. Como todos os fragmentos do datagrama original contêm esse identificador, o host que os remonta será capaz de reconhecer os fragmentos que estão juntos. Caso todos os fragmentos não cheguem ao host receptor, o host desiste do processo de remontagem e descarta os fragmentos que chegaram. O IP não tenta se recuperar de fragmentos ausentes.

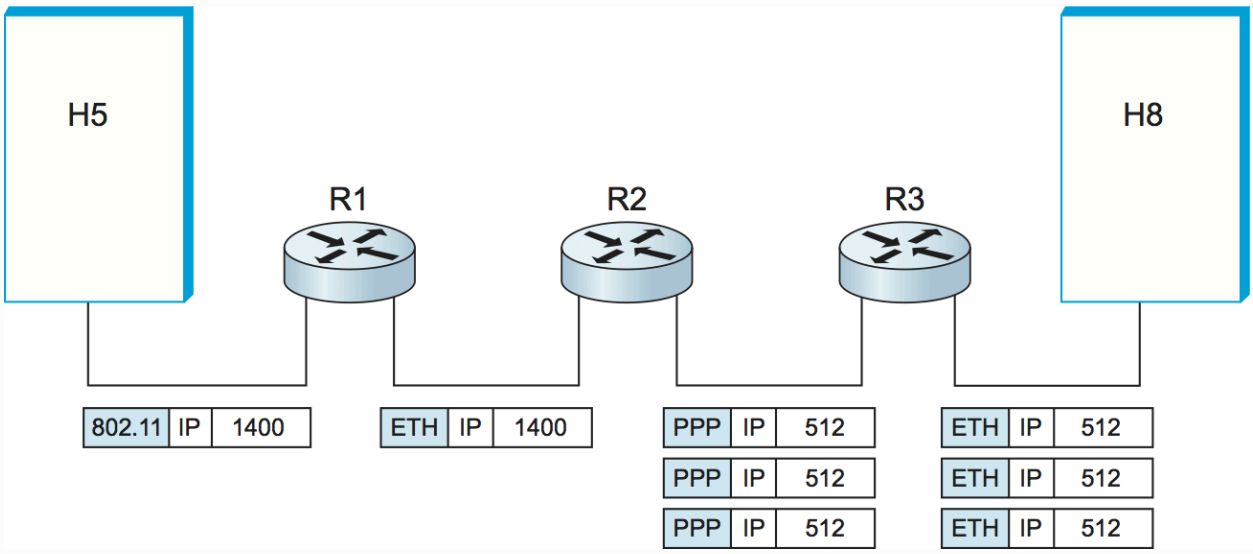
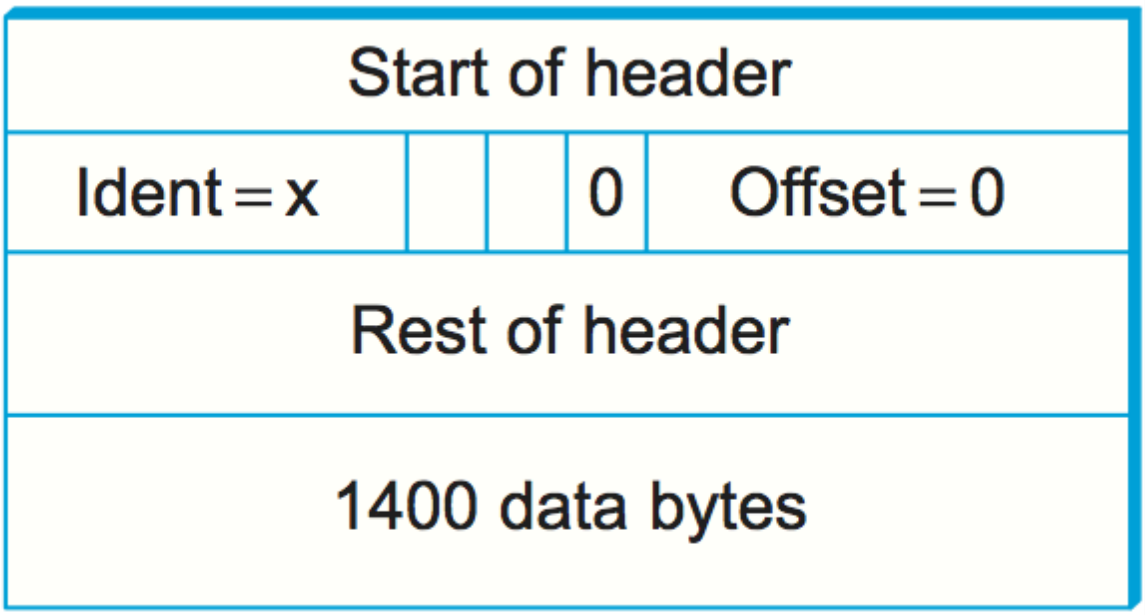


Figura 73. *Datagramas IP percorrendo a sequência de redes físicas representadas na figura anterior.*

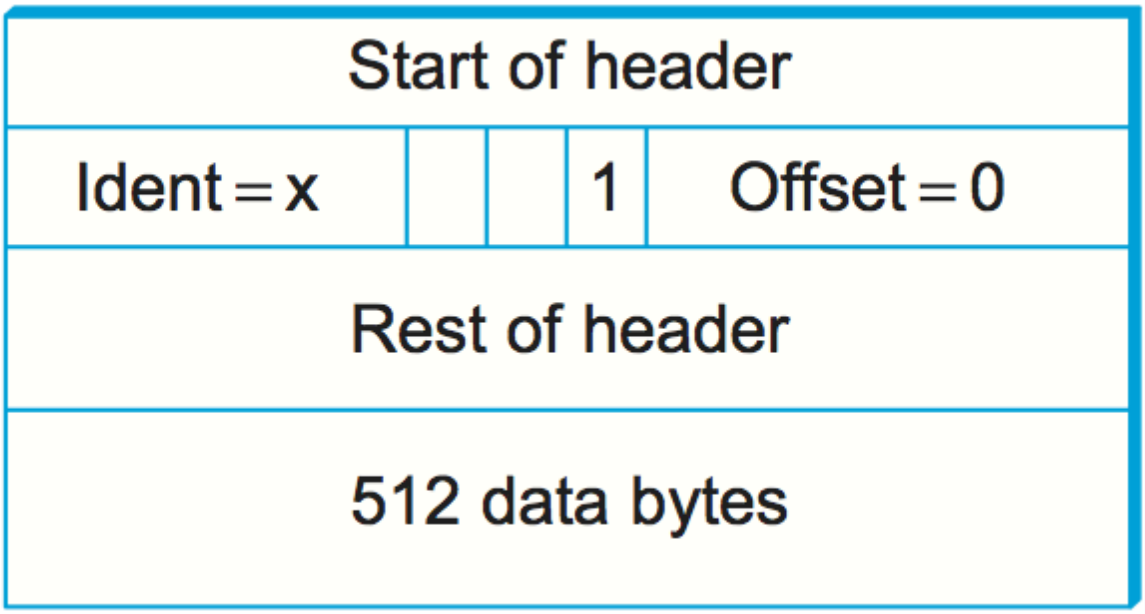
Para entender o que tudo isso significa, considere o que acontece quando o host H5 envia um datagrama para o host H8 no exemplo de internet mostrado na Figura 70. Supondo que a MTU seja de 1500 bytes para as duas Ethernets e a rede 802.11, e 532 bytes para a rede ponto a ponto, então um datagrama de 1420 bytes (cabeçalho IP de 20 bytes mais 1400 bytes de dados) enviado do H5 atravessa a rede 802.11 e a primeira Ethernet sem fragmentação, mas deve ser fragmentado em três datagramas no roteador R2. Esses três fragmentos são então encaminhados pelo roteador R3 através da segunda Ethernet para o host de destino. Essa situação é ilustrada na Figura 73. Esta figura também serve para reforçar dois pontos importantes:

1. Cada fragmento é um datagrama IP independente que é transmitido por uma sequência de redes físicas, independente dos outros fragmentos.
2. Cada datagrama IP é reencapsulado para cada rede física pela qual ele trafega.

(a)


Start of header
Ident = x
0
Offset = 0
Rest of header
1400 data bytes

(b)


Start of header
Ident = x
1
Offset = 0
Rest of header
512 data bytes

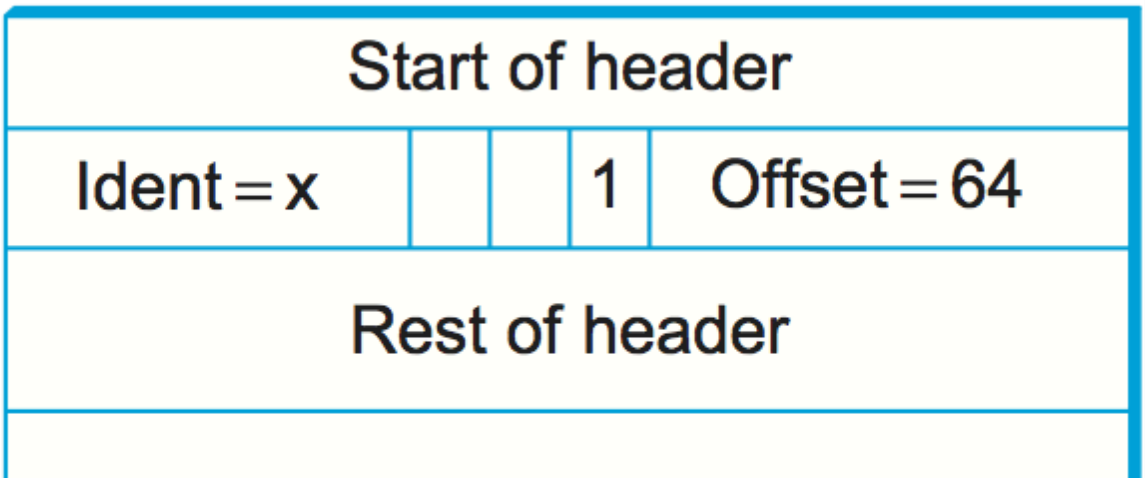

Start of header
Ident = x
1
Offset = 64
Rest of header

Figura 74. *Campos de cabeçalho usados na fragmentação de IP: (a) pacote não fragmentado; (b) pacotes fragmentados.*

O processo de fragmentação pode ser compreendido em detalhes observando os campos de cabeçalho de cada datagrama, como é feito na Figura 74. O pacote não fragmentado, mostrado na parte superior, tem 1400 bytes de dados e um cabeçalho IP de 20 bytes. Quando o pacote chega ao roteador R2, que tem uma MTU de 532 bytes, ele precisa ser fragmentado. Uma MTU de 532 bytes deixa 512 bytes para dados após o cabeçalho IP de 20 bytes, então o primeiro fragmento contém 512 bytes de dados. O roteador define o bit M no `Flags`campo (veja a Figura 72), significando que há mais fragmentos a seguir, e define o `Offset`como 0, já que este fragmento contém a primeira parte do datagrama original. Os dados transportados no segundo fragmento começam com o 513º byte dos dados originais, então o `Offset`campo neste cabeçalho é definido como 64, que é 512/8. Por que a divisão por 8? Porque os projetistas do IP decidiram que a fragmentação deveria sempre ocorrer em limites de 8 bytes, o que significa que o `Offset`campo conta blocos de 8 bytes, não bytes. (Deixamos como exercício para você descobrir por que essa decisão de projeto foi tomada.) O terceiro fragmento contém os últimos 376 bytes de dados, e o deslocamento agora é 2 × 512/8 = 128. Como este é o último fragmento, o bit M não é definido.

Observe que o processo de fragmentação é feito de tal forma que poderia ser repetido caso um fragmento chegasse a outra rede com uma MTU ainda menor. A fragmentação produz datagramas IP menores e válidos que podem ser facilmente remontados no datagrama original após o recebimento, independentemente da ordem de chegada. A remontagem é feita no host receptor e não em cada roteador.

A remontagem de IPs está longe de ser um processo simples. Por exemplo, se um único fragmento for perdido, o receptor ainda tentará remontar o datagrama e, eventualmente, desistirá e terá que coletar os recursos que foram usados para realizar a remontagem malsucedida. Fazer com que um host ocupe recursos desnecessariamente pode ser a base de um ataque de negação de serviço.

Por esse motivo, entre outros, a fragmentação de IP é geralmente considerada algo positivo a ser evitado. Os hosts agora são fortemente incentivados a realizar a "descoberta de MTU de caminho", um processo pelo qual a fragmentação é evitada enviando pacotes pequenos o suficiente para atravessar o link com a menor MTU no caminho, do remetente ao destinatário.

3.3.3 Endereços globais

Na discussão acima sobre o modelo de serviço IP, mencionamos que um dos elementos que ele fornece é um esquema de endereçamento. Afinal, se você deseja enviar dados para qualquer host em qualquer rede, é necessário que haja uma maneira de identificar todos os hosts. Portanto, precisamos de um esquema de endereçamento global — um em que não haja dois hosts com o mesmo endereço. A exclusividade global é a primeira propriedade que deve ser fornecida em um esquema de endereçamento.

Endereços Ethernet são globalmente únicos, mas isso por si só não é suficiente para um esquema de endereçamento em uma grande inter-rede. Endereços Ethernet também são *planos* , o que significa que eles não têm estrutura e fornecem muito poucas pistas para protocolos de roteamento. (Na verdade, endereços Ethernet têm uma estrutura para fins de *atribuição* — os primeiros 24 bits identificam o fabricante — mas isso não fornece nenhuma informação útil para protocolos de roteamento, já que essa estrutura não tem nada a ver com a topologia da rede.) Em contraste, endereços IP são *hierárquicos* , o que significa que eles são compostos de várias partes que correspondem a algum tipo de hierarquia na inter-rede. Especificamente, endereços IP consistem em duas partes, geralmente chamadas de parte *de rede* e parte de *host* . Esta é uma estrutura bastante lógica para uma inter-rede, que é composta de muitas redes interconectadas. A parte de rede de um endereço IP identifica a rede à qual o host está conectado; todos os hosts conectados à mesma rede têm a mesma parte de rede em seu endereço IP. A parte de host então identifica cada host exclusivamente naquela rede específica. Assim, na inter-rede simples da Figura 70 , os endereços dos

hosts na rede 1, por exemplo, teriam todos a mesma parte de rede e partes de host diferentes.

Observe que os roteadores na Figura 70 estão conectados a duas redes. Eles precisam ter um endereço em cada rede, um para cada interface. Por exemplo, o roteador R1, que fica entre a rede sem fio e uma Ethernet, tem um endereço IP na interface para a rede sem fio cuja parte da rede é a mesma que todos os hosts nessa rede. Ele também tem um endereço IP na interface para a Ethernet que tem a mesma parte da rede que os hosts nessa Ethernet. Portanto, tendo em mente que um roteador pode ser implementado como um host com duas interfaces de rede, é mais preciso pensar em endereços IP como pertencentes a interfaces do que a hosts.

Agora, como são esses endereços hierárquicos? Ao contrário de outras formas de endereço hierárquico, os tamanhos das duas partes não são os mesmos para todos os endereços. Originalmente, os endereços IP eram divididos em três classes diferentes, como mostrado na Figura 75 , cada uma definindo partes de rede e host de tamanhos diferentes. (Há também endereços de classe D que especificam um grupo multicast e endereços de classe E que atualmente não são utilizados.) Em todos os casos, o endereço tem 32 bits.

A classe de um endereço IP é identificada nos poucos bits mais significativos. Se o primeiro bit for 0, é um endereço de classe A. Se o primeiro bit for 1 e o segundo for 0, é um endereço de classe B. Se os dois primeiros bits forem 1 e o terceiro for 0, é um endereço de classe C. Assim, dos aproximadamente 4 bilhões de endereços IP possíveis, metade são de classe A, um quarto são de classe B e um oitavo são de classe C. Cada classe aloca um certo número de bits para a parte de rede do endereço e o restante para a parte do host. Redes de classe A têm 7 bits para a parte de rede e 24 bits para a parte do host, o que significa que pode haver apenas 126 redes de classe A (os valores 0 e 127 são reservados), mas cada uma delas pode acomodar até

(cerca de 16 milhões) de hosts (novamente, há dois valores reservados). Endereços de classe B alocam 14 bits para a rede e 16 bits para o host, o que significa que cada rede de classe B tem espaço para 65.534 hosts. Por fim, endereços de classe C têm apenas 8 bits para o host e 21 para a parte de rede. Portanto, uma rede de classe C pode ter apenas 256 identificadores de host exclusivos, o que significa apenas 254 hosts conectados (um identificador de host, 255, é reservado para transmissão, e 0 não é um número de host válido). No entanto, o esquema de endereçamento suporta 2^{21} redes de classe C.

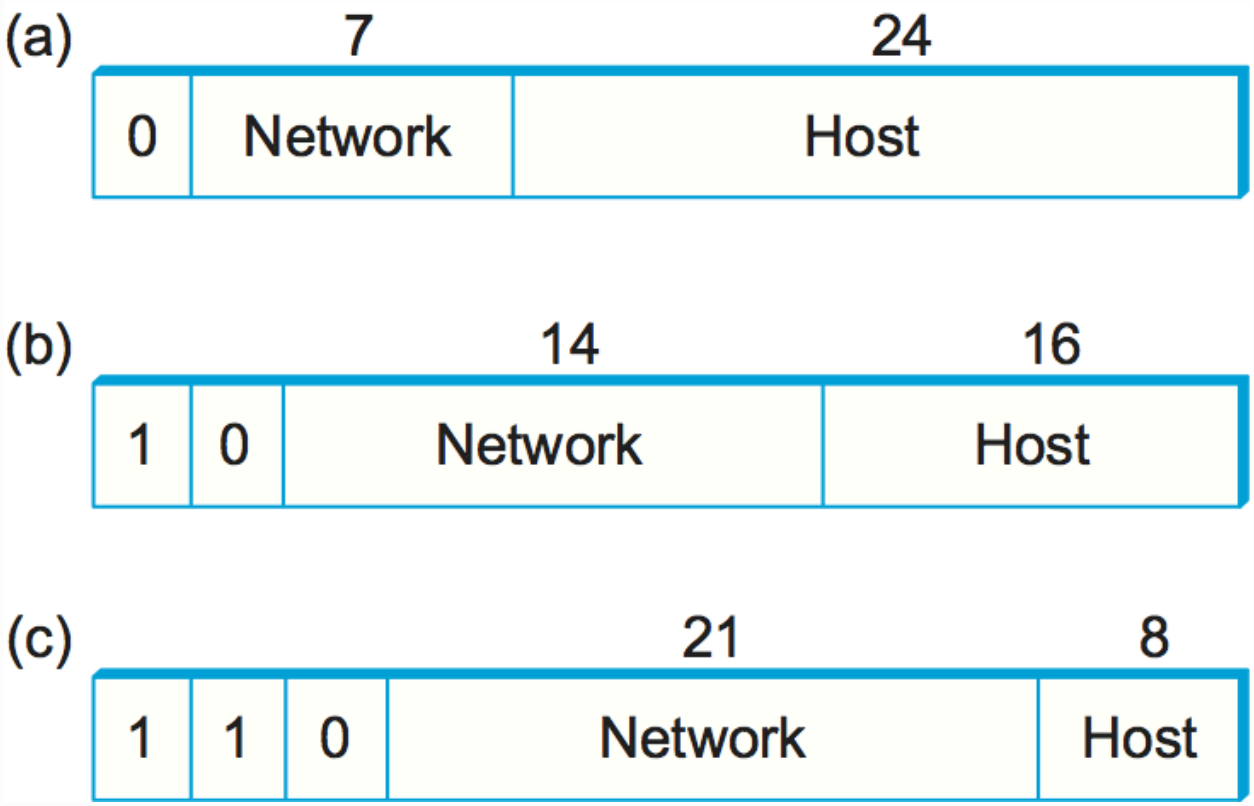


Figura 75. *Endereços IP: (a) classe A; (b) classe B; (c) classe C.*

À primeira vista, esse esquema de endereçamento apresenta bastante flexibilidade, permitindo que redes de tamanhos muito diferentes sejam acomodadas com bastante eficiência. A ideia original era que a Internet consistisse em um pequeno número de redes de longa distância (seriam redes de classe A), um número modesto de redes do tamanho de um campus (seriam redes de classe B) e um grande número de LANs (seriam redes de classe C). No entanto, ele se mostrou insuficientemente flexível, como

veremos em breve. Hoje, os endereços IP são normalmente "sem classes"; os detalhes disso são explicados a seguir.

Antes de analisarmos como os endereços IP são usados, é útil analisar algumas questões práticas, como a forma como você os escreve. Por convenção, os endereços IP são escritos como quatro números inteiros *decimais* separados por pontos. Cada número inteiro representa o valor decimal contido em 1 byte do endereço, começando pelo mais significativo. Por exemplo, o endereço do computador no qual esta frase foi digitada é `171.69.210.245`.

É importante não confundir endereços IP com nomes de domínio da Internet, que também são hierárquicos. Nomes de domínio tendem a ser strings ASCII separadas por pontos, como `cs.princeton.edu`. O importante sobre endereços IP é que eles são o que é transportado nos cabeçalhos dos pacotes IP, e são esses endereços que são usados nos roteadores IP para tomar decisões de encaminhamento.

3.3.4 Encaminhamento de datagramas em IP

Agora estamos prontos para analisar o mecanismo básico pelo qual roteadores IP encaminham datagramas em uma inter-rede. Lembre-se de uma seção anterior: o *encaminhamento* é o processo de pegar um pacote de uma entrada e enviá-lo na saída apropriada, enquanto *o roteamento* é o processo de construir as tabelas que permitem determinar a saída correta de um pacote. A discussão aqui se concentra no encaminhamento; abordaremos o roteamento em uma seção posterior.

Os principais pontos a ter em mente ao discutir o encaminhamento de datagramas IP são os seguintes:

- Cada datagrama IP contém o endereço IP do host de destino.
- A parte de rede de um endereço IP identifica exclusivamente uma única rede física que faz parte da Internet maior.

- Todos os hosts e roteadores que compartilham a mesma parte da rede de seus endereços estão conectados à mesma rede física e podem, portanto, se comunicar entre si enviando quadros por essa rede.
- Toda rede física que faz parte da Internet tem pelo menos um roteador que, por definição, também está conectado a pelo menos uma outra rede física; esse roteador pode trocar pacotes com hosts ou roteadores em qualquer uma das redes.

O encaminhamento de datagramas IP pode, portanto, ser tratado da seguinte maneira. Um datagrama é enviado de um host de origem para um host de destino, possivelmente passando por vários roteadores ao longo do caminho. Qualquer nó, seja um host ou um roteador, primeiro tenta estabelecer se está conectado à mesma rede física que o destino. Para fazer isso, ele compara a parte da rede do endereço de destino com a parte da rede do endereço de cada uma de suas interfaces de rede. (Os hosts normalmente têm apenas uma interface, enquanto os roteadores normalmente têm duas ou mais, já que normalmente estão conectados a duas ou mais redes.) Se ocorrer uma correspondência, isso significa que o destino está na mesma rede física que a interface, e o pacote pode ser entregue diretamente por essa rede. Uma seção posterior explica alguns detalhes desse processo.

Se o nó não estiver conectado à mesma rede física que o nó de destino, ele precisará enviar o datagrama para um roteador. Em geral, cada nó terá a opção de escolher entre vários roteadores e, portanto, precisará escolher o melhor, ou pelo menos um que tenha uma chance razoável de levar o datagrama para mais perto de seu destino. O roteador escolhido é conhecido como roteador do *próximo salto* . O roteador encontra o próximo salto correto consultando sua tabela de encaminhamento. A tabela de encaminhamento é conceitualmente apenas uma lista de pares. (Como veremos a seguir, as tabelas de encaminhamento, na prática, geralmente contêm algumas informações adicionais relacionadas ao próximo salto.) Normalmente, há também um roteador padrão que é usado se nenhuma das entradas na tabela corresponder ao número da rede de destino. Para um host, pode ser bastante aceitável ter um roteador

padrão e nada mais — isso significa que todos os datagramas destinados a hosts que não estejam na rede física à qual o host remetente está conectado serão enviados pelo roteador padrão. `(NetworkNum, NextHop)`

Podemos descrever o algoritmo de encaminhamento de datagramas da seguinte maneira:

```
if (NetworkNum of destination = NetworkNum of one of my interfaces) then
    deliver packet to destination over that interface
else
    if (NetworkNum of destination is in my forwarding table) then
        deliver packet to NextHop router
    else
        deliver packet to default router
```

Para um host com apenas uma interface e apenas um roteador padrão em sua tabela de encaminhamento, isso simplifica para

```
if (NetworkNum of destination = my NetworkNum) then
    deliver packet to destination directly
else
    deliver packet to default router
```

Vejamos como isso funciona no exemplo de interconexão de redes da Figura 70. Primeiro, suponha que H1 queira enviar um datagrama para H2. Como estão na mesma rede física, H1 e H2 têm o mesmo número de rede em seus endereços IP. Assim, H1 deduz que pode entregar o datagrama diretamente para H2 pela Ethernet. A única questão que precisa ser resolvida é como H1 descobre o endereço Ethernet correto para H2 — o mecanismo de resolução descrito em uma seção posterior aborda essa questão.

Agora, suponha que H5 queira enviar um datagrama para H8. Como esses hosts estão em redes físicas diferentes, eles têm números de rede diferentes, então H5 deduz que precisa enviar o datagrama para um roteador. R1 é a única opção — o roteador padrão — então H1 envia o datagrama pela rede sem fio para R1. Da mesma forma, R1 sabe

que não pode entregar um datagrama diretamente para H8 porque nenhuma das interfaces de R1 está na mesma rede que H8. Suponha que o roteador padrão de R1 seja R2; R1 então envia o datagrama para R2 pela Ethernet. Supondo que R2 tenha a tabela de encaminhamento mostrada na Tabela 10 , ele procura o número de rede de H8 (rede 4) e encaminha o datagrama pela rede ponto a ponto para R3. Finalmente, R3, como está na mesma rede que H8, encaminha o datagrama diretamente para H8.

Número da rede	Próximo salto
1	R1
4	R3

Observe que é possível incluir informações sobre redes conectadas diretamente na tabela de encaminhamento. Por exemplo, poderíamos rotular as interfaces de rede do roteador R2 como interface 0 para o link ponto a ponto (rede 3) e interface 1 para a Ethernet (rede 2). Assim, R2 teria a tabela de encaminhamento mostrada na Tabela 11 .

Número da rede	Próximo salto
1	R1

2	Interface 1
3	Interface 0
4	R3

Assim, para qualquer número de rede que o R2 encontre em um pacote, ele sabe o que fazer. Ou essa rede está diretamente conectada ao R2, caso em que o pacote pode ser entregue ao seu destino por essa rede, ou a rede pode ser acessada por meio de algum roteador de próximo salto que o R2 pode alcançar por meio de uma rede à qual está conectado. Em ambos os casos, o R2 usará o ARP, descrito abaixo, para encontrar o endereço MAC do nó para o qual o pacote será enviado em seguida.

A tabela de encaminhamento usada pelo R2 é simples o suficiente para ser configurada manualmente. Normalmente, porém, essas tabelas são mais complexas e seriam construídas executando um protocolo de roteamento como um dos descritos em uma seção posterior. Observe também que, na prática, os números de rede costumam ser maiores (por exemplo, 128,96).

Agora podemos ver como o endereçamento hierárquico — a divisão do endereço em partes de rede e de host — melhorou a escalabilidade de uma rede grande. Os roteadores agora contêm tabelas de encaminhamento que listam apenas um conjunto de números de rede, em vez de todos os nós da rede. Em nosso exemplo simples, isso significava que o R2 podia armazenar as informações necessárias para alcançar todos os hosts da rede (dos quais havia oito) em uma tabela de quatro entradas. Mesmo que houvesse 100 hosts em cada rede física, o R2 ainda precisaria apenas dessas mesmas quatro entradas. Este é um bom primeiro passo (embora de forma alguma o último) para alcançar a escalabilidade.

Conclusão principal

Isso ilustra um dos princípios mais importantes da construção de redes escaláveis: para alcançar a escalabilidade, é necessário reduzir a quantidade de informações armazenadas em cada nó e trocadas entre eles. A maneira mais comum de fazer isso é *a agregação hierárquica* . O IP introduz uma hierarquia de dois níveis, com as redes no nível superior e os nós no nível inferior. Agregamos informações permitindo que os roteadores se preocupem apenas em alcançar a rede correta; as informações de que um roteador precisa para entregar um datagrama a qualquer nó em uma determinada rede são representadas por uma única informação agregada. [Próximo]

3.3.5 Sub-redes e endereçamento sem classes

A intenção original dos endereços IP era que a parte de rede identificasse exclusivamente uma rede física. Acontece que essa abordagem tem algumas desvantagens. Imagine um grande campus com muitas redes internas e que decide se conectar à internet. Para cada rede, não importa quão pequena, o local precisa de pelo menos um endereço de rede classe C. Pior ainda, para qualquer rede com mais de 255 hosts, é necessário um endereço classe B. Isso pode não parecer grande coisa, e de fato não era quando a internet foi concebida pela primeira vez, mas há apenas um número finito de números de rede e muito menos endereços classe B do que classe C. Endereços classe B tendem a ter uma demanda particularmente alta porque você nunca sabe se sua rede pode expandir além de 255 nós, então é mais fácil usar um endereço classe B desde o início do que ter que renumerar todos os hosts quando você ficar sem espaço em uma rede classe C. O problema que observamos aqui é a ineficiência na atribuição de endereços: uma rede com dois nós usa um endereço de rede classe C inteiro, desperdiçando 253 endereços perfeitamente úteis; uma rede de classe B com pouco mais de 255 hosts desperdiça mais de 64.000 endereços.

Atribuir um número de rede por rede física, portanto, consome o espaço de endereços IP potencialmente muito mais rápido do que gostaríamos. Embora precisemos conectar

mais de 4 bilhões de hosts para usar todos os endereços válidos, precisamos conectar apenas 2^{14} (cerca de 16.000) redes classe B antes que essa parte do espaço de endereços se esgote. Portanto, gostaríamos de encontrar uma maneira de usar os números de rede de forma mais eficiente.

Atribuir muitos números de rede apresenta outra desvantagem que se torna aparente quando se pensa em roteamento. Lembre-se de que a quantidade de estado armazenado em um nó participante de um protocolo de roteamento é proporcional ao número de outros nós, e que o roteamento em uma internet consiste na construção de tabelas de encaminhamento que informam a um roteador como alcançar diferentes redes. Portanto, quanto mais números de rede estiverem em uso, maiores se tornarão as tabelas de encaminhamento. Tabelas de encaminhamento grandes adicionam custos aos roteadores e são potencialmente mais lentas para pesquisar do que tabelas menores para uma determinada tecnologia, degradando o desempenho do roteador. Isso fornece outra motivação para atribuir números de rede com cuidado.

A sub-rede fornece um primeiro passo para reduzir o número total de números de rede atribuídos. A ideia é pegar um único número de rede IP e alocar os endereços IP com esse número de rede para várias redes físicas, que agora são chamadas de *sub-redes* . Várias coisas precisam ser feitas para que isso funcione. Primeiro, as sub-redes devem estar próximas umas das outras. Isso ocorre porque, de um ponto distante na Internet, todas parecerão uma única rede, com apenas um número de rede entre elas. Isso significa que um roteador só poderá selecionar uma rota para alcançar qualquer uma das sub-redes, então é melhor que todas estejam na mesma direção geral. Uma situação perfeita para usar a sub-rede é um grande campus ou corporação que tenha muitas redes físicas. De fora do campus, tudo o que você precisa saber para alcançar qualquer sub-rede dentro do campus é onde o campus se conecta ao restante da Internet. Isso geralmente ocorre em um único ponto, então uma entrada na sua tabela de encaminhamento será suficiente. Mesmo que haja vários pontos em que o campus esteja conectado ao restante da Internet, saber como chegar a um ponto na rede do campus ainda é um bom começo.

O mecanismo pelo qual um único número de rede pode ser compartilhado entre múltiplas redes envolve a configuração de todos os nós em cada sub-rede com uma *máscara de sub-rede* . Com endereços IP simples, todos os hosts na mesma rede devem ter o mesmo número de rede. A máscara de sub-rede nos permite introduzir um *número de sub-rede* ; todos os hosts na mesma rede física terão o mesmo número de sub-rede, o que significa que os hosts podem estar em redes físicas diferentes, mas compartilhar um único número de rede. Este conceito é ilustrado na Figura 76 .

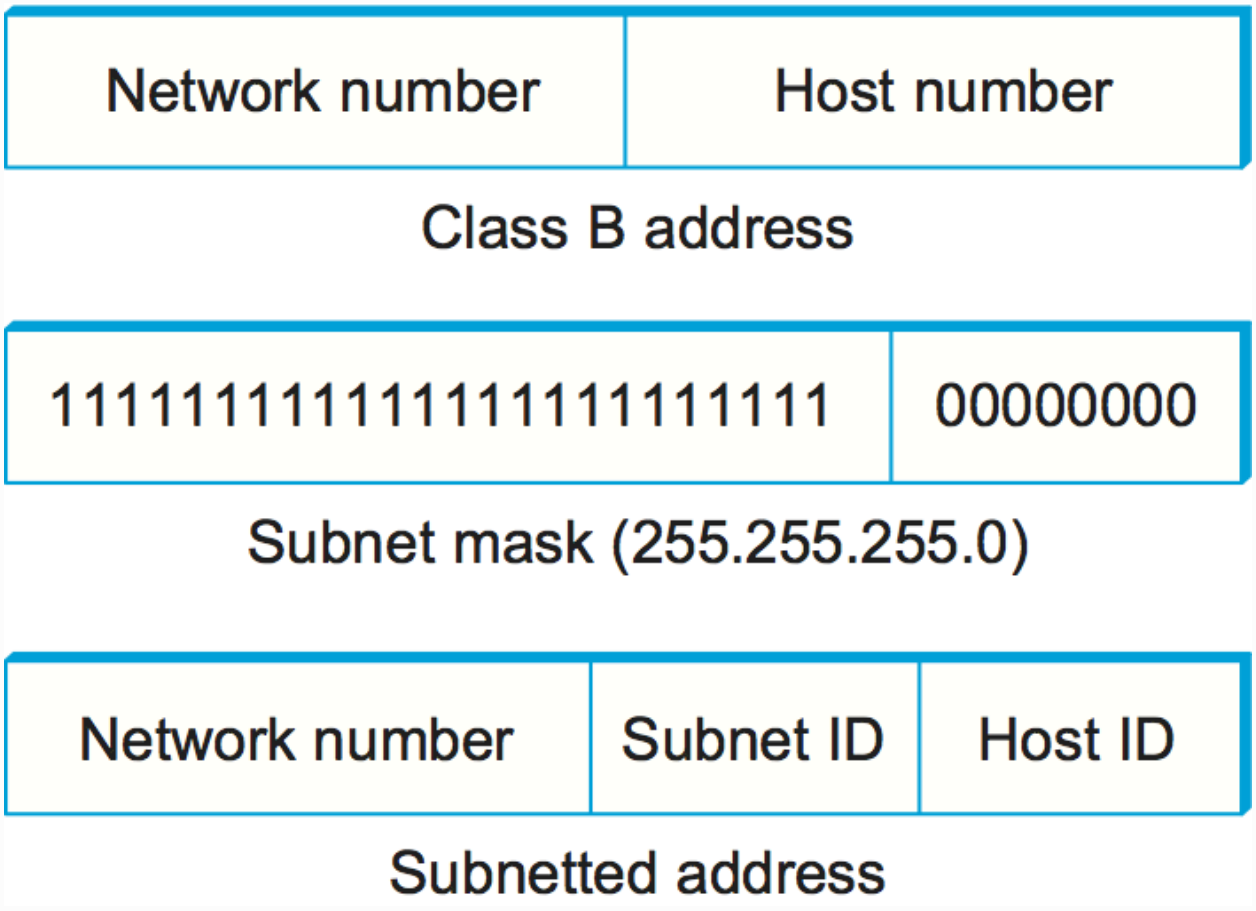


Figura 76. *Endereçamento de sub-rede.*

O que a sub-rede significa para um host é que ele agora está configurado com um endereço IP e uma máscara de sub-rede para a sub-rede à qual está conectado. Por exemplo, o host H1 na Figura 77 está configurado com um endereço de 128.96.34.15 e uma máscara de sub-rede de 255.255.255.128. (Todos os hosts em uma determinada sub-rede são configurados com a mesma máscara; ou seja, há exatamente uma

máscara de sub-rede por sub-rede.) O AND bit a bit desses dois números define o número de sub-rede do host e de todos os outros hosts na mesma sub-rede. Nesse caso, 128.96.34.15 AND 255.255.255.128 é igual a 128.96.34.0, portanto, este é o número de sub-rede para a sub-rede mais alta na figura.

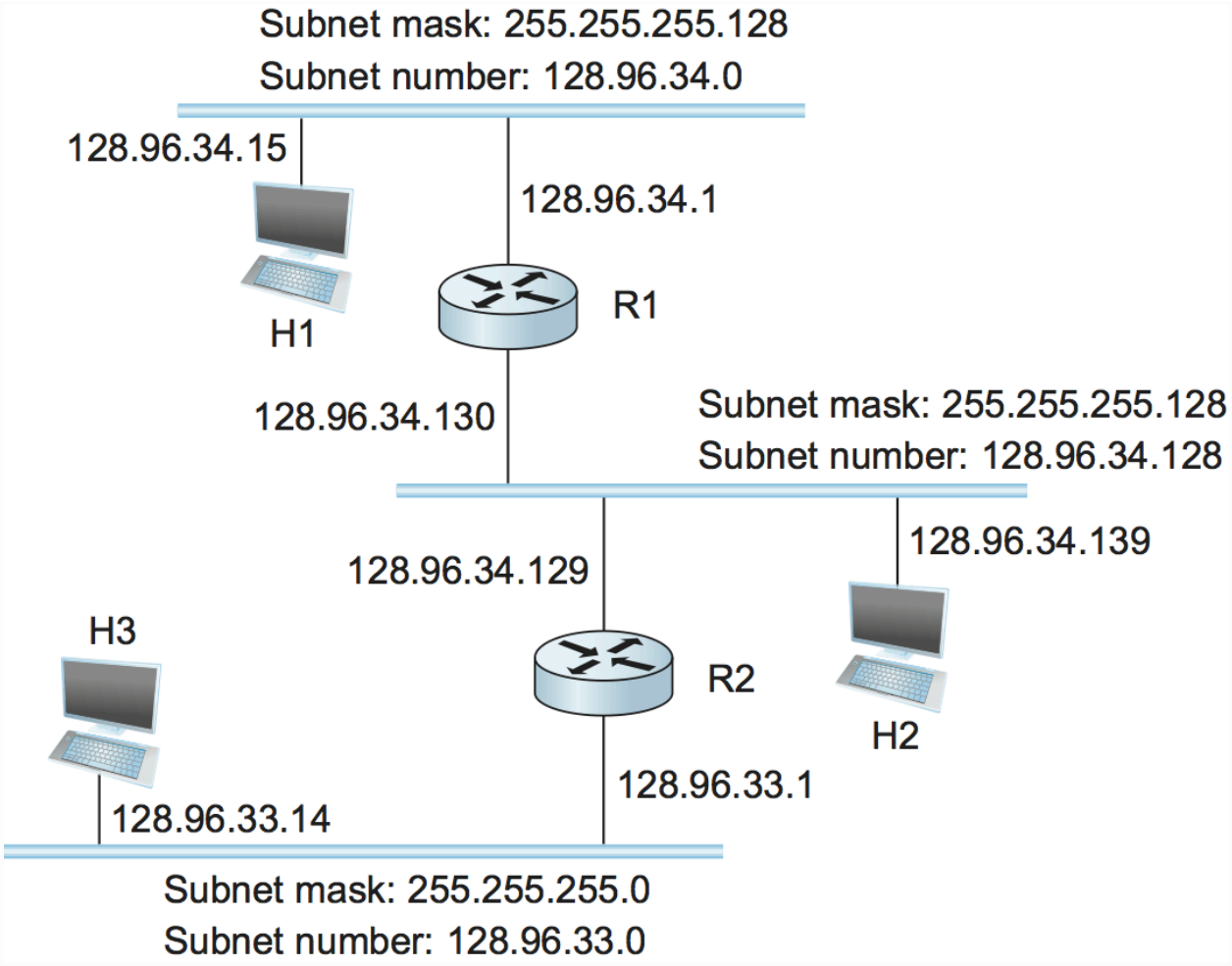


Figura 77. *Um exemplo de sub-rede.*

Quando o host deseja enviar um pacote para um determinado endereço IP, a primeira coisa que ele faz é executar um AND bit a bit entre sua própria máscara de sub-rede e o endereço IP de destino. Se o resultado for igual ao número da sub-rede do host remetente, ele sabe que o host de destino está na mesma sub-rede e o pacote pode ser entregue diretamente pela sub-rede. Se os resultados não forem iguais, o pacote precisa ser enviado a um roteador para ser encaminhado para outra sub-rede. Por exemplo, se H1 estiver enviando para H2, então H1 aplica um AND à sua máscara de

sub-rede (255.255.255.128) com o endereço de H2 (128.96.34.139) para obter 128.96.34.128. Isso não corresponde ao número da sub-rede de H1 (128.96.34.0), então H1 sabe que H2 está em uma sub-rede diferente. Como H1 não pode entregar o pacote para H2 diretamente pela sub-rede, ele envia o pacote para seu roteador padrão R1.

A tabela de encaminhamento de um roteador também muda ligeiramente quando introduzimos a sub-rede. Lembre-se de que anteriormente tínhamos uma tabela de encaminhamento que consistia em entradas do formato . Para suportar a sub-rede, a tabela agora deve conter entradas do formato . Para encontrar a entrada correta na tabela, o roteador aplica um AND ao endereço de destino do pacote com o para cada entrada por vez; se o resultado corresponder ao da entrada, então esta é a entrada correta a ser usada, e ele encaminha o pacote para o roteador do próximo salto indicado. Na rede de exemplo da Figura 77 , o roteador R1 teria as entradas mostradas na Tabela 12 .`(NetworkNum, NextHop)` `(SubnetNumber, SubnetMask, NextHop)` `SubnetMask` `SubnetNumber`

Número de sub-rede	Máscara de sub-rede	Próximo salto
128.96.34.0	255.255.255.128	Interface 0
128.96.34.128	255.255.255.128	Interface 1
128.96.33.0	255.255.255.0	R2

Continuando com o exemplo de um datagrama de H1 sendo enviado para H2, R1 realizaria uma operação AND (e) no endereço de H2 (128.96.34.139) com a máscara

de sub-rede da primeira entrada (255.255.255.128) e compararia o resultado (128.96.34.128) com o número de rede dessa entrada (128.96.34.0). Como não há correspondência, ele prossegue para a próxima entrada. Desta vez, ocorre uma correspondência, então R1 entrega o datagrama para H2 usando a interface 1, que é a interface conectada à mesma rede que H2.

Agora podemos descrever o algoritmo de encaminhamento de datagramas da seguinte maneira:

```
D = destination IP address
for each forwarding table entry (SubnetNumber, SubnetMask, NextHop)
    D1 = SubnetMask & D
    if D1 = SubnetNumber
        if NextHop is an interface
            deliver datagram directly to destination
        else
            deliver datagram to NextHop (a router)
```

Embora não mostrado neste exemplo, uma rota padrão normalmente seria incluída na tabela e seria usada se nenhuma correspondência explícita fosse encontrada. Observe que uma implementação ingênua desse algoritmo — envolvendo a repetição de ANDs do endereço de destino com uma máscara de sub-rede que pode não ser diferente a cada vez e uma busca linear na tabela — seria muito ineficiente.

Uma consequência importante da sub-rede é que diferentes partes da internet veem o mundo de forma diferente. De fora do nosso campus hipotético, os roteadores veem uma única rede. No exemplo acima, os roteadores fora do campus veem o conjunto de redes na Figura 77 como apenas a rede 128.96 e mantêm uma entrada em suas tabelas de encaminhamento para informar como alcançá-la. Os roteadores dentro do campus, no entanto, precisam ser capazes de rotear pacotes para a sub-rede correta. Portanto, nem todas as partes da internet veem exatamente as mesmas informações de roteamento. Este é um exemplo de *agregação* de informações de roteamento, fundamental para o dimensionamento do sistema de roteamento. A próxima seção mostra como a agregação pode ser levada a outro nível.

Endereçamento sem classes

A sub-rede tem uma contrapartida, às vezes chamada de *super-rede* , mas mais frequentemente chamada de *Roteamento Interdomínio Sem Classes* ou CIDR, pronunciado "cider". O CIDR leva a ideia de sub-rede à sua conclusão lógica, essencialmente eliminando completamente as classes de endereços. Por que a sub-rede por si só não é suficiente? Em essência, a sub-rede apenas nos permite dividir um endereço classful entre várias sub-redes, enquanto o CIDR nos permite unir vários endereços classfulful em uma única "super-rede". Isso aborda ainda mais a ineficiência do espaço de endereços mencionada acima e o faz de forma a evitar que o sistema de roteamento seja sobrecarregado.

Para ver como as questões de eficiência do espaço de endereço e escalabilidade do sistema de roteamento se relacionam, considere o caso hipotético de uma empresa cuja rede possui 256 hosts. Isso é um pouco demais para um endereço de Classe C, então você ficaria tentado a atribuir um de Classe B. No entanto, usar um pedaço do espaço de endereço que poderia endereçar 65535 para endereçar 256 hosts tem uma eficiência de apenas 256/65.535 = 0,39%. Embora a sub-rede possa nos ajudar a atribuir endereços com cuidado, ela não contorna o fato de que qualquer organização com mais de 255 hosts, ou com a expectativa de eventualmente ter esse número, deseja um endereço de Classe B.

A primeira maneira de lidar com esse problema seria recusar-se a fornecer um endereço de classe B a qualquer organização que o solicite, a menos que demonstre a necessidade de algo próximo a 64 mil endereços, e, em vez disso, fornecer a ela um número apropriado de endereços de classe C para cobrir o número esperado de hosts. Como agora estaríamos distribuindo espaço de endereço em blocos de 256 endereços por vez, poderíamos comparar com mais precisão a quantidade de espaço de endereço consumido ao tamanho da organização. Para qualquer organização com pelo menos 256 hosts, podemos garantir uma utilização de endereços de pelo menos 50%, e normalmente muito mais. (Infelizmente, mesmo que você possa justificar a solicitação

de um número de rede de classe B, não se incomode, pois todos eles já foram discutidos há muito tempo.)

Essa solução, no entanto, levanta um problema pelo menos tão sério: requisitos excessivos de armazenamento nos roteadores. Se um único site tiver, digamos, 16 números de rede classe C atribuídos a ele, isso significa que cada roteador de backbone da Internet precisa de 16 entradas em suas tabelas de roteamento para direcionar pacotes para esse site. Isso é verdadeiro mesmo que o caminho para todas essas redes seja o mesmo. Se tivéssemos atribuído um endereço classe B ao site, as mesmas informações de roteamento poderiam ser armazenadas em uma entrada da tabela. No entanto, nossa eficiência de atribuição de endereços seria então de apenas 16 x 255 / 65.536 = 6,2%.

O CIDR, portanto, tenta equilibrar o desejo de minimizar o número de rotas que um roteador precisa conhecer com a necessidade de distribuir endereços de forma eficiente. Para isso, o CIDR nos ajuda a *agregar* rotas. Ou seja, ele nos permite usar uma única entrada em uma tabela de encaminhamento para nos informar como alcançar diversas redes diferentes. Como observado acima, ele faz isso rompendo os limites rígidos entre as classes de endereço. Para entender como isso funciona, considere nossa organização hipotética com 16 números de rede de classe C. Em vez de distribuir 16 endereços aleatoriamente, podemos distribuir um bloco de endereços de classe C *contíguos* . Suponha que atribuímos os números de rede de classe C de 192.4.16 a 192.4.31. Observe que os 20 bits superiores de todos os endereços nesse intervalo são os mesmos ( ). Assim, o que efetivamente criamos é um número de rede de 20 bits — algo entre um número de rede de classe B e um número de classe C em termos do número de hosts que ele pode suportar. Em outras palavras, obtemos tanto a alta eficiência de endereços ao distribuir endereços em blocos menores do que uma rede de classe B, quanto um único prefixo de rede que pode ser usado em tabelas de encaminhamento. Observe que, para que esse esquema funcione, precisamos distribuir blocos de endereços de classe C que compartilham um prefixo comum, o que

significa que cada bloco deve conter um número de redes de classe C que seja uma potência de dois.`11000000 00000100 0001`

O CIDR requer um novo tipo de notação para representar números de rede, ou *prefixos* , como são conhecidos, porque os prefixos podem ter qualquer comprimento. A convenção é colocar um `/x`após o prefixo, onde `x`é o comprimento do prefixo em bits. Assim, para o exemplo acima, o prefixo de 20 bits para todas as redes de 192.4.16 a 192.4.31 é representado como 192.4.16/20. Em contraste, se quiséssemos representar um único número de rede classe C, que tem 24 bits de comprimento, o escreveríamos como 192.4.16/24. Hoje, com o CIDR sendo a norma, é mais comum ouvir as pessoas falando sobre prefixos "barra 24" do que sobre redes classe C. Observe que representar um endereço de rede dessa maneira é semelhante à abordagem usada em sub-redes, desde que consista em bits contíguos começando pelo bit mais significativo (o que na prática é quase sempre o caso).`(mask, value)masks`

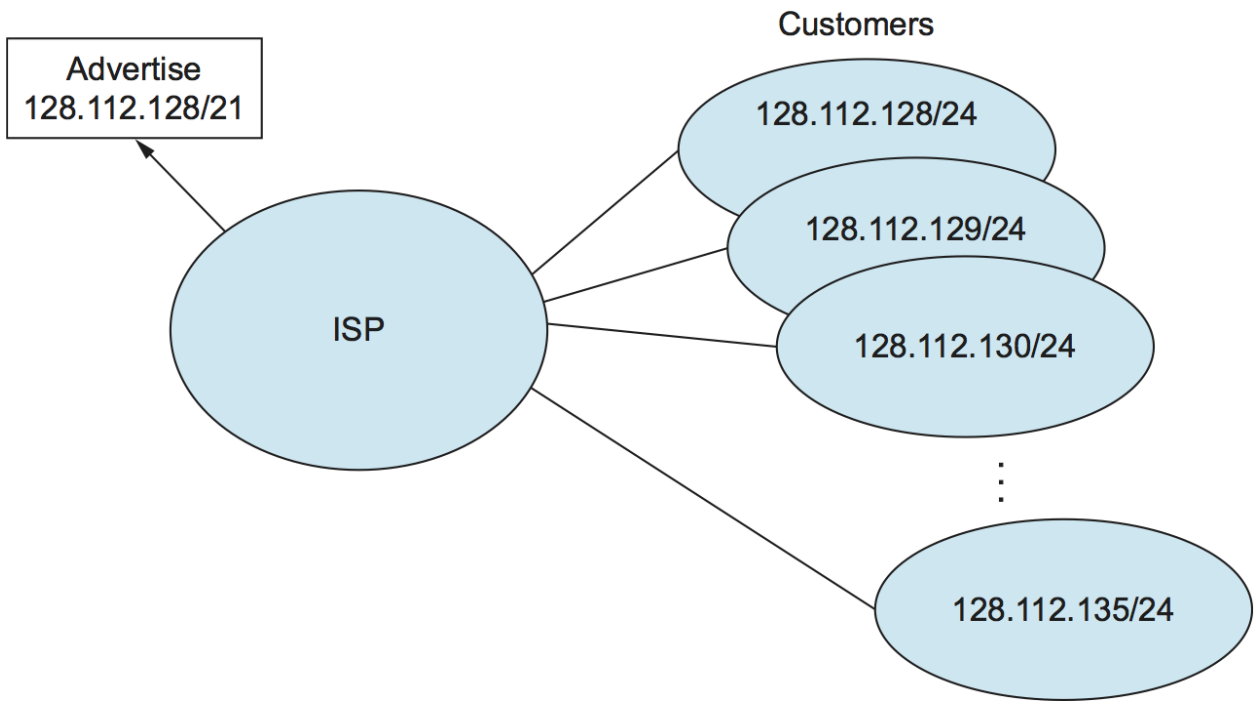


Figura 78. *Agregação de rotas com CIDR.*

A capacidade de agregar rotas na borda da rede, como acabamos de ver, é apenas o primeiro passo. Imagine uma rede de provedor de serviços de Internet, cuja principal

função é fornecer conectividade à Internet para um grande número de empresas e campi (clientes). Se atribuirmos prefixos aos clientes de forma que muitas redes de clientes diferentes conectadas à rede do provedor compartilhem um prefixo de endereço comum e mais curto, podemos obter uma agregação de rotas ainda maior. Considere o exemplo da Figura 78. Suponha que oito clientes atendidos pela rede do provedor tenham recebido prefixos de rede adjacentes de 24 bits. Todos esses prefixos começam com os mesmos 21 bits. Como todos os clientes são acessíveis pela mesma rede do provedor, ele pode anunciar uma única rota para todos eles anunciando apenas o prefixo comum de 21 bits que eles compartilham. E pode fazer isso mesmo que nem todos os prefixos de 24 bits tenham sido distribuídos, desde que o provedor, em última análise, *tenha* o direito de distribuir esses prefixos a um cliente. Uma maneira de fazer isso é atribuir uma parte do espaço de endereço ao provedor com antecedência e, em seguida, permitir que o provedor de rede atribua endereços desse espaço aos seus clientes, conforme necessário. Observe que, ao contrário deste exemplo simples, não há necessidade de que todos os prefixos dos clientes tenham o mesmo comprimento.

Encaminhamento de IP revisitado

Em toda a nossa discussão sobre encaminhamento IP até agora, presumimos que poderíamos encontrar o número da rede em um pacote e, em seguida, procurar esse número em uma tabela de encaminhamento. No entanto, agora que introduzimos o CIDR, precisamos reexaminar essa suposição. CIDR significa que os prefixos podem ter qualquer comprimento, de 2 a 32 bits. Além disso, às vezes é possível ter prefixos na tabela de encaminhamento que se "sobrepõem", no sentido de que alguns endereços podem corresponder a mais de um prefixo. Por exemplo, podemos encontrar 171.69 (um prefixo de 16 bits) e 171.69.10 (um prefixo de 24 bits) na tabela de encaminhamento de um único roteador. Nesse caso, um pacote destinado a, digamos, 171.69.10.5 corresponde claramente a ambos os prefixos. A regra, nesse caso, baseia-se no princípio da "correspondência mais longa"; ou seja, o pacote corresponde ao prefixo mais longo, que seria 171.69.10 neste exemplo. Por outro lado, um pacote destinado a 171.69.20.5 corresponderia a 171.69 e *não* a 171.69.10, e na

ausência de qualquer outra entrada correspondente na tabela de roteamento, 171.69 seria a correspondência mais longa.

A tarefa de encontrar com eficiência a correspondência mais longa entre um endereço IP e os prefixos de comprimento variável em uma tabela de encaminhamento tem sido um campo de pesquisa frutífero há muitos anos. O algoritmo mais conhecido utiliza uma abordagem conhecida como *árvore PATRICIA* , que foi desenvolvida bem antes do CIDR.

3.3.6 Tradução de Endereços (ARP)

Na seção anterior, falamos sobre como enviar datagramas IP para a rede física correta, mas ignoramos a questão de como enviar um datagrama para um host ou roteador específico nessa rede. O principal problema é que os datagramas IP contêm endereços IP, mas o hardware da interface física no host ou roteador para o qual você deseja enviar o datagrama entende apenas o esquema de endereçamento dessa rede específica. Portanto, precisamos traduzir o endereço IP para um endereço de nível de enlace que faça sentido nessa rede (por exemplo, um endereço Ethernet de 48 bits). Podemos então encapsular o datagrama IP dentro de um quadro que contém esse endereço de nível de enlace e enviá-lo ao destino final ou a um roteador que promete encaminhar o datagrama para o destino final.

Uma maneira simples de mapear um endereço IP em um endereço de rede física é codificar o endereço físico de um host na parte referente ao host do seu endereço IP. Por exemplo, um host com endereço físico (que tem o valor decimal 33 no byte superior e 81 no byte inferior) pode receber o endereço IP . Embora essa solução tenha sido usada em algumas redes, ela é limitada, pois os endereços físicos da rede não podem ter mais de 16 bits neste exemplo; eles podem ter apenas 8 bits em uma rede classe C. Isso claramente não funcionará para endereços Ethernet de 48 bits.00100001 01010001128.96.33.81

Uma solução mais geral seria que cada host mantivesse uma tabela de pares de endereços; ou seja, a tabela mapearia endereços IP em endereços físicos. Embora essa tabela pudesse ser gerenciada centralmente por um administrador de sistema e, em seguida, copiada para cada host na rede, uma abordagem melhor seria que cada host aprendesse dinamicamente o conteúdo da tabela usando a rede. Isso pode ser feito usando o Protocolo de Resolução de Endereços (ARP). O objetivo do ARP é permitir que cada host em uma rede crie uma tabela de mapeamentos entre endereços IP e endereços em nível de link. Como esses mapeamentos podem mudar ao longo do tempo (por exemplo, porque uma placa Ethernet em um host quebra e é substituída por uma nova com um novo endereço), as entradas são temporizadas periodicamente e removidas. Isso acontece na ordem de 15 minutos. O conjunto de mapeamentos atualmente armazenados em um host é conhecido como cache ARP ou tabela ARP.

O ARP aproveita o fato de que muitas tecnologias de rede em nível de link, como Ethernet, suportam broadcast. Se um host quiser enviar um datagrama IP para um host (ou roteador) que ele sabe estar na mesma rede (ou seja, os nós de envio e recebimento têm o mesmo número de rede IP), ele primeiro verifica se há um mapeamento no cache. Se nenhum mapeamento for encontrado, ele precisa invocar o Protocolo de Resolução de Endereços pela rede. Ele faz isso transmitindo uma consulta ARP para a rede. Essa consulta contém o endereço IP em questão (o endereço IP de destino). Cada host recebe a consulta e verifica se ela corresponde ao seu endereço IP. Se corresponder, o host envia uma mensagem de resposta que contém seu endereço de camada de link de volta para o originador da consulta. O originador adiciona as informações contidas nessa resposta à sua tabela ARP.

A mensagem de consulta também inclui o endereço IP e o endereço da camada de enlace do host remetente. Assim, quando um host transmite uma mensagem de consulta, cada host na rede pode aprender os endereços IP e de nível de enlace do remetente e inserir essas informações em sua tabela ARP. No entanto, nem todo host adiciona essas informações à sua tabela ARP. Se o host já tiver uma entrada para esse host em sua tabela, ele "atualiza" essa entrada; ou seja, redefine o tempo até descartar

a entrada. Se esse host for o alvo da consulta, ele adiciona as informações sobre o remetente à sua tabela, mesmo que ainda não tenha uma entrada para esse host. Isso ocorre porque há uma boa chance de que o host de origem esteja prestes a enviar uma mensagem em nível de aplicativo e, eventualmente, ele pode ter que enviar uma resposta ou ACK de volta para a origem; ele precisará do endereço físico da origem para fazer isso. Se um host não for o alvo e ainda não tiver uma entrada para a origem em sua tabela ARP, ele não adiciona uma entrada para a origem. Isso ocorre porque não há razão para acreditar que esse host precisará do endereço de nível de link da origem; não há necessidade de sobrecarregar sua tabela ARP com essas informações.

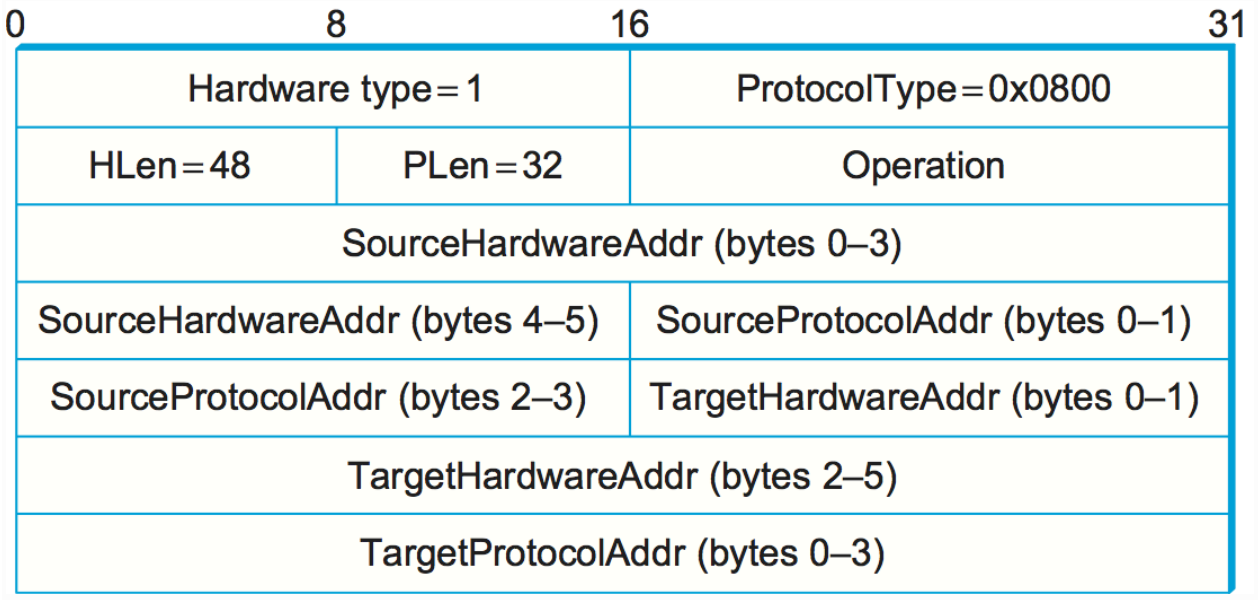


Figura 79. *Formato de pacote ARP para mapear endereços IP em endereços Ethernet.*

A Figura 79 mostra o formato do pacote ARP para mapeamentos de endereços IP para Ethernet. De fato, o ARP pode ser usado para muitos outros tipos de mapeamento — as principais diferenças estão nos tamanhos dos endereços. Além dos endereços IP e da camada de enlace do remetente e do destino, o pacote contém

- Um `HardwareType`campo que especifica o tipo de rede física (por exemplo, Ethernet)

- Um `ProtocolType`campo que especifica o protocolo de camada superior (por exemplo, IP)
- `HLen`(comprimento do endereço de “hardware”) e `PLen`(comprimento do endereço de “protocolo”) campos, que especificam o comprimento do endereço da camada de enlace e do endereço do protocolo da camada superior, respectivamente
- Um `Operation`campo que especifica se esta é uma solicitação ou uma resposta
- Os endereços de hardware de origem e destino (Ethernet) e protocolo (IP)

Observe que os resultados do processo ARP podem ser adicionados como uma coluna extra em uma tabela de encaminhamento como a da Tabela 10. Assim, por exemplo, quando R2 precisa encaminhar um pacote para a rede 2, ele não apenas descobre que o próximo salto é R1, mas também encontra o endereço MAC para colocar no pacote para enviá-lo a R1.

Conclusão principal

Vimos agora os mecanismos básicos que o IP fornece para lidar com heterogeneidade e escala. Em relação à heterogeneidade, o IP começa definindo um modelo de serviço de melhor esforço que faz suposições mínimas sobre as redes subjacentes; mais notavelmente, esse modelo de serviço é baseado em datagramas não confiáveis. O IP então faz duas adições importantes a esse ponto de partida: (1) um formato de pacote comum (fragmentação/remontagem é o mecanismo que faz esse formato funcionar em redes com diferentes MTUs) e (2) um espaço de endereço global para identificar todos os hosts (ARP é o mecanismo que faz esse espaço de endereço global funcionar em redes com diferentes esquemas de endereçamento físico). Em relação à escala, o IP usa agregação hierárquica para reduzir a quantidade de informações necessárias para encaminhar pacotes. Especificamente, os endereços IP são particionados em componentes de rede e host, com os pacotes primeiro roteados para a rede de destino e, em seguida, entregues ao host correto nessa rede. [Próximo]

3.3.7 Configuração do Host (DHCP)

Os endereços Ethernet são configurados no adaptador de rede pelo fabricante, e esse processo é gerenciado de forma a garantir que esses endereços sejam globalmente únicos. Esta é claramente uma condição suficiente para garantir que qualquer conjunto de hosts conectados a uma única Ethernet (incluindo uma LAN estendida) tenha endereços únicos. Além disso, exclusividade é tudo o que exigimos dos endereços Ethernet.

Os endereços IP, por outro lado, não só devem ser únicos em uma determinada rede, como também devem refletir a estrutura da rede. Como observado acima, eles contêm uma parte de rede e uma parte de host, e a parte de rede deve ser a mesma para todos os hosts na mesma rede. Portanto, não é possível que o endereço IP seja configurado uma vez em um host quando ele é fabricado, pois isso implicaria que o fabricante sabia quais hosts iriam parar em quais redes, e isso significaria que um host, uma vez conectado a uma rede, nunca poderia se mover para outra. Por esse motivo, os endereços IP precisam ser reconfiguráveis.

Além do endereço IP, existem outras informações que um host precisa ter antes de começar a enviar pacotes. A mais importante delas é o endereço de um roteador padrão — o local para onde ele pode enviar pacotes cujo endereço de destino não esteja na mesma rede do host remetente.

A maioria dos sistemas operacionais de host oferece uma maneira para um administrador de sistema, ou mesmo um usuário, configurar manualmente as informações de IP necessárias para um host; no entanto, existem algumas desvantagens óbvias nessa configuração manual. Uma delas é que configurar todos os hosts em uma grande rede diretamente dá muito trabalho, especialmente quando se considera que esses hosts não podem ser acessados pela rede até que sejam configurados. Ainda mais importante, o processo de configuração é muito propenso a erros, pois é necessário garantir que cada host receba o número de rede correto e que dois hosts não recebam o mesmo endereço IP. Por esses motivos, métodos de

configuração automatizados são necessários. O método principal utiliza um protocolo conhecido como *Protocolo de Configuração Dinâmica de Hosts* (DHCP).

O DHCP depende da existência de um servidor DHCP responsável por fornecer informações de configuração aos hosts. Há pelo menos um servidor DHCP para um domínio administrativo. Em um nível mais simples, o servidor DHCP pode funcionar como um repositório centralizado para informações de configuração do host. Considere, por exemplo, o problema de administrar endereços na interconexão de redes de uma grande empresa. O DHCP evita que os administradores de rede precisem percorrer todos os hosts da empresa com uma lista de endereços e um mapa de rede em mãos e configurar cada host manualmente. Em vez disso, as informações de configuração de cada host podem ser armazenadas no servidor DHCP e recuperadas automaticamente por cada host quando ele é inicializado ou conectado à rede. No entanto, o administrador ainda escolheria o endereço que cada host deve receber; ele apenas armazenaria isso no servidor. Nesse modelo, as informações de configuração de cada host são armazenadas em uma tabela indexada por algum tipo de identificador exclusivo de cliente, normalmente o endereço de hardware (por exemplo, o endereço Ethernet do adaptador de rede).

Um uso mais sofisticado do DHCP evita que o administrador de rede precise atribuir endereços a hosts individuais. Nesse modelo, o servidor DHCP mantém um conjunto de endereços disponíveis que ele distribui aos hosts sob demanda. Isso reduz consideravelmente a quantidade de configuração que um administrador precisa fazer, já que agora é necessário apenas alocar um intervalo de endereços IP (todos com o mesmo número de rede) para cada rede.

Como o objetivo do DHCP é minimizar a quantidade de configuração manual necessária para o funcionamento de um host, seria inútil se cada host tivesse que ser configurado com o endereço de um servidor DHCP. Assim, o primeiro problema enfrentado pelo DHCP é a descoberta de servidores.

Para contatar um servidor DHCP, um host recém-inicializado ou conectado envia uma `DHCPDISCOVER`mensagem para um endereço IP especial (255.255.255.255) que é um endereço IP de broadcast. Isso significa que ela será recebida por todos os hosts e roteadores naquela rede. (Os roteadores não encaminham esses pacotes para outras redes, impedindo a transmissão para toda a Internet.) No caso mais simples, um desses nós é o servidor DHCP da rede. O servidor então responderia ao host que gerou a mensagem de descoberta (todos os outros nós a ignorariam). No entanto, não é realmente desejável exigir um servidor DHCP em cada rede, porque isso ainda cria um número potencialmente grande de servidores que precisam ser configurados de forma correta e consistente. Assim, o DHCP usa o conceito de um *agente de retransmissão* . Há pelo menos um agente de retransmissão em cada rede, e ele é configurado com apenas uma informação: o endereço IP do servidor DHCP. Quando um agente de retransmissão recebe uma `DHCPDISCOVER`mensagem, ele a transmite por unicast para o servidor DHCP e aguarda a resposta, que então enviará de volta ao cliente solicitante. O processo de retransmissão de uma mensagem de um host para um servidor DHCP remoto é mostrado na Figura 80 .

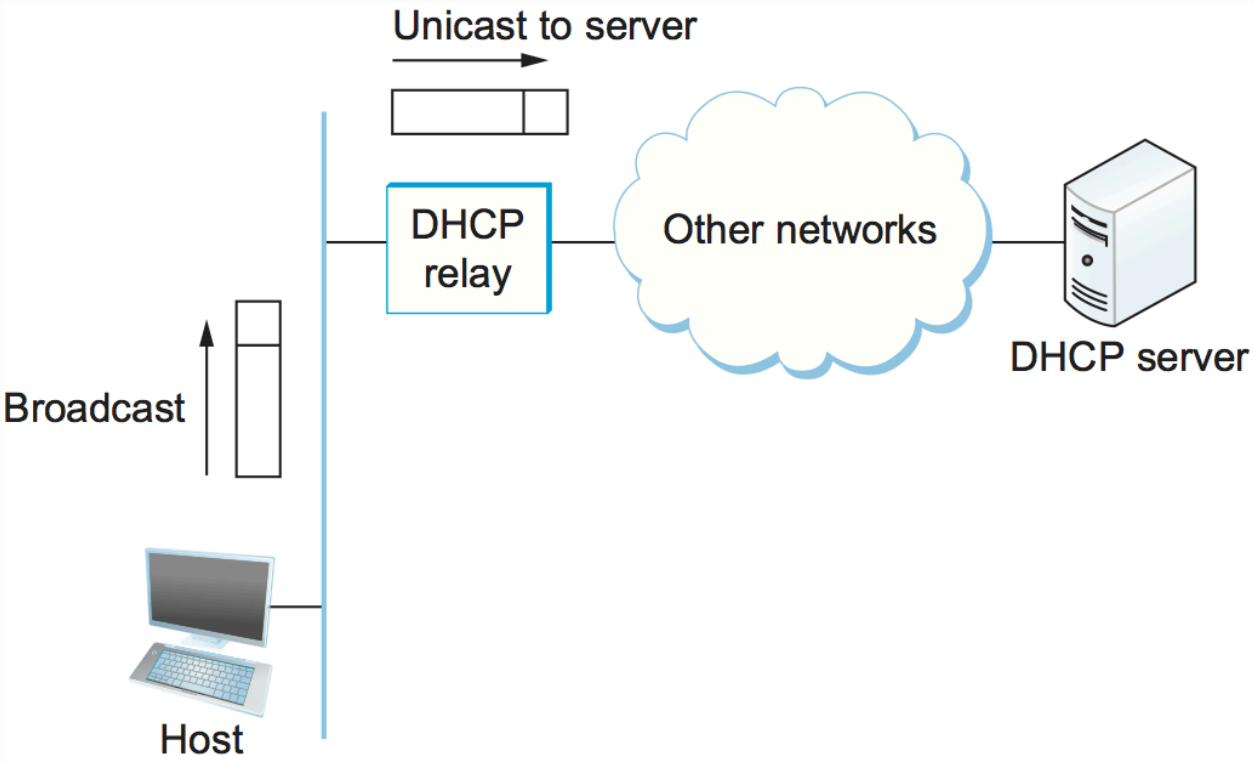


Figura 80. *Um agente de retransmissão DHCP recebe uma mensagem DHCPDISCOVER de transmissão de um host e envia uma mensagem DHCPDISCOVER unicast para o servidor DHCP.*

A Figura 81 abaixo mostra o formato de uma mensagem DHCP. A mensagem é, na verdade, enviada usando um protocolo chamado *Protocolo de Datagrama de Usuário* (UDP), que opera sobre IP. O UDP é discutido em detalhes no próximo capítulo, mas a única coisa interessante que ele faz neste contexto é fornecer uma chave de demultiplexação que diz: "Este é um pacote DHCP".

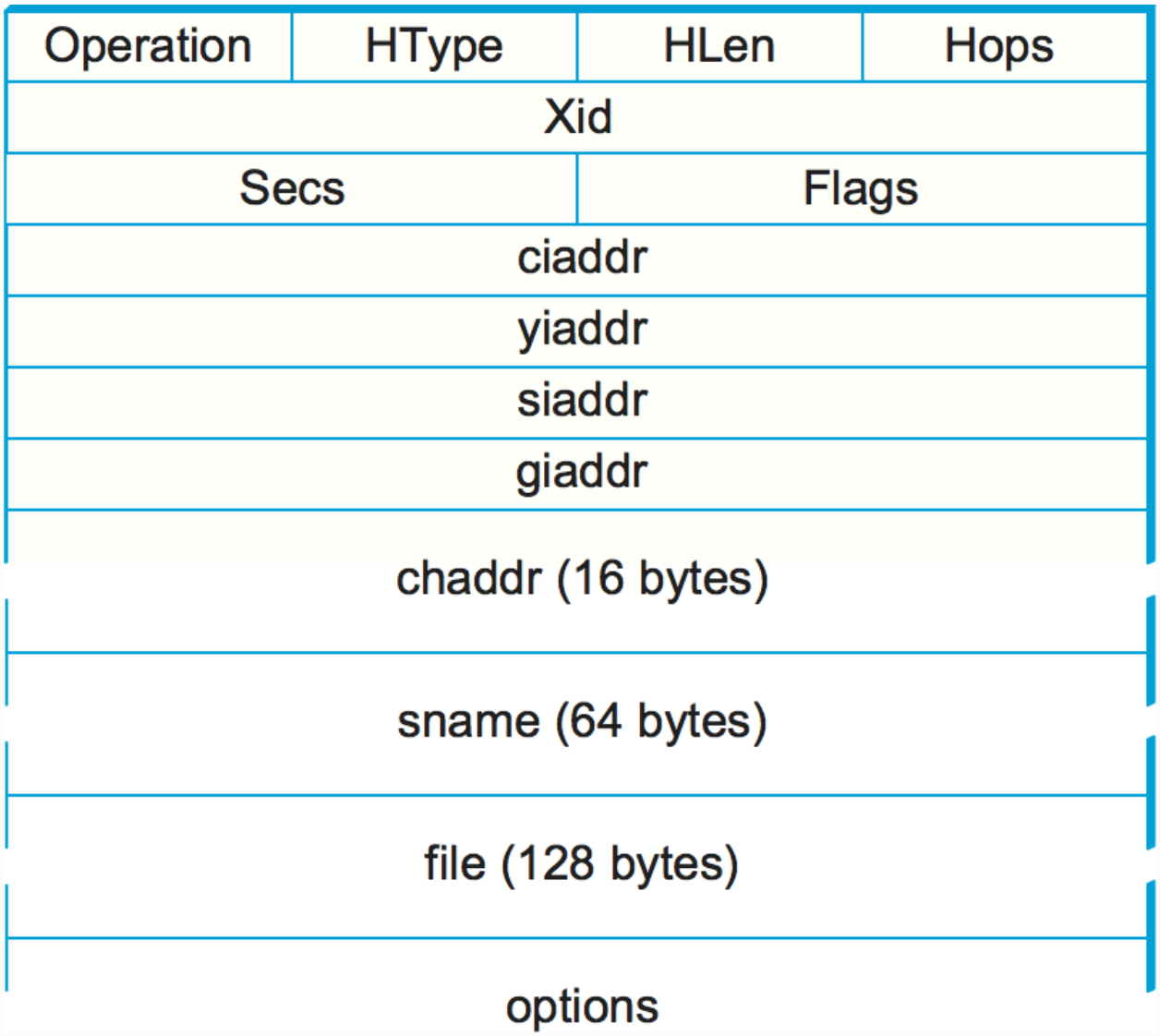


Figura 81. *Formato do pacote DHCP.*

O DHCP é derivado de um protocolo anterior chamado BOOTP e, portanto, alguns campos do pacote não são estritamente relevantes para a configuração do host. Ao tentar obter informações de configuração, o cliente insere seu endereço de hardware (por exemplo, seu endereço Ethernet) no `chaddr`campo. O servidor DHCP responde preenchendo o `yiaddr`campo ("seu" endereço IP) e enviando-o ao cliente. Outras informações, como o roteador padrão a ser usado por este cliente, podem ser incluídas no `options` campo.

No caso em que o DHCP atribui dinamicamente endereços IP aos hosts, fica claro que os hosts não podem manter endereços indefinidamente, pois isso acabaria levando o servidor a esgotar seu pool de endereços. Ao mesmo tempo, não se pode depender de um host para devolver seu endereço, pois ele pode ter travado, sido desconectado da rede ou ter sido desligado. Assim, o DHCP permite que endereços sejam alugados por um determinado período. Uma vez que o aluguel expire, o servidor está livre para devolver esse endereço ao seu pool. Um host com um endereço alugado claramente precisa renovar o aluguel periodicamente se, de fato, ainda estiver conectado à rede e funcionando corretamente.

Conclusão principal

O DHCP ilustra um aspecto importante do escalonamento: o escalonamento do gerenciamento de rede. Embora as discussões sobre escalonamento frequentemente se concentrem em evitar que o estado dos dispositivos de rede cresça muito rápido, é importante atentar para o aumento da complexidade do gerenciamento de rede. Ao permitir que os gerentes de rede configurem um intervalo de endereços IP por rede, em vez de um endereço IP por host, o DHCP melhora a capacidade de gerenciamento de uma rede. [Próximo]

Observe que o DHCP também pode introduzir um pouco mais de complexidade no gerenciamento de rede, pois torna a vinculação entre hosts físicos e endereços IP

muito mais dinâmica. Isso pode dificultar o trabalho do gerente de rede se, por exemplo, for necessário localizar um host com defeito.

3.3.8 Relatório de Erros (ICMP)

A próxima questão é como a Internet trata erros. Embora o IP esteja perfeitamente disposto a descartar datagramas quando as coisas ficam difíceis — por exemplo, quando um roteador não sabe como encaminhar o datagrama ou quando um fragmento de um datagrama não chega ao destino —, ele não necessariamente falha silenciosamente. O IP é sempre configurado com um protocolo complementar, conhecido como *Internet Control Message Protocol* (ICMP), que define um conjunto de mensagens de erro enviadas de volta ao host de origem sempre que um roteador ou host não consegue processar um datagrama IP com sucesso. Por exemplo, o ICMP define mensagens de erro indicando que o host de destino está inacessível (talvez devido a uma falha de link), que o processo de remontagem falhou, que o TTL atingiu 0, que a soma de verificação do cabeçalho IP falhou e assim por diante.

O ICMP também define um punhado de mensagens de controle que um roteador pode enviar de volta para um host de origem. Uma das mensagens de controle mais úteis, chamada de *ICMP-Redirect* , informa ao host de origem que há uma rota melhor para o destino. Os ICMP-Redirects são usados na seguinte situação. Suponha que um host esteja conectado a uma rede que tenha dois roteadores conectados a ele, chamados *R1* e *R2* , onde o host usa R1 como seu roteador padrão. Caso R1 receba um datagrama do host, onde, com base em sua tabela de encaminhamento, ele sabe que R2 teria sido uma escolha melhor para um endereço de destino específico, ele envia um ICMP-Redirect de volta ao host, instruindo-o a usar R2 para todos os datagramas futuros endereçados a esse destino. O host então adiciona essa nova rota à sua tabela de encaminhamento.

O ICMP também fornece a base para duas ferramentas de depuração amplamente utilizadas `ping`e `traceroute`. `ping`usa mensagens de eco ICMP para determinar se um nó está acessível e ativo. `traceroute`usa uma técnica um pouco não intuitiva para

determinar o conjunto de roteadores ao longo do caminho para um destino, que é o tópico de um dos exercícios no final deste capítulo.

3.3.9 Redes e Túneis Virtuais

Concluímos nossa introdução ao IP considerando uma questão que você talvez não tenha previsto, mas que se torna cada vez mais importante. Nossa discussão até este ponto se concentrou em tornar possível que nós em diferentes redes se comuniquem entre si de forma irrestrita. Esse é geralmente o objetivo na internet — todos querem poder enviar e-mails para todos, e o criador de um novo site quer atingir o maior público possível. No entanto, existem muitas situações em que uma conectividade mais controlada é necessária. Um exemplo importante dessa situação é a *rede privada virtual* (VPN).

O termo *VPN* é muito utilizado e as definições variam, mas intuitivamente podemos definir uma VPN considerando primeiro a ideia de uma rede privada. Corporações com muitos sites frequentemente constroem redes privadas alugando circuitos de companhias telefônicas e usando essas linhas para interconectar sites. Em tal rede, a comunicação é restrita a ocorrer apenas entre os sites daquela corporação, o que geralmente é desejável por razões de segurança. Para tornar uma rede privada *virtual* , as linhas de transmissão alugadas — que não são compartilhadas com nenhuma outra corporação — seriam substituídas por algum tipo de rede compartilhada. Um circuito virtual (VC) é um substituto muito razoável para uma linha alugada porque ainda fornece uma conexão lógica ponto a ponto entre os sites da corporação. Por exemplo, se a corporação X tem um VC do site A para o site B, então claramente ela pode enviar pacotes entre os sites A e B. Mas não há como a corporação Y conseguir que seus pacotes sejam entregues ao site B sem primeiro estabelecer seu próprio circuito virtual para o site B, e o estabelecimento de tal VC pode ser impedido administrativamente, prevenindo assim a conectividade indesejada entre a corporação X e a corporação Y.

A Figura 82(a) mostra duas redes privadas para duas empresas distintas. Na Figura 82(b), ambas são migradas para uma rede de circuito virtual. A conectividade limitada

de uma rede privada real é mantida, mas como as redes privadas agora compartilham as mesmas instalações de transmissão e switches, dizemos que duas redes privadas virtuais foram criadas.

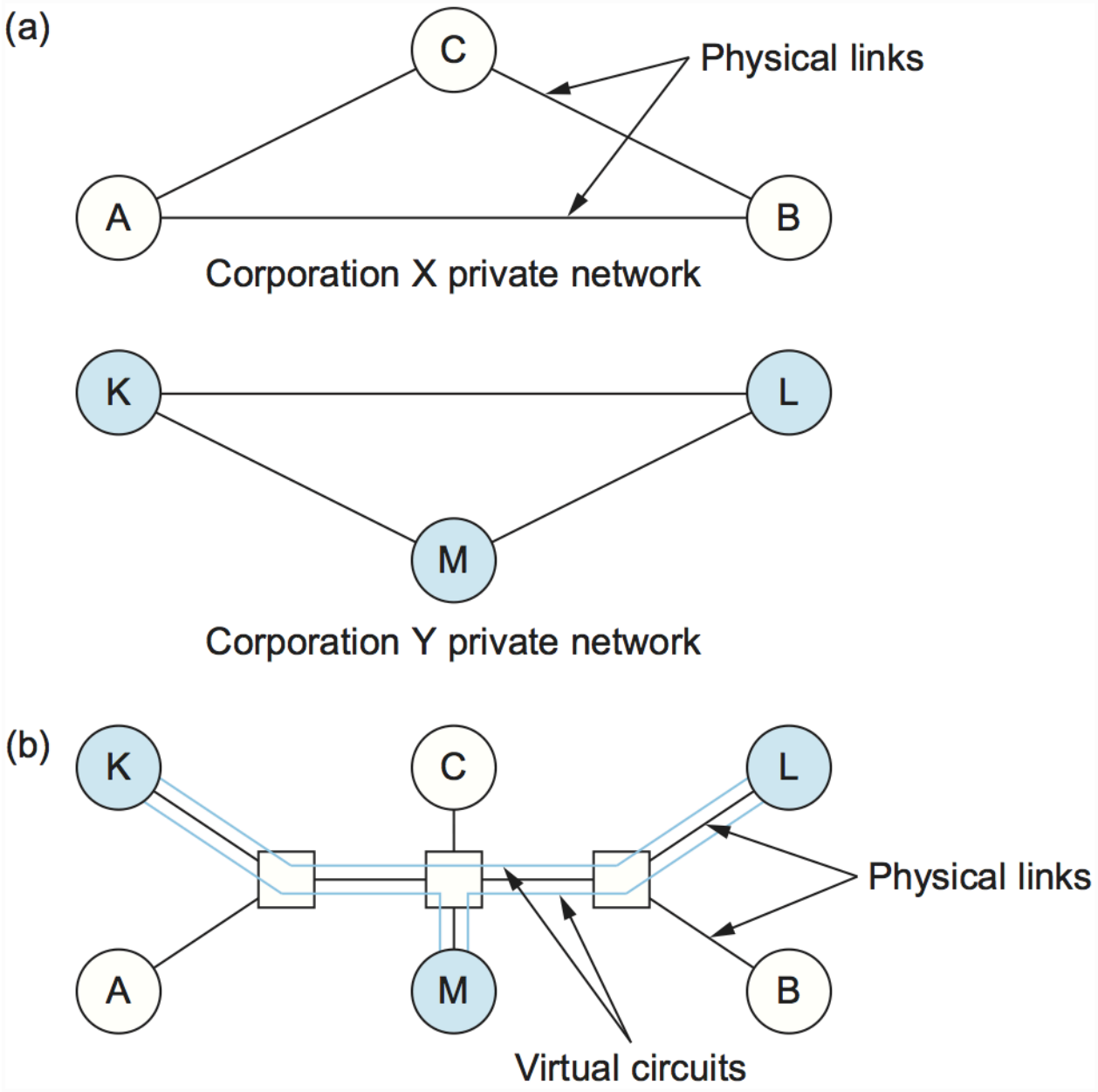


Figura 82. *Um exemplo de redes privadas virtuais: (a) duas redes privadas separadas; (b) duas redes privadas virtuais compartilhando switches comuns.*

Na Figura 82 , uma rede de circuitos virtuais (usando ATM, por exemplo) é usada para fornecer conectividade controlada entre sites. Também é possível fornecer uma função

semelhante usando uma rede IP para fornecer conectividade. No entanto, não podemos simplesmente conectar os sites de várias empresas a uma única interconexão de redes, pois isso forneceria conectividade entre a empresa X e a empresa Y, o que desejamos evitar. Para resolver esse problema, precisamos introduzir um novo conceito, o *túnel IP* .

Podemos pensar em um túnel IP como um enlace virtual ponto a ponto entre um par de nós que, na verdade, estão separados por um número arbitrário de redes. O enlace virtual é criado dentro do roteador na entrada do túnel, fornecendo a ele o endereço IP do roteador na extremidade oposta do túnel. Sempre que o roteador na entrada do túnel deseja enviar um pacote por esse enlace virtual, ele o encapsula dentro de um datagrama IP. O endereço de destino no cabeçalho IP é o endereço do roteador na extremidade oposta do túnel, enquanto o endereço de origem é o do roteador encapsulador.

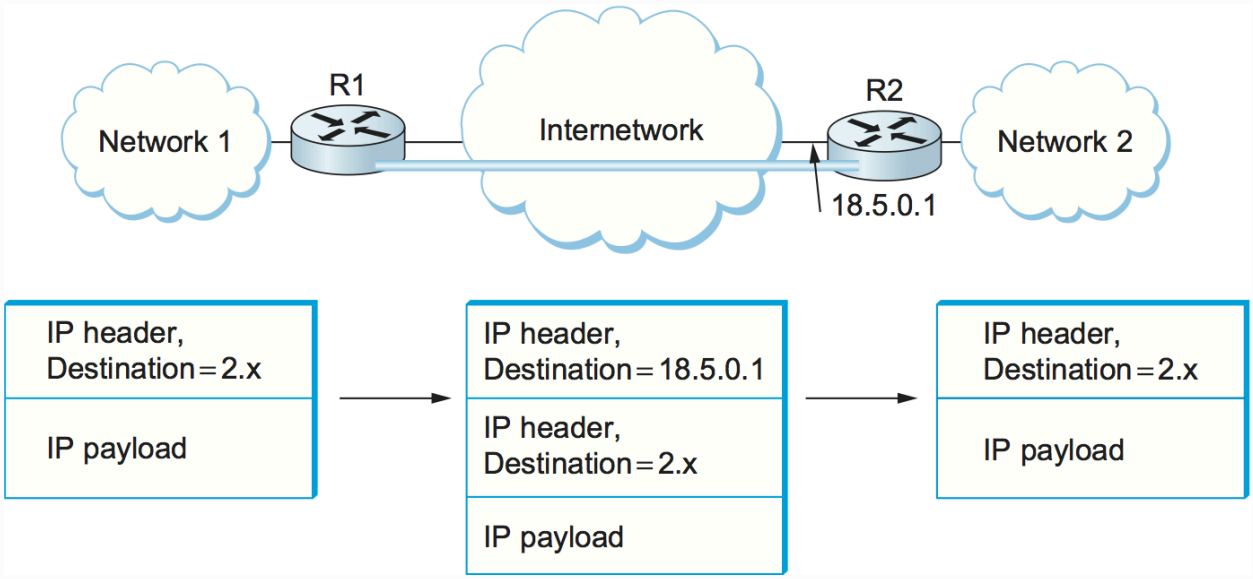


Figura 83. *Um túnel através de uma interconexão de redes. 18.5.0.1 é o endereço de R2 que pode ser acessado de R1 através da interconexão de redes.*

Na tabela de encaminhamento do roteador na entrada do túnel, este enlace virtual se parece muito com um enlace normal. Considere, por exemplo, a rede na Figura 83. Um túnel foi configurado de R1 a R2 e recebeu um número de interface virtual 0. A tabela de encaminhamento em R1 pode, portanto, se parecer com a Tabela 13 .

Número da rede	Próximo salto
1	Interface 0
2	Interface virtual 0
Padrão	Interface 1

R1 possui duas interfaces físicas. A interface 0 conecta-se à rede 1; a interface 1 conecta-se a uma grande rede interconectada e, portanto, é o padrão para todo o tráfego que não corresponde a algo mais específico na tabela de encaminhamento. Além disso, R1 possui uma interface virtual, que é a interface para o túnel. Suponha que R1 receba um pacote da rede 1 que contém um endereço na rede 2. A tabela de encaminhamento indica que esse pacote deve ser enviado pela interface virtual 0. Para enviar um pacote por essa interface, o roteador recebe o pacote, adiciona um cabeçalho IP endereçado a R2 e, em seguida, encaminha o pacote como se tivesse acabado de ser recebido. O endereço de R2 é 18.5.0.1; como o número de rede desse endereço é 18, e não 1 ou 2, um pacote destinado a R2 será encaminhado pela interface padrão para a rede interconectada.

Assim que o pacote sai de R1, ele se apresenta ao resto do mundo como um pacote IP normal destinado a R2 e é encaminhado adequadamente. Todos os roteadores na rede o encaminham usando meios normais até que ele chegue a R2. Quando R2 recebe o pacote, ele descobre que ele carrega seu próprio endereço, então remove o cabeçalho IP e examina a carga útil do pacote. O que ele encontra é um pacote IP interno cujo endereço de destino está na rede 2. R2 agora processa esse pacote como qualquer

outro pacote IP que recebe. Como R2 está conectado diretamente à rede 2, ele encaminha o pacote para essa rede. A Figura 83 mostra a mudança no encapsulamento do pacote à medida que ele se move pela rede.

Embora o R2 atue como ponto final do túnel, não há nada que o impeça de executar as funções normais de um roteador. Por exemplo, ele pode receber alguns pacotes que não estão encapsulados, mas que são endereçados a redes que ele sabe como alcançar, e os encaminharia normalmente.

Você pode se perguntar por que alguém se daria ao trabalho de criar um túnel e alterar o encapsulamento de um pacote à medida que ele atravessa uma rede interconectada. Um dos motivos é a segurança. Complementado com criptografia, um túnel pode se tornar um tipo de enlace muito privado em uma rede pública. Outro motivo pode ser que R1 e R2 possuem alguns recursos que não estão amplamente disponíveis nas redes intervenientes, como roteamento multicast. Ao conectar esses roteadores a um túnel, podemos construir uma rede virtual na qual todos os roteadores com esse recurso parecem estar conectados diretamente. Um terceiro motivo para construir túneis é transportar pacotes de protocolos diferentes do IP através de uma rede IP. Contanto que os roteadores em cada extremidade do túnel saibam como lidar com esses outros protocolos, o túnel IP parece para eles um enlace ponto a ponto pelo qual podem enviar pacotes não IP. Os túneis também fornecem um mecanismo pelo qual podemos forçar a entrega de um pacote a um local específico, mesmo que seu cabeçalho original — aquele que é encapsulado dentro do cabeçalho do túnel — sugira que ele deva ir para outro lugar. Assim, vemos que o tunelamento é uma técnica poderosa e bastante geral para a construção de enlaces virtuais entre inter-redes. Tão geral, na verdade, que a técnica é recorrente, sendo o caso de uso mais comum o tunelamento IP sobre IP.

O tunelamento tem suas desvantagens. Uma delas é que ele aumenta o comprimento dos pacotes; isso pode representar um desperdício significativo de largura de banda para pacotes curtos. Pacotes mais longos podem estar sujeitos à fragmentação, o que tem suas próprias desvantagens. Também pode haver implicações de desempenho

para os roteadores em cada extremidade do túnel, uma vez que eles precisam realizar mais trabalho do que o encaminhamento normal, adicionando e removendo o cabeçalho do túnel. Por fim, há um custo de gerenciamento para a entidade administrativa responsável por configurar os túneis e garantir que eles sejam tratados corretamente pelos protocolos de roteamento.

3.4 Roteamento

Até agora neste capítulo, assumimos que os switches e roteadores têm conhecimento suficiente da topologia da rede para que possam escolher a porta correta para a qual cada pacote deve ser enviado. No caso de circuitos virtuais, o roteamento é um problema apenas para o pacote de solicitação de conexão; todos os pacotes subsequentes seguem o mesmo caminho da solicitação. Em redes de datagramas, incluindo redes IP, o roteamento é um problema para todos os pacotes. Em ambos os casos, um switch ou roteador precisa ser capaz de analisar um endereço de destino e, em seguida, determinar qual das portas de saída é a melhor escolha para enviar um pacote a esse endereço. Como vimos em uma seção anterior, o switch toma essa decisão consultando uma tabela de encaminhamento. O problema fundamental do roteamento é como os switches e roteadores obtêm as informações em suas tabelas de encaminhamento.

Conclusão principal

Reafirmamos uma distinção importante, frequentemente negligenciada, entre *encaminhamento* e *roteamento* . O encaminhamento consiste em receber um pacote, consultar seu endereço de destino em uma tabela e enviá-lo em uma direção determinada por essa tabela. Vimos vários exemplos de encaminhamento na seção anterior. É um processo simples e bem definido, executado localmente em cada nó, e frequentemente chamado de *plano de dados da rede.* O roteamento é o processo pelo qual as tabelas de encaminhamento são construídas. Ele depende de algoritmos

distribuídos complexos e é frequentemente chamado de *plano de controle da rede.* [Próximo]

Embora os termos *tabela de encaminhamento* e *tabela de roteamento* sejam às vezes usados indistintamente, faremos uma distinção entre eles aqui. A tabela de encaminhamento é usada quando um pacote está sendo encaminhado e, portanto, deve conter informações suficientes para realizar a função de encaminhamento. Isso significa que uma linha na tabela de encaminhamento contém o mapeamento de um prefixo de rede para uma interface de saída e algumas informações MAC, como o endereço Ethernet do próximo salto. A tabela de roteamento, por outro lado, é a tabela construída pelos algoritmos de roteamento como um precursor para a construção da tabela de encaminhamento. Ela geralmente contém mapeamentos de prefixos de rede para os próximos saltos. Ela também pode conter informações sobre como essas informações foram aprendidas, para que o roteador possa decidir quando deve descartar algumas informações.

Se a tabela de roteamento e a tabela de encaminhamento são realmente estruturas de dados separadas é uma questão de implementação, mas há inúmeras razões para mantê-las separadas. Por exemplo, a tabela de encaminhamento precisa ser estruturada para otimizar o processo de busca de endereços ao encaminhar um pacote, enquanto a tabela de roteamento precisa ser otimizada para fins de cálculo de alterações na topologia. Em muitos casos, a tabela de encaminhamento pode até ser implementada em hardware especializado, enquanto isso raramente, ou nunca, é feito para a tabela de roteamento.

A Tabela 14 fornece um exemplo de uma linha de uma tabela de roteamento, que nos informa que o prefixo de rede 18/8 deve ser alcançado por um roteador de próximo salto com o endereço IP 171.69.245.10

Prefixo/Comprimento	Próximo salto
18/8	171.69.245.10

Em contraste, a Tabela 15 apresenta um exemplo de uma linha de uma tabela de encaminhamento, que contém informações sobre como encaminhar exatamente um pacote para o próximo salto: envie-o pela interface número 0 com um endereço MAC de 8:0:2b:e4:b:1:2. Observe que a última informação é fornecida pelo Protocolo de Resolução de Endereços.

Prefixo/Comprimento	Interface	Endereço MAC
18/8	se0	8:0:2b:e4:b:1:2

Antes de entrarmos nos detalhes do roteamento, precisamos nos lembrar da pergunta-chave que devemos nos fazer sempre que tentamos construir um mecanismo para a Internet: "Esta solução é escalável?" A resposta para os algoritmos e protocolos descritos nesta seção é "nem tanto". Eles são projetados para redes de tamanho bastante modesto — até algumas centenas de nós, na prática. No entanto, as soluções que descrevemos servem como um bloco de construção para uma infraestrutura de roteamento hierárquico que é usada na Internet hoje. Especificamente, os protocolos descritos nesta seção são conhecidos coletivamente como protocolos de roteamento *intradomínio* ou *protocolos de gateway interior* (IGPs). Para entender esses termos, precisamos definir um *domínio* de roteamento . Uma boa definição de trabalho é uma

inter-rede na qual todos os roteadores estão sob o mesmo controle administrativo (por exemplo, um único campus universitário ou a rede de um único provedor de serviços de Internet). A relevância dessa definição se tornará aparente no próximo capítulo, quando analisarmos os protocolos de roteamento *interdomínio* . Por enquanto, o importante a ter em mente é que estamos considerando o problema de roteamento no contexto de redes de pequeno e médio porte, não para uma rede do tamanho da Internet.

3.4.1 Rede como um grafo

O roteamento é, em essência, um problema de teoria dos grafos. A Figura 84 mostra um grafo representando uma rede. Os nós do grafo, rotulados de A a F, podem ser hosts, switches, roteadores ou redes. Para nossa discussão inicial, vamos nos concentrar no caso em que os nós são roteadores. As arestas do grafo correspondem aos enlaces da rede. Cada aresta tem um *custo* associado , que fornece alguma indicação da conveniência de enviar tráfego por esse enlace. Uma discussão sobre como os custos das arestas são atribuídos será apresentada em uma seção posterior.

Observe que as redes de exemplo (grafos) usadas ao longo deste capítulo têm arestas não direcionadas às quais é atribuído um único custo. Na verdade, trata-se de uma pequena simplificação. É mais preciso direcionar as arestas, o que normalmente significa que haveria um par de arestas entre cada nó — uma fluindo em cada direção e cada uma com seu próprio custo de aresta.

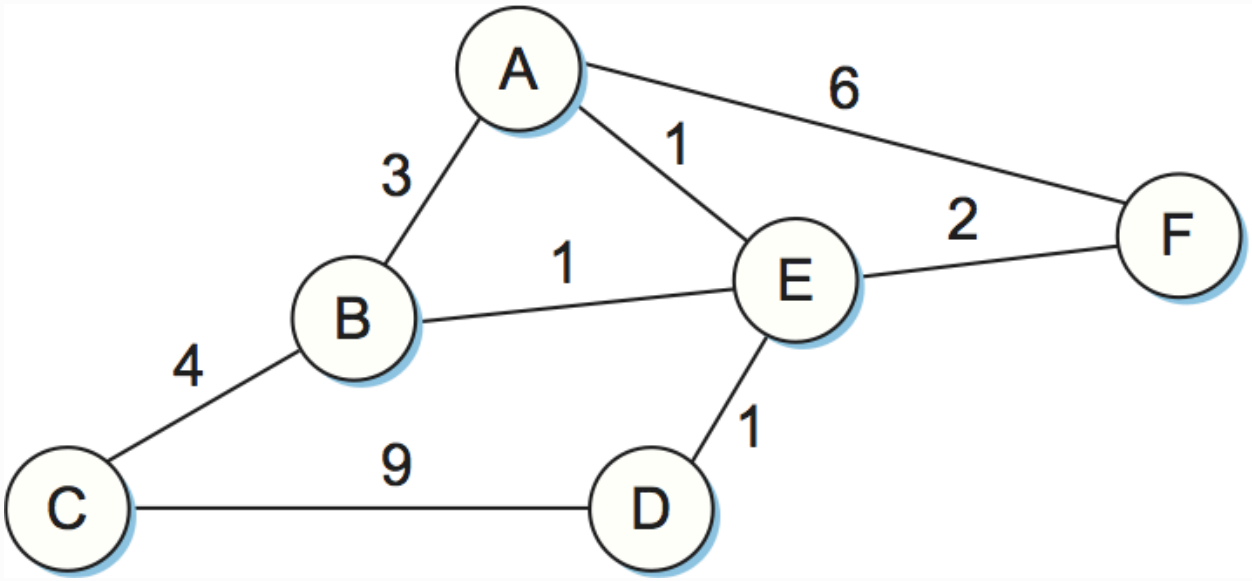


Figura 84. *Rede representada como um gráfico.*

O problema básico do roteamento é encontrar o caminho de menor custo entre dois nós, onde o custo de um caminho é igual à soma dos custos de todas as arestas que o compõem. Para uma rede simples como a da Figura 84 , você poderia imaginar simplesmente calcular todos os caminhos mais curtos e carregá-los em algum armazenamento não volátil em cada nó. Essa abordagem estática apresenta várias deficiências:

- Ele não lida com falhas de nós ou links.
- Não considera a adição de novos nós ou links.
- Isso implica que os custos de borda não podem mudar, mesmo que possamos desejar razoavelmente que os custos de link mudem ao longo do tempo (por exemplo, atribuindo alto custo a um link muito carregado).

Por essas razões, o roteamento é alcançado na maioria das redes práticas pela execução de protocolos de roteamento entre os nós. Esses protocolos fornecem uma maneira distribuída e dinâmica de resolver o problema de encontrar o caminho de menor custo na presença de falhas de enlace e nó e custos de borda variáveis. Observe a palavra " *distribuído"* na frase anterior; é difícil tornar soluções centralizadas

escaláveis, portanto, todos os protocolos de roteamento amplamente utilizados utilizam algoritmos distribuídos.

A natureza distribuída dos algoritmos de roteamento é um dos principais motivos pelos quais este tem sido um campo tão rico em pesquisa e desenvolvimento — há muitos desafios para fazer algoritmos distribuídos funcionarem bem. Por exemplo, algoritmos distribuídos levantam a possibilidade de que dois roteadores, em um instante, tenham ideias diferentes sobre o caminho mais curto para um destino. De fato, cada um pode pensar que o outro está mais próximo do destino e decidir enviar pacotes para o outro. Claramente, esses pacotes ficarão presos em um loop até que a discrepância entre os dois roteadores seja resolvida, e seria bom resolvê-la o mais rápido possível. Este é apenas um exemplo do tipo de problema que os protocolos de roteamento devem abordar.

Para iniciar nossa análise, assumimos que os custos de borda na rede são conhecidos. Examinaremos as duas principais classes de protocolos de roteamento: *vetor de distância* e *estado do enlace* . Em uma seção posterior, retornaremos ao problema de calcular os custos de borda de forma significativa.

3.4 2 Distância-Vetor (RIP)

A ideia por trás do algoritmo de vetor de distância é sugerida pelo seu nome. (O outro nome comum para essa classe de algoritmo é Bellman-Ford, em homenagem aos seus inventores.) Cada nó constrói uma matriz unidimensional (um vetor) contendo as "distâncias" (custos) para todos os outros nós e distribui esse vetor para seus vizinhos imediatos. A premissa inicial para o roteamento de vetor de distância é que cada nó conhece o custo do link para cada um de seus vizinhos diretamente conectados. Esses custos podem ser fornecidos quando o roteador é configurado por um gerenciador de rede. Um link inativo recebe um custo infinito.

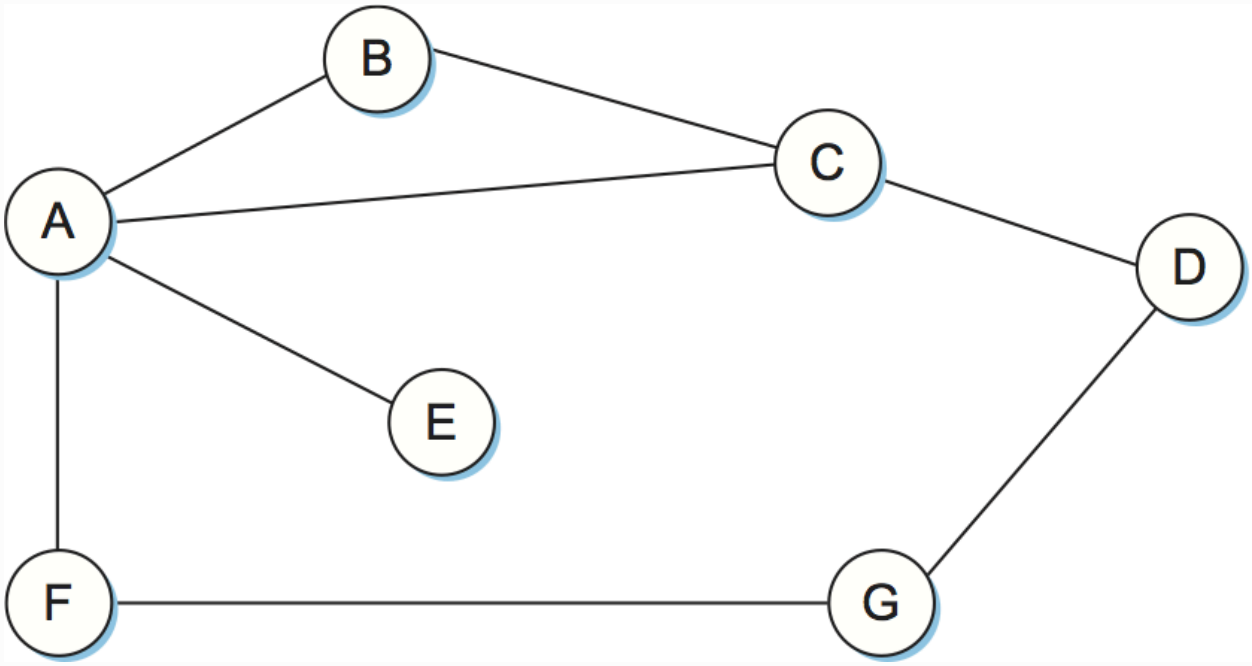


Figura 85. *Roteamento de vetor de distância: um exemplo de rede.*

	UM	B	C	D	E	F	G
UM	0	1	1	∞	1	1	∞
B	1	0	1	∞	∞	∞	∞
C	1	1	0	1	∞	∞	∞
D	∞	∞	1	0	∞	∞	1

E	1	∞	∞	∞	0	∞	∞
F	1	∞	∞	∞	∞	0	1
G	∞	∞	∞	1	∞	1	0

Para ver como um algoritmo de roteamento de vetor de distância funciona, é mais fácil considerar um exemplo como o mostrado na Figura 85. Neste exemplo, o custo de cada link é definido como 1, de modo que um caminho de menor custo é simplesmente aquele com o menor número de saltos. (Como todas as arestas têm o mesmo custo, não mostramos os custos no gráfico.) Podemos representar o conhecimento de cada nó sobre as distâncias para todos os outros nós como uma tabela como a Tabela 16. Observe que cada nó conhece apenas as informações em uma linha da tabela (aquela que leva seu nome na coluna da esquerda). A visão global apresentada aqui não está disponível em nenhum ponto único da rede.

Podemos considerar cada linha da Tabela 16 como uma lista de distâncias de um nó a todos os outros nós, representando as crenças atuais desse nó. Inicialmente, cada nó define um custo de 1 para seus vizinhos diretamente conectados e ∞ para todos os outros nós. Assim, A inicialmente acredita que pode alcançar B em um salto e que D é inalcançável. A tabela de roteamento armazenada em A reflete esse conjunto de crenças e inclui o nome do próximo salto que A usaria para alcançar qualquer nó alcançável. Inicialmente, então, a tabela de roteamento de A se pareceria com a Tabela 17 .

Destino	Custo	Próximo salto
B	1	B
C	1	C
D	∞	—
E	1	E
F	1	F
G	∞	—

O próximo passo no roteamento do vetor distância é que cada nó envia uma mensagem para seus vizinhos diretamente conectados contendo sua lista pessoal de distâncias. Por exemplo, o nó F diz ao nó A que ele pode alcançar o nó G a um custo de 1; A também sabe que pode alcançar F a um custo de 1, então ele soma esses custos para obter o custo de alcançar G por meio de F. Esse custo total de 2 é menor que o custo atual do infinito, então A registra que pode alcançar G a um custo de 2 passando por F. Similarmente, A aprende com C que D pode ser alcançado de C a um custo de 1; ele soma isso ao custo de alcançar C (1) e decide que D pode ser alcançado via C a um custo de 2, o que é melhor que o antigo custo do infinito. Ao

mesmo tempo, A aprende com C que B pode ser alcançado de C a um custo de 1, então ele conclui que o custo de alcançar B via C é 2. Como isso é pior que o custo atual de alcançar B (1), essa nova informação é ignorada. Neste ponto, A pode atualizar sua tabela de roteamento com custos e próximos saltos para todos os nós da rede. O resultado é mostrado na Tabela 18 .

Destino	Custo	Próximo salto
B	1	B
C	1	C
D	2	C
E	1	E
F	1	F
G	2	F

Na ausência de alterações na topologia, são necessárias apenas algumas trocas de informações entre vizinhos para que cada nó tenha uma tabela de roteamento completa. O processo de obtenção de informações de roteamento consistentes para

todos os nós é chamado de *convergência* . A Tabela 19 mostra o conjunto final de custos de cada nó para todos os outros nós quando o roteamento convergiu. Devemos enfatizar que não há um único nó na rede que tenha todas as informações desta tabela — cada nó conhece apenas o conteúdo de sua própria tabela de roteamento. A beleza de um algoritmo distribuído como este é que ele permite que todos os nós obtenham uma visão consistente da rede na ausência de qualquer autoridade centralizada.

	UM	**B**	**C**	**D**	**E**	**F**	**G**
UM	0	1	1	2	1	1	2
B	1	0	1	2	2	2	3
C	1	1	0	1	2	2	2
D	2	2	1	0	3	2	1
E	1	2	2	3	0	2	3
F	1	2	2	2	2	0	1

G	2	3	2	1	3	1	0

Há alguns detalhes a serem preenchidos antes de concluirmos nossa discussão sobre roteamento por vetor de distância. Primeiro, observamos que há duas circunstâncias diferentes nas quais um determinado nó decide enviar uma atualização de roteamento para seus vizinhos. Uma dessas circunstâncias é a atualização *periódica* . Nesse caso, cada nó envia automaticamente uma mensagem de atualização de tempos em tempos, mesmo que nada tenha mudado. Isso serve para informar aos outros nós que este nó ainda está em execução. Também garante que eles continuem recebendo informações de que podem precisar caso suas rotas atuais se tornem inviáveis. A frequência dessas atualizações periódicas varia de protocolo para protocolo, mas normalmente é da ordem de vários segundos a vários minutos. O segundo mecanismo, às vezes chamado de atualização *disparada* , ocorre sempre que um nó percebe uma falha de link ou recebe uma atualização de um de seus vizinhos que o faz alterar uma das rotas em sua tabela de roteamento. Sempre que a tabela de roteamento de um nó muda, ele envia uma atualização para seus vizinhos, o que pode levar a uma alteração em suas tabelas, fazendo com que eles enviem uma atualização para seus vizinhos.

Agora, considere o que acontece quando um enlace ou nó falha. Os nós que percebem primeiro enviam novas listas de distâncias para seus vizinhos e, normalmente, o sistema se estabiliza rapidamente para um novo estado. Quanto à questão de como um nó detecta uma falha, existem algumas respostas diferentes. Em uma abordagem, um nó testa continuamente o enlace para outro nó enviando um pacote de controle e verificando se ele recebe uma confirmação. Em outra abordagem, um nó determina que o enlace (ou o nó na outra extremidade do enlace) está inativo se não receber a atualização periódica de roteamento esperada nos últimos ciclos de atualização.

Para entender o que acontece quando um nó detecta uma falha de enlace, considere o que acontece quando F detecta que seu enlace com G falhou. Primeiro, F define sua nova distância até G como infinito e passa essa informação para A. Como A sabe que

seu caminho de 2 saltos até G passa por F, A também definiria sua distância até G como infinito. No entanto, com a próxima atualização de C, A descobriria que C tem um caminho de 2 saltos até G. Assim, A saberia que poderia alcançar G em 3 saltos passando por C, que é menor que infinito, e então A atualizaria sua tabela de acordo. Ao anunciar isso para F, o nó F descobriria que pode alcançar G a um custo de 4 saltos passando por A, que é menor que infinito, e o sistema se tornaria estável novamente.

Infelizmente, circunstâncias ligeiramente diferentes podem impedir a estabilização da rede. Suponha, por exemplo, que o link de A para E caia. Na próxima rodada de atualizações, A anuncia uma distância de infinito para E, mas B e C anunciam uma distância de 2 para E. Dependendo do momento exato dos eventos, o seguinte pode acontecer: o nó B, ao ouvir que E pode ser alcançado em 2 saltos de C, conclui que pode alcançar E em 3 saltos e anuncia isso para A; o nó A conclui que pode alcançar E em 4 saltos e anuncia isso para C; o nó C conclui que pode alcançar E em 5 saltos; e assim por diante. Esse ciclo para somente quando as distâncias atingem um número grande o suficiente para ser considerado infinito. Enquanto isso, nenhum dos nós realmente sabe que E é inalcançável, e as tabelas de roteamento da rede não se estabilizam. Essa situação é conhecida como o problema *da contagem até o infinito .*

Existem várias soluções parciais para este problema. A primeira é usar um número relativamente pequeno como aproximação do infinito. Por exemplo, podemos decidir que o número máximo de saltos para atravessar uma determinada rede nunca será superior a 15 e, portanto, podemos escolher 16 como o valor que representa o infinito. Isso pelo menos limita o tempo que leva para contar até o infinito. É claro que também poderia representar um problema se nossa rede crescesse a um ponto em que alguns nós estivessem separados por mais de 15 saltos.

Uma técnica para melhorar o tempo para estabilizar o roteamento é chamada de *horizonte dividido* . A ideia é que quando um nó envia uma atualização de roteamento para seus vizinhos, ele não envia as rotas que aprendeu de cada vizinho de volta para aquele vizinho. Por exemplo, se B tem a rota (E, 2, A) em sua tabela, então ele sabe que deve ter aprendido essa rota de A e, portanto, sempre que B envia uma

atualização de roteamento para A, ele não inclui a rota (E, 2) nessa atualização. Em uma variação mais forte do horizonte dividido, chamada *horizonte dividido com reversão de veneno* , B na verdade envia essa rota de volta para A, mas coloca informações negativas na rota para garantir que A não usará B para chegar a E. Por exemplo, B envia a rota (E, ∞) para A. O problema com ambas as técnicas é que elas só funcionam para loops de roteamento que envolvem dois nós. Para loops de roteamento maiores, medidas mais drásticas são necessárias. Continuando o exemplo acima, se B e C tivessem esperado um pouco depois de saber da falha do link de A antes de anunciar rotas para E, eles teriam descoberto que nenhum deles realmente tinha uma rota para E. Infelizmente, essa abordagem atrasa a convergência do protocolo; a velocidade de convergência é uma das principais vantagens de seu concorrente, o roteamento link-state, assunto de uma seção posterior.

Implementação

O código que implementa esse algoritmo é bastante simples; apresentamos aqui apenas alguns dos conceitos básicos. A estrutura `Route`define cada entrada na tabela de roteamento, e a constante `MAX_TTL`especifica por quanto tempo uma entrada é mantida na tabela antes de ser descartada.

```
#define MAX_ROUTES      128     /* maximum size of routing table */
#define MAX_TTL         120     /* time (in seconds) until route expires */

typedef struct {
    NodeAddr  Destination;    /* address of destination */
    NodeAddr  NextHop;        /* address of next hop */
    int        Cost;          /* distance metric */
    u_short   TTL;            /* time to live */
} Route;

int       numRoutes = 0;
Route     routingTable[MAX_ROUTES];
```

A rotina que atualiza a tabela de roteamento do nó local com base em uma nova rota é dada por `mergeRoute`. Embora não seja mostrada, uma função de temporizador verifica periodicamente a lista de rotas na tabela de roteamento do nó, decrementa o `TTL`campo

(tempo de vida) de cada rota e descarta quaisquer rotas com tempo de vida 0. Observe, no entanto, que o `TTL`campo é redefinido para `MAX_TTL`sempre que a rota for reconfirmada por uma mensagem de atualização de um nó vizinho.

```
void
mergeRoute (Route *new)
{
    int i;

    for (i = 0; i < numRoutes; ++i)
    {
        if (new->Destination == routingTable[i].Destination)
        {
            if (new->Cost + 1 < routingTable[i].Cost)
            {
                /* found a better route: */
                break;
            } else if (new->NextHop == routingTable[i].NextHop) {
                /* metric for current next-hop may have changed: */
                break;
            } else {
                /* route is uninteresting---just ignore it */
                return;
            }
        }
    }
    if (i == numRoutes)
    {
        /* this is a completely new route; is there room for it? */
        if (numRoutes < MAXROUTES)
        {
            ++numRoutes;
        } else {
            /* can`t fit this route in table so give up */
            return;
        }
    }
    routingTable[i] = *new;
    /* reset TTL */
    routingTable[i].TTL = MAX_TTL;
    /* account for hop to get to next node */
    ++routingTable[i].Cost;
}
```

Por fim, o procedimento `updateRoutingTable`é a rotina principal que chama `mergeRoute`para incorporar todas as rotas contidas em uma atualização de roteamento recebida de um nó vizinho.

```
void
updateRoutingTable (Route *newRoute, int numNewRoutes)
{
    int i;

    for (i=0; i < numNewRoutes; ++i)
    {
        mergeRoute(&newRoute[i]);
    }
}
```

Protocolo de Informações de Roteamento (RIP)

Um dos protocolos de roteamento mais utilizados em redes IP é o Routing Information Protocol (RIP). Seu amplo uso nos primórdios do IP deveu-se, em grande parte, ao fato de ter sido distribuído junto com a popular versão Berkeley Software Distribution (BSD) do Unix, da qual muitas versões comerciais do Unix foram derivadas. Ele também é extremamente simples. O RIP é o exemplo canônico de um protocolo de roteamento baseado no algoritmo de vetor de distância que acabamos de descrever.

Os protocolos de roteamento em redes interconectadas diferem muito pouco do modelo de grafo idealizado descrito acima. Em uma rede interconectada, o objetivo dos roteadores é aprender como encaminhar pacotes para várias *redes* . Assim, em vez de anunciar o custo de alcançar outros roteadores, os roteadores anunciam o custo de alcançar redes. Por exemplo, na Figura 86 , o roteador C anunciaria ao roteador A o fato de que ele pode alcançar as redes 2 e 3 (às quais está diretamente conectado) a um custo 0, as redes 5 e 6 a um custo 1 e a rede 4 a um custo 2.

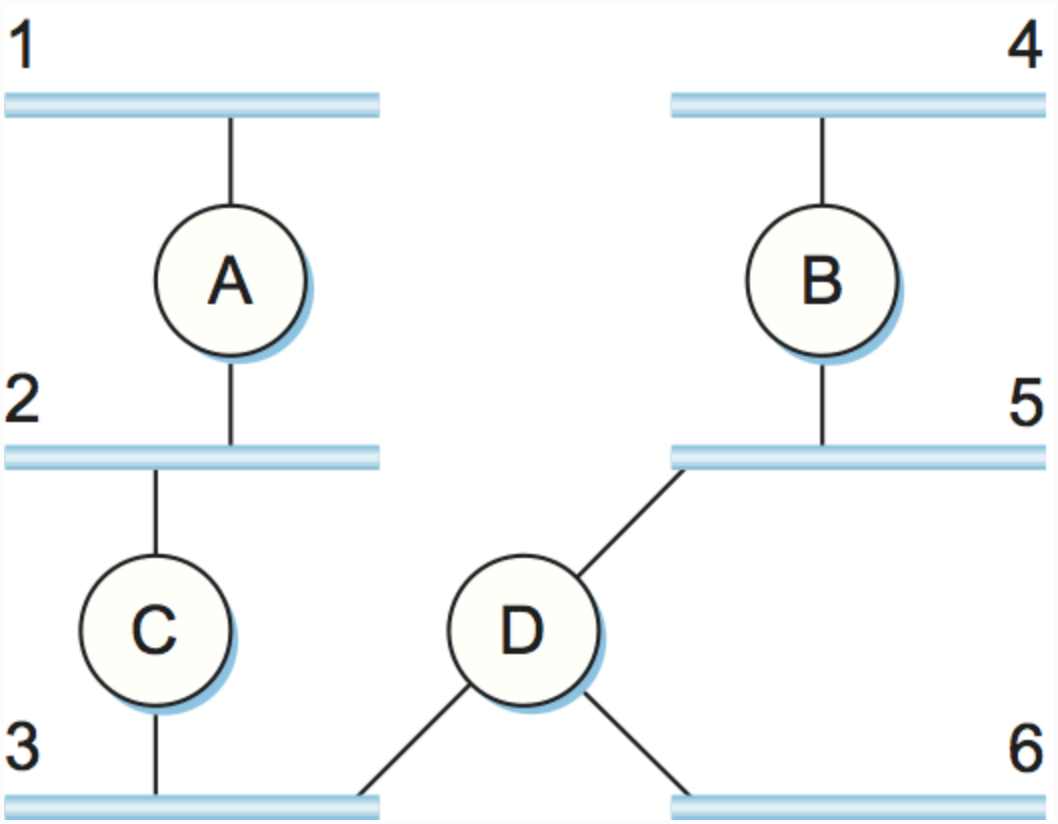


Figura 86. *Exemplo de rede executando RIP.*

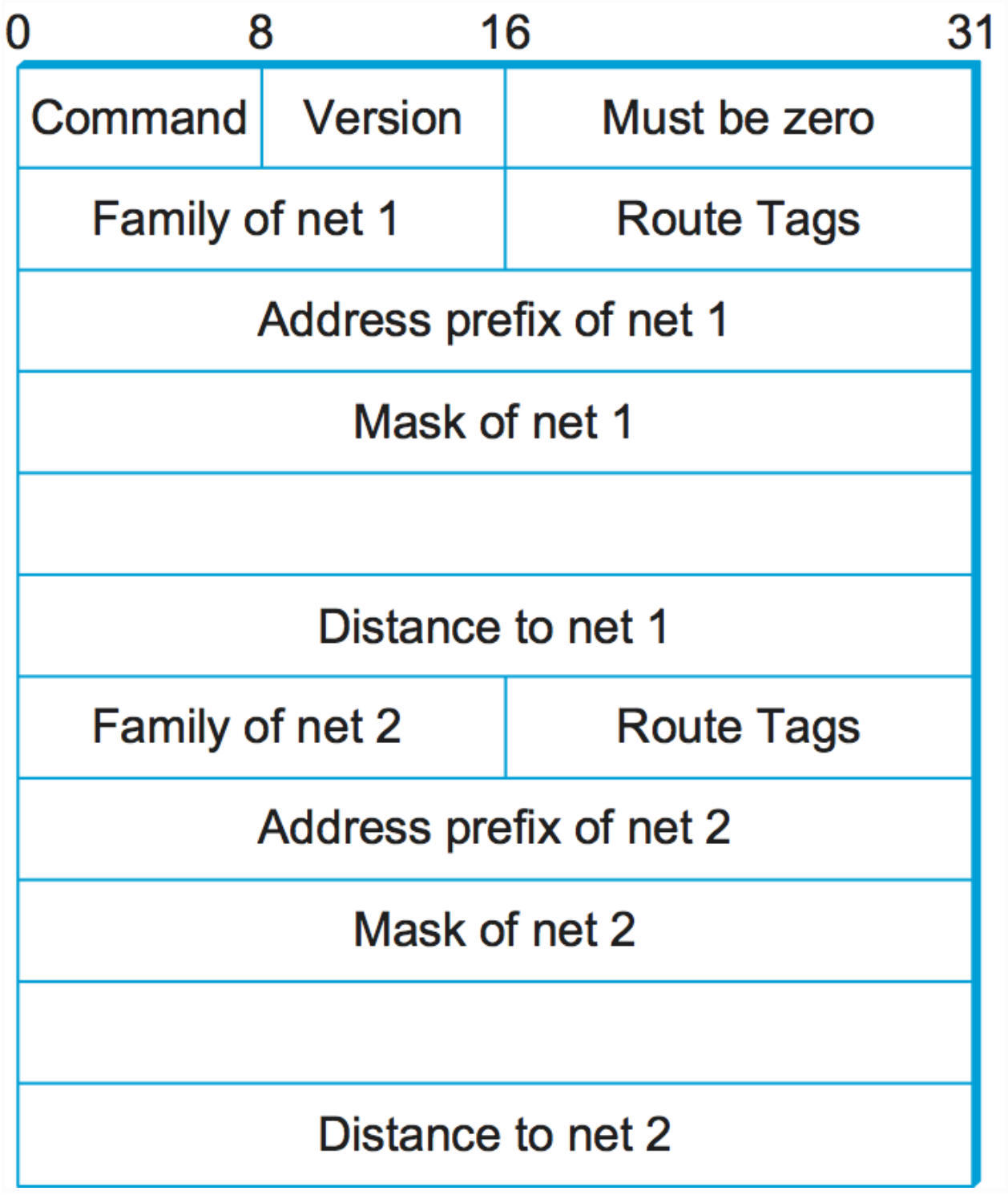


Figura 87. *Formato do pacote RIPv2.*

Podemos ver evidências disso no formato de pacote RIP (versão 2) na Figura 87. A maior parte do pacote é ocupada por triplos. No entanto, os princípios do algoritmo de roteamento são os mesmos. Por exemplo, se o roteador A aprende com o roteador B

que a rede X pode ser alcançada a um custo menor via B do que via o próximo salto existente na tabela de roteamento, A atualiza as informações de custo e próximo salto para o número da rede de acordo.`(address, mask, distance)`

O RIP é, na verdade, uma implementação bastante direta do roteamento por vetor de distância. Roteadores que executam o RIP enviam seus anúncios a cada 30 segundos; um roteador também envia uma mensagem de atualização sempre que uma atualização de outro roteador o faz alterar sua tabela de roteamento. Um ponto interessante é que ele suporta múltiplas famílias de endereços, não apenas IP — essa é a razão para a `Family`parte dos anúncios. O RIP versão 2 (RIPv2) também introduziu as máscaras de sub-rede descritas em uma seção anterior, enquanto o RIP versão 1 funcionava com os antigos endereços IP com classes.

Como veremos a seguir, é possível usar uma variedade de métricas ou custos diferentes para os links em um protocolo de roteamento. O RIP adota a abordagem mais simples, com todos os custos de link sendo iguais a 1, exatamente como no nosso exemplo acima. Assim, ele sempre tenta encontrar a rota de menor salto. As distâncias válidas vão de 1 a 15, com 16 representando infinito. Isso também limita o RIP à execução em redes relativamente pequenas — aquelas sem caminhos com mais de 15 saltos.

3.4.3 Estado do Link (OSPF)

O roteamento link-state é a segunda classe principal de protocolos de roteamento intradomínio. As premissas iniciais para o roteamento link-state são bastante semelhantes às do roteamento por vetor distância. Supõe-se que cada nó seja capaz de descobrir o estado do enlace para seus vizinhos (ativo ou inativo) e o custo de cada enlace. Novamente, queremos fornecer a cada nó informações suficientes para que ele encontre o caminho de menor custo para qualquer destino. A ideia básica por trás dos protocolos link-state é muito simples: cada nó sabe como alcançar seus vizinhos diretamente conectados e, se garantirmos que a totalidade desse conhecimento seja disseminada para cada nó, cada nó terá conhecimento suficiente da rede para construir

um mapa completo da rede. Esta é claramente uma condição suficiente (embora não necessária) para encontrar o caminho mais curto para qualquer ponto da rede. Assim, os protocolos de roteamento link-state baseiam-se em dois mecanismos: a disseminação confiável de informações de link-state e o cálculo de rotas a partir da soma de todo o conhecimento acumulado sobre link-state.

Inundações confiáveis

Inundação confiável é o processo que garante que todos os nós participantes do protocolo de roteamento recebam uma cópia das informações de estado do link de todos os outros nós. Como o termo *inundação* sugere, a ideia básica é que um nó envie suas informações de estado do link para todos os seus links diretamente conectados; cada nó que recebe essas informações as encaminha para todos os *seus* links. Esse processo continua até que as informações cheguem a todos os nós da rede.

Mais precisamente, cada nó cria um pacote de atualização, também chamado de *pacote de estado de link* (LSP), que contém as seguintes informações:

- O ID do nó que criou o LSP
- Uma lista de vizinhos diretamente conectados daquele nó, com o custo do link para cada um
- Um número de sequência
- Um tempo para viver este pacote

Os dois primeiros itens são necessários para permitir o cálculo da rota; os dois últimos são usados para tornar confiável o processo de inundação do pacote para todos os nós. Confiabilidade inclui garantir que você tenha a cópia mais recente das informações, pois pode haver múltiplos LSPs contraditórios de um nó atravessando a rede. Tornar a inundação confiável tem se mostrado bastante difícil. (Por exemplo, uma versão inicial do roteamento link-state usada na ARPANET causou a falha dessa rede em 1981.)

A inundação funciona da seguinte maneira. Primeiro, a transmissão de LSPs entre roteadores adjacentes é tornada confiável por meio de confirmações e retransmissões, assim como no protocolo de camada de enlace confiável. No entanto, várias outras etapas são necessárias para inundar de forma confiável um LSP para todos os nós de uma rede.

Considere um nó X que recebe uma cópia de um LSP originário de outro nó Y. Observe que Y pode ser qualquer outro roteador no mesmo domínio de roteamento que X. X verifica se já armazenou uma cópia de um LSP de Y. Caso contrário, armazena o LSP. Se já tiver uma cópia, compara os números de sequência; se o novo LSP tiver um número de sequência maior, presume-se que seja o mais recente e que o LSP seja armazenado, substituindo o antigo. Um número de sequência menor (ou igual) implicaria um LSP mais antigo (ou não mais recente) do que o armazenado, portanto, seria descartado e nenhuma ação adicional seria necessária. Se o LSP recebido for o mais recente, X envia uma cópia desse LSP para todos os seus vizinhos, exceto para o vizinho do qual o LSP acabou de ser recebido. O fato de o LSP não ser enviado de volta para o nó do qual foi recebido ajuda a acabar com a inundação de um LSP. Como X passa o LSP para todos os seus vizinhos, que então fazem a mesma coisa, a cópia mais recente do LSP eventualmente alcança todos os nós.

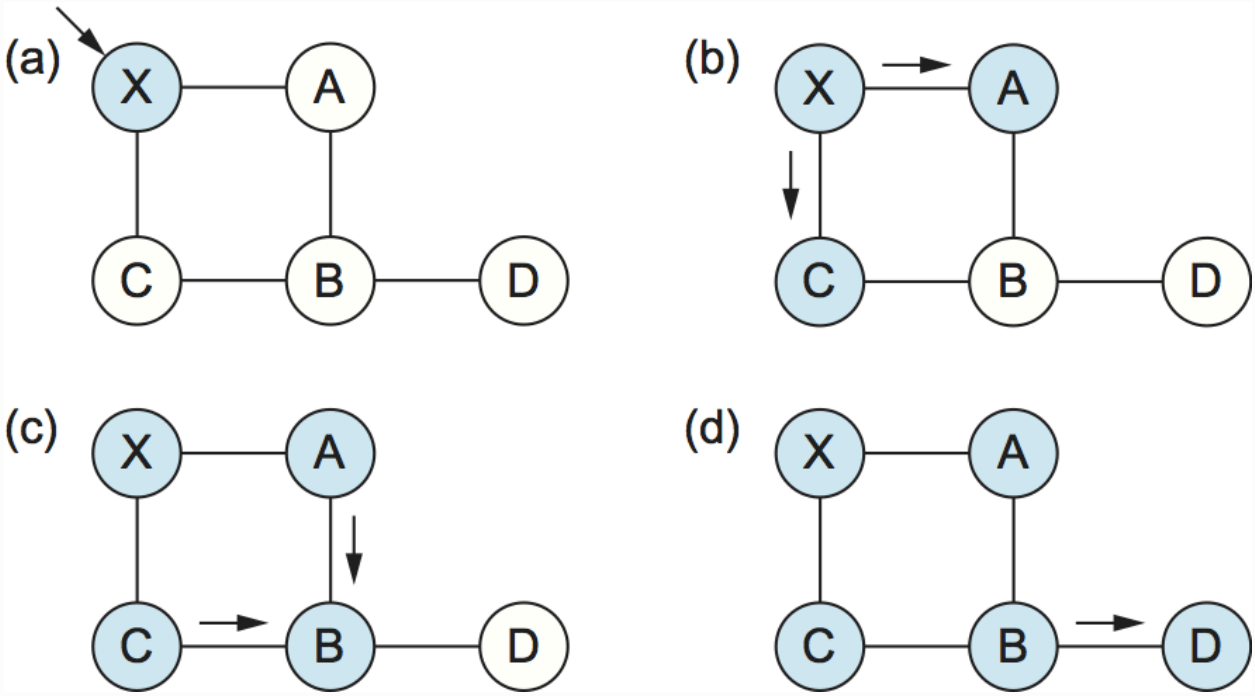


Figura 88. *Inundação de pacotes de estado de link: (a) LSP chega ao nó X; (b) X inunda LSP para A e C; (c) A e C inundam LSP para B (mas não para X); (d) a inundação é concluída.*

A Figura 88 mostra um LSP sendo inundado em uma pequena rede. Cada nó fica sombreado à medida que armazena o novo LSP. Na Figura 88(a), o LSP chega ao nó X, que o envia aos vizinhos A e C na Figura 88(b) . A e C não o enviam de volta para X, mas sim para B. Como B recebe duas cópias idênticas do LSP, ele aceitará a que chegar primeiro e ignorará a segunda como duplicata. Em seguida, ele passa o LSP para D, que não tem vizinhos para inundá-lo, e o processo é concluído.

Assim como no RIP, cada nó gera LSPs em duas circunstâncias. A expiração de um temporizador periódico ou uma mudança na topologia pode fazer com que um nó gere um novo LSP. No entanto, a única razão baseada na topologia para um nó gerar um LSP é se um de seus enlaces diretamente conectados ou vizinhos imediatos tiver caído. A falha de um enlace pode ser detectada em alguns casos pelo protocolo da camada de enlace. A queda de um vizinho ou a perda de conectividade com esse vizinho pode ser detectada usando pacotes periódicos de "olá". Cada nó os envia para seus vizinhos imediatos em intervalos definidos. Se passar um tempo suficientemente

longo sem o recebimento de um "olá" de um vizinho, o enlace para esse vizinho será declarado inativo e um novo LSP será gerado para refletir esse fato.

Um dos objetivos importantes do projeto do mecanismo de inundação de um protocolo link-state é que as informações mais recentes sejam inundadas para todos os nós o mais rápido possível, enquanto as informações antigas devem ser removidas da rede e impedidas de circular. Além disso, é claramente desejável minimizar a quantidade total de tráfego de roteamento que circula pela rede; afinal, isso representa apenas uma sobrecarga da perspectiva daqueles que realmente usam a rede para suas aplicações. Os próximos parágrafos descrevem algumas das maneiras pelas quais esses objetivos são alcançados.

Uma maneira fácil de reduzir a sobrecarga é evitar a geração de LSPs, a menos que seja absolutamente necessário. Isso pode ser feito usando temporizadores muito longos — geralmente da ordem de horas — para a geração periódica de LSPs. Dado que o protocolo de inundação é realmente confiável quando a topologia muda, é seguro presumir que mensagens dizendo "nada mudou" não precisam ser enviadas com muita frequência.

Para garantir que informações antigas sejam substituídas por informações mais recentes, os LSPs carregam números de sequência. Cada vez que um nó gera um novo LSP, ele incrementa o número de sequência em 1. Ao contrário da maioria dos números de sequência usados em protocolos, não se espera que esses números de sequência sejam encapsulados, então o campo precisa ser bem grande (digamos, 64 bits). Se um nó cair e depois voltar a funcionar, ele começa com um número de sequência 0. Se o nó ficou inativo por um longo tempo, todos os LSPs antigos para aquele nó terão expirado (conforme descrito abaixo); caso contrário, este nó eventualmente receberá uma cópia de seu próprio LSP com um número de sequência maior, que ele pode então incrementar e usar como seu próprio número de sequência. Isso garantirá que seu novo LSP substitua qualquer um de seus LSPs antigos restantes de antes da queda do nó.

Os LSPs também possuem um tempo de vida (time to live). Isso é usado para garantir que informações antigas de estado do link sejam eventualmente removidas da rede. Um nó sempre diminui o TTL de um LSP recém-recebido antes de inundá-lo para seus vizinhos. Ele também "envelhece" o LSP ao longo do tempo enquanto ele é armazenado no nó. Quando o TTL chega a 0, o nó inunda novamente o LSP (com o TTL de 0), o que é interpretado por todos os nós da rede como um sinal para excluir esse LSP.

Cálculo de Rota

Uma vez que um determinado nó tenha uma cópia do LSP de todos os outros nós, ele é capaz de calcular um mapa completo da topologia da rede e, a partir desse mapa, decidir a melhor rota para cada destino. A questão, então, é exatamente como ele calcula as rotas a partir dessas informações. A solução se baseia em um algoritmo bem conhecido da teoria dos grafos: o algoritmo do caminho mais curto de Dijkstra.

Primeiro, definimos o algoritmo de Dijkstra em termos de teoria dos grafos. Imagine que um nó recebe todos os LSPs que recebeu e constrói uma representação gráfica da rede, na qual N denota o conjunto de nós no grafo, l(i,j) denota o custo não negativo (peso) associado à aresta entre os nós i, j em N e l(i, j) = ∞ se nenhuma aresta conecta i e j. Na descrição a seguir, deixamos s em N denotar este nó, ou seja, o nó que executa o algoritmo para encontrar o caminho mais curto para todos os outros nós em N. Além disso, o algoritmo mantém as duas variáveis a seguir: M denota o conjunto de nós incorporados até o momento pelo algoritmo, e C(n) denota o custo do caminho de s para cada nó n. Dadas essas definições, o algoritmo é definido da seguinte forma:

```
M = {s}
for each n in N - {s}
    C(n) = l(s,n)
while (N != M)
    M = M + {w} such that C(w) is the minimum for all w in (N-M)
    for each n in (N-M)
    C(n) = MIN(C(n), C(w)+l(w,n))
```

Basicamente, o algoritmo funciona da seguinte maneira. Começamos com M contendo este nó s e então inicializamos a tabela de custos (o array `C(n)`) para outros nós usando os custos conhecidos para nós diretamente conectados. Então procuramos o nó que é alcançável com o menor custo (w) e o adicionamos a M. Finalmente, atualizamos a tabela de custos considerando o custo de alcançar os nós através de w. Na última linha do algoritmo, escolhemos uma nova rota para o nó n que passa pelo nó w se o custo total de ir da origem até w e então seguir o link de w até n for menor que a rota antiga que tínhamos para n. Este procedimento é repetido até que todos os nós sejam incorporados em M.

Na prática, cada switch calcula sua tabela de roteamento diretamente a partir dos LSPs coletados, usando uma implementação do algoritmo de Dijkstra, chamado algoritmo *de busca direta* . Especificamente, cada switch mantém duas listas, conhecidas como `Tentative`e `Confirmed`. Cada uma dessas listas contém um conjunto de entradas no formato . O algoritmo funciona da seguinte forma:`(Destination, Cost, NextHop)`

1. Inicialize a `Confirmed`lista com uma entrada para mim; esta entrada tem um custo de 0.
2. Para o nó adicionado à `Confirmed`lista na etapa anterior, chame-o de nó `Next`e selecione seu LSP.
3. Para cada vizinho (`Neighbor`) de `Next`, calcule o custo (`Cost`) para chegar a este `Neighbor`como a soma do custo de mim para `Next`e de `Next`para `Neighbor`.
    1. Se `Neighbor`atualmente não estiver nem na `Confirmed`nem na `Tentative`lista, então adicione à lista, onde está a direção que devo seguir para chegar lá .`(Neighbor, Cost, NextHop)``Tentative``NextHop``Next`
    2. Se `Neighbor`estiver atualmente na `Tentative`lista e `Cost`for menor que o custo listado atualmente para `Neighbor`, então substitua a entrada atual por , onde é a direção que devo seguir para chegar a .`(Neighbor, Cost, NextHop)``NextHop``Next`

4. Se a `Tentative`lista estiver vazia, pare. Caso contrário, escolha a entrada da `Tentative`lista com o menor custo, mova-a para a `Confirmed`lista e volte para a etapa 2.

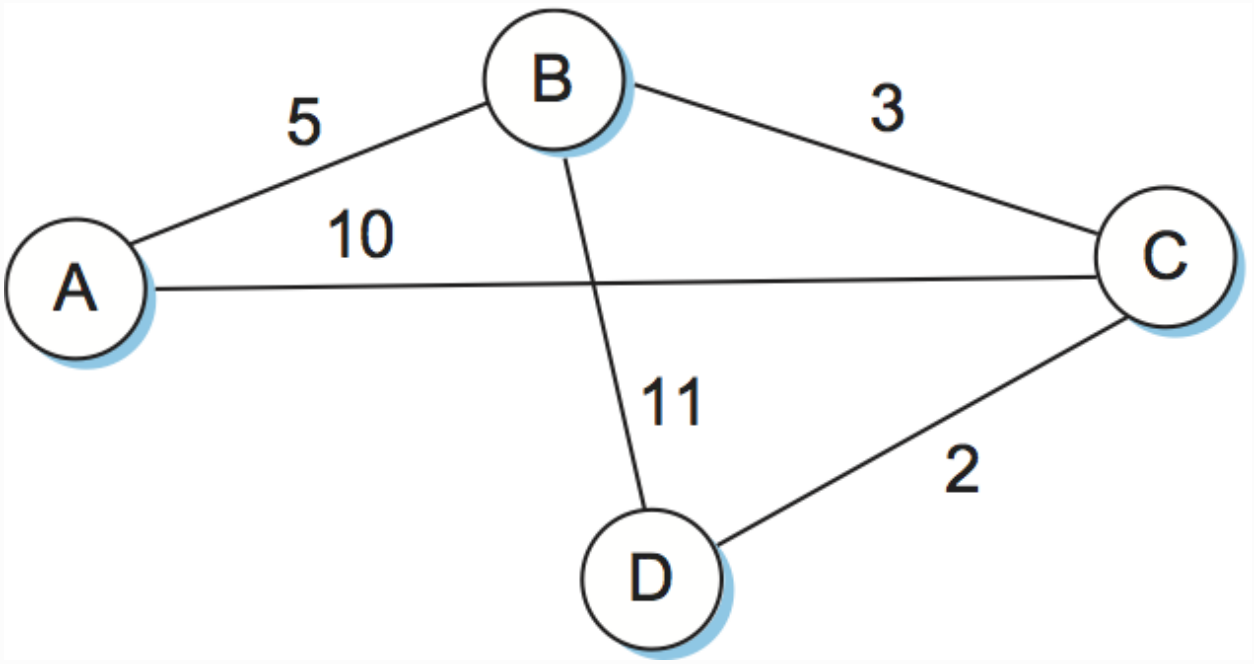


Figura 89. *Roteamento link-state: um exemplo de rede.*

Isso ficará muito mais fácil de entender quando analisarmos um exemplo. Considere a rede representada na Figura 89. Observe que, diferentemente do nosso exemplo anterior, esta rede possui uma gama de custos de borda diferentes. A Tabela 20 traça as etapas para a construção da tabela de roteamento para o nó D. Denotamos as duas saídas de D usando os nomes dos nós aos quais elas se conectam, B e C. Observe como o algoritmo parece seguir pistas falsas (como o caminho de custo de 11 unidades para B, que foi a primeira adição à `Tentative`lista), mas termina com os caminhos de menor custo para todos os nós.

Etapa	Confirmado	Tentativo	Comentários

1	(D,0,–)		Como D é o único novo membro da lista confirmada, observe seu LSP.
2	(D,0,–)	(B,11,B) (C,2,C)	O LSP de D diz que podemos chegar a B através de B ao custo 11, o que é melhor do que qualquer outra coisa em qualquer lista, então coloque-o na `Tentative` lista; o mesmo para C.
3	(D,0,–) (C,2,C)	(B,11,B)	Coloque o membro de menor custo de `Tentative`(C) na `Confirmed` lista. Em seguida, examine o LSP do membro recém-confirmado (C).
4	(D,0,–) (C,2,C)	(B,5,C) (A,12,C)	O custo para chegar a B através de C é 5, então substitua (B,11,B). O LSP de C nos diz que podemos chegar a A ao custo 12.
5	(D,0,–) (C,2,C) (B,5,C)	(A,12,C)	Mova o membro de menor custo de `Tentative`(B) para

			`Confirmed`e observe seu LSP.
6	(D,0,–) (C,2,C) (B,5,C)	(A,10,C)	Como podemos chegar a A com custo 5 através de B, substitua a `Tentative` entrada.
7	(D,0,–) (C,2,C) (B,5,C) (A,10,C)		Mova o membro de menor custo de `Tentative`(A) para `Confirmed`, e estamos prontos.

O algoritmo de roteamento link-state possui muitas propriedades interessantes: comprovadamente, estabiliza-se rapidamente, gera pouco tráfego e responde rapidamente a alterações de topologia ou falhas de nós. A desvantagem é que a quantidade de informações armazenadas em cada nó (um LSP para cada nó da rede) pode ser bastante grande. Este é um dos problemas fundamentais do roteamento e é uma instância do problema mais geral da escalabilidade. Algumas soluções para o problema específico (a quantidade de armazenamento potencialmente necessária em cada nó) e para o problema geral (escalabilidade) serão discutidas na próxima seção.

Conclusão principal

Vetor de distância e estado de enlace são algoritmos de roteamento distribuídos, mas adotam estratégias diferentes. No vetor de distância, cada nó se comunica apenas com seus vizinhos diretamente conectados, mas informa a eles tudo o que aprendeu (ou seja, a distância até todos os nós). No estado de enlace, cada nó se comunica com todos os outros nós, mas informa a eles apenas o que sabe com certeza (ou seja,

apenas o estado de seus enlaces diretamente conectados). Em contraste com ambos os algoritmos, consideraremos uma abordagem mais centralizada para o roteamento na Seção 3.5, quando introduzirmos Redes Definidas por Software (SDN). [Próximo]

O Protocolo de Caminho Mais Curto Aberto Primeiro (OSPF)

Um dos protocolos de roteamento link-state mais utilizados é o OSPF. A primeira palavra, "Aberto", refere-se ao fato de ser um padrão aberto e não proprietário, criado sob os auspícios da Internet Engineering Task Force (IETF). A parte "SPF" vem de um nome alternativo para roteamento link-state. O OSPF adiciona diversos recursos ao algoritmo básico de link-state descrito acima, incluindo os seguintes:

- *Autenticação de mensagens de roteamento* — Uma característica dos algoritmos de roteamento distribuído é que eles dispersam informações de um nó para muitos outros, e toda a rede pode, portanto, ser impactada por informações incorretas de um nó. Por esse motivo, é uma boa ideia garantir que todos os nós que participam do protocolo sejam confiáveis. A autenticação de mensagens de roteamento ajuda a alcançar esse objetivo. As primeiras versões do OSPF usavam uma senha simples de 8 bytes para autenticação. Esta não é uma forma de autenticação forte o suficiente para impedir usuários mal-intencionados dedicados, mas alivia alguns problemas causados por configurações incorretas ou ataques casuais. (Uma forma semelhante de autenticação foi adicionada ao RIP na versão 2.) A autenticação criptográfica forte foi adicionada posteriormente.
- *Hierarquia adicional* — A hierarquia é uma das ferramentas fundamentais usadas para tornar os sistemas mais escaláveis. O OSPF introduz outra camada de hierarquia no roteamento, permitindo que um domínio seja particionado em *áreas* . Isso significa que um roteador dentro de um domínio não precisa necessariamente saber como acessar todas as redes dentro desse domínio — ele pode conseguir isso sabendo apenas como acessar a área correta. Assim, há uma redução na quantidade de informações que precisam ser transmitidas e armazenadas em cada nó.

- *Balanceamento de carga* — O OSPF permite que várias rotas para o mesmo local recebam o mesmo custo e fará com que o tráfego seja distribuído uniformemente entre essas rotas, fazendo melhor uso da capacidade de rede disponível.

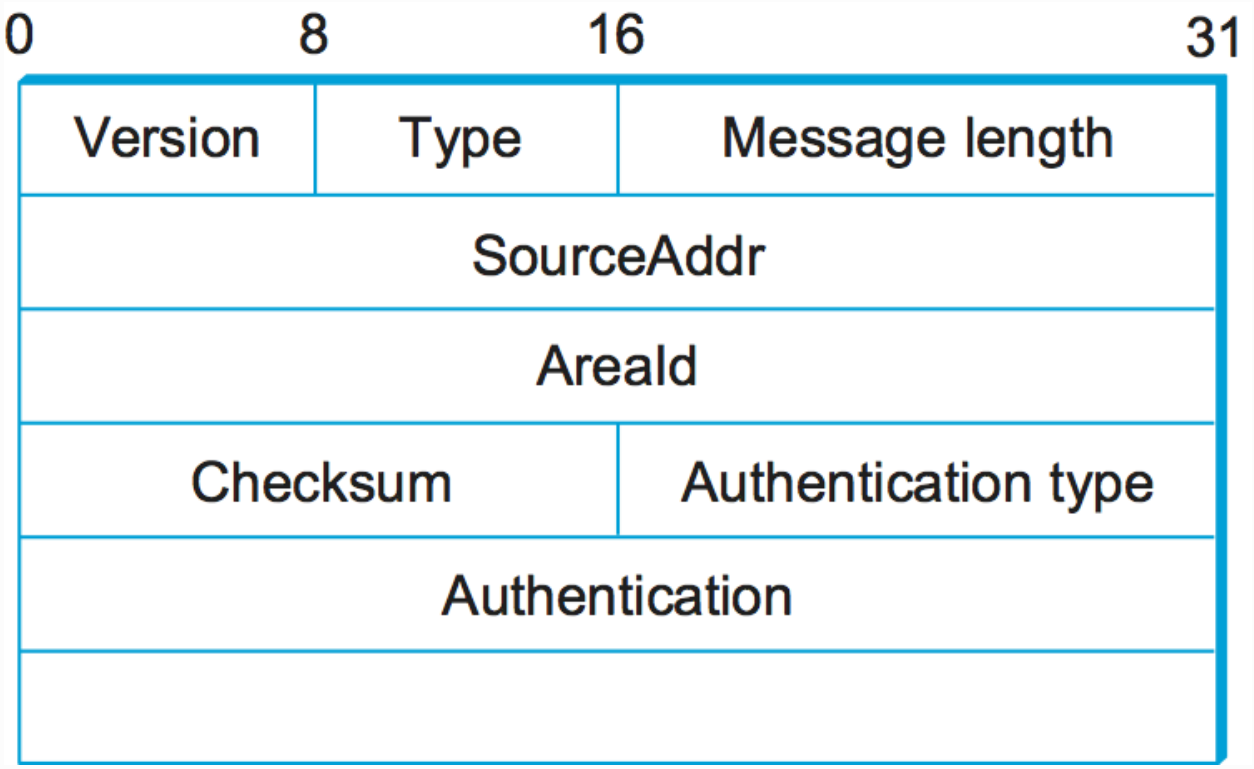


Figura 90. *Formato do cabeçalho OSPF.*

Existem vários tipos diferentes de mensagens OSPF, mas todas começam com o mesmo cabeçalho, como mostrado na Figura 90. O `Version`campo está atualmente definido como 2 e `Type`pode assumir os valores de 1 a 5. O `SourceAddr`identifica o remetente da mensagem e o `AreaId`é um identificador de 32 bits da área em que o nó está localizado. O pacote inteiro, exceto os dados de autenticação, é protegido por uma soma de verificação de 16 bits usando o mesmo algoritmo do cabeçalho IP. O é 0 se nenhuma autenticação for usada; caso contrário, pode ser 1, implicando que uma senha simples é usada, ou 2, que indica que uma soma de verificação de autenticação criptográfica é usada. Nos últimos casos, o campo carrega a senha ou a soma de verificação criptográfica.`Authentication typeAuthentication`

Dos cinco tipos de mensagens OSPF, o tipo 1 é a mensagem "hello", que um roteador envia aos seus pares para notificá-los de que ainda está ativo e conectado, conforme descrito acima. Os tipos restantes são usados para solicitar, enviar e confirmar o recebimento de mensagens de estado do link. O bloco de construção básico das mensagens de estado do link no OSPF é o anúncio de estado do link (LSA). Uma mensagem pode conter vários LSAs. Fornecemos alguns detalhes sobre o LSA aqui.

Como qualquer protocolo de roteamento de interconexão de redes, o OSPF deve fornecer informações sobre como alcançar as redes. Portanto, o OSPF deve fornecer um pouco mais de informações do que o protocolo simples baseado em gráficos descrito acima. Especificamente, um roteador executando OSPF pode gerar pacotes de estado de link que anunciam uma ou mais redes diretamente conectadas a esse roteador. Além disso, um roteador conectado a outro roteador por algum link deve anunciar o custo de alcançar esse roteador por meio do link. Esses dois tipos de anúncios são necessários para permitir que todos os roteadores em um domínio determinem o custo de alcançar todas as redes nesse domínio e o próximo salto apropriado para cada rede.

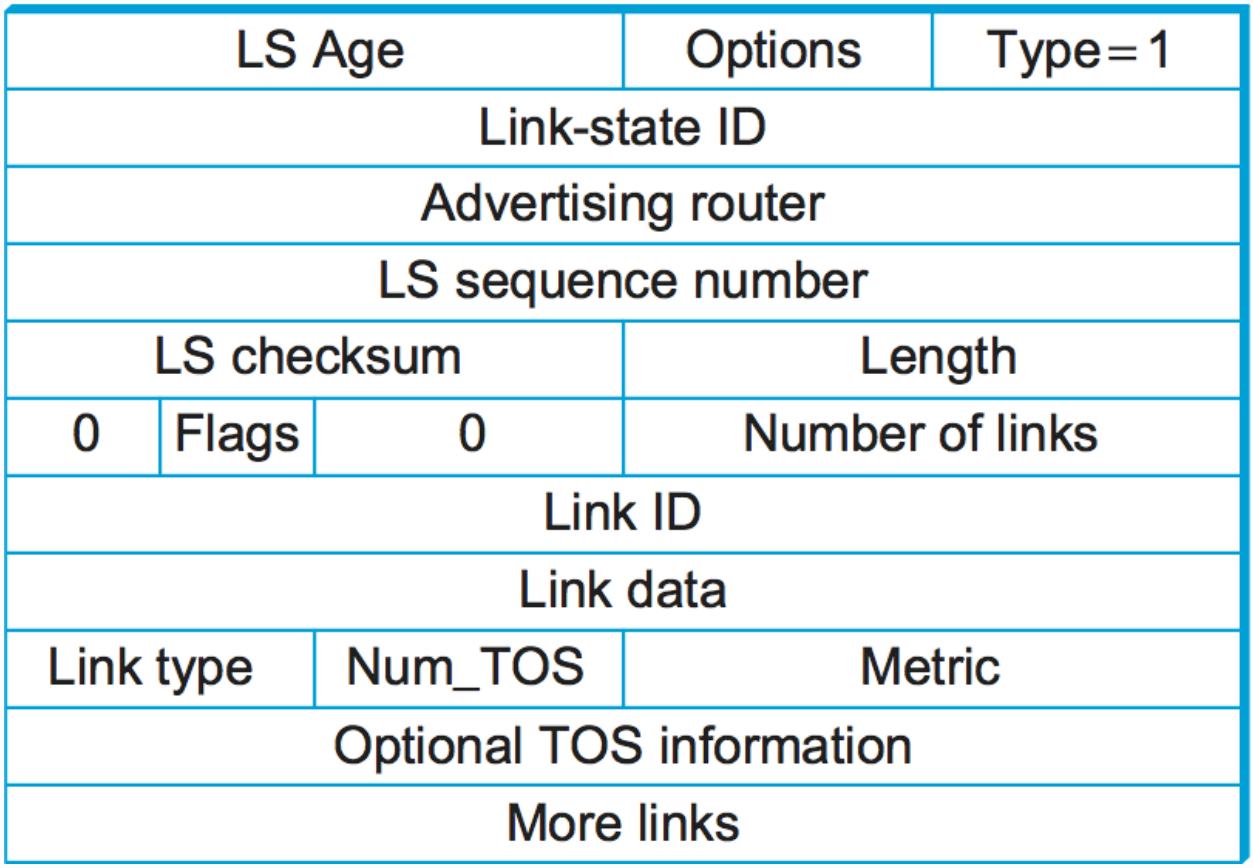


Figura 91. *Anúncio de estado de link OSPF.*

A Figura 91 mostra o formato do pacote para um anúncio de estado de link do tipo 1. LSAs do tipo 1 anunciam o custo dos links entre roteadores. LSAs do tipo 2 são usados para anunciar redes às quais o roteador de anúncio está conectado, enquanto outros tipos são usados para oferecer suporte a hierarquia adicional, conforme descrito na próxima seção. Muitos campos no LSA devem ser familiares da discussão anterior. O campoé equivalente a um tempo de vida, exceto que ele conta progressivamente e o LSA expira quando a idade atinge um valor máximo definido. Ocampo nos diz que este é um LSA do tipo 1.`LS AgeType`

Em um LSA tipo 1, os campos e são idênticos. Cada um carrega um identificador de 32 bits para o roteador que criou este LSA. Embora diversas estratégias de atribuição possam ser usadas para atribuir esse ID, é essencial que ele seja único no domínio de roteamento e que um determinado roteador use consistentemente o mesmo ID de roteador. Uma maneira de escolher um ID de roteador que atenda a esses requisitos

seria escolher o menor endereço IP entre todos os endereços IP atribuídos a esse roteador. (Lembre-se de que um roteador pode ter um endereço IP diferente em cada uma de suas interfaces.)`Link state IDAdvertising router`

O é usado exatamente como descrito acima para detectar LSAs antigos ou duplicados. O é semelhante a outros que vimos em outros protocolos; é, obviamente, usado para verificar se os dados não foram corrompidos. Ele abrange todos os campos do pacote, exceto , portanto, não é necessário recalcular uma soma de verificação toda vez que é incrementado. é o comprimento em bytes do LSA completo.`LS sequence numberLS checksumLS AgeLS AgeLength`

Agora chegamos às informações reais sobre o estado do link. Isso se torna um pouco mais complicado devido à presença de informações sobre o TOS (tipo de serviço). Ignorando isso por um momento, cada link no LSA é representado por um `Link ID`, algum e um . Os dois primeiros campos identificam o link; uma maneira comum de fazer isso seria usar o ID do roteador na extremidade oposta do link como o e, em seguida, usar o para diferenciar entre vários links paralelos, se necessário. O é, obviamente, o custo do link. nos diz algo sobre o link — por exemplo, se é um link ponto a ponto.`Link DatametricLink IDLink DatametricType`

As informações do TOS estão presentes para permitir que o OSPF escolha rotas diferentes para pacotes IP com base no valor do campo TOS. Em vez de atribuir uma única métrica a um link, é possível atribuir métricas diferentes dependendo do valor do TOS dos dados. Por exemplo, se tivéssemos um link em nossa rede com boa capacidade para tráfego sensível a atrasos, poderíamos atribuir a ele uma métrica baixa para o valor do TOS, que representa um atraso baixo, e uma métrica alta para todo o restante. O OSPF, então, escolheria um caminho mais curto diferente para os pacotes cujo campo TOS tivesse esse valor definido. Vale ressaltar que, no momento da redação deste texto, esse recurso não havia sido amplamente implementado.

3.4.4 Métricas

A discussão anterior pressupõe que os custos de enlace, ou métricas, sejam conhecidos quando executamos o algoritmo de roteamento. Nesta seção, examinamos algumas maneiras de calcular os custos de enlace que se mostraram eficazes na prática. Um exemplo que já vimos, bastante razoável e simples, é atribuir um custo de 1 a todos os enlaces — a rota de menor custo será então aquela com o menor número de saltos. Essa abordagem, no entanto, apresenta várias desvantagens. Primeiro, ela não distingue entre enlaces com base na latência. Assim, um enlace via satélite com latência de 250 ms parece tão atraente para o protocolo de roteamento quanto um enlace terrestre com latência de 1 ms. Segundo, ela não distingue entre rotas com base na capacidade, fazendo com que um enlace de 1 Mbps pareça tão bom quanto um enlace de 10 Gbps. Por fim, ela não distingue entre enlaces com base em sua carga atual, impossibilitando o roteamento em enlaces sobrecarregados. Acontece que este último problema é o mais difícil, pois você está tentando capturar as características complexas e dinâmicas de um enlace em um único custo escalar.

A ARPANET foi o campo de testes para diversas abordagens diferentes de cálculo de custo de enlace. (Foi também o local onde se demonstrou a estabilidade superior do roteamento por estado de enlace em relação ao roteamento por vetor de distância; o mecanismo original utilizava o vetor de distância, enquanto a versão posterior utilizava o estado de enlace.) A discussão a seguir traça a evolução da métrica de roteamento da ARPANET e, ao fazê-lo, explora os aspectos sutis do problema.

A métrica de roteamento original da ARPANET media o número de pacotes enfileirados aguardando transmissão em cada enlace, o que significava que um enlace com 10 pacotes enfileirados aguardando transmissão recebia um peso de custo maior do que um enlace com 5 pacotes enfileirados para transmissão. Usar o comprimento da fila como métrica de roteamento não funcionou bem, no entanto, visto que o comprimento da fila é uma medida artificial da carga — ele move os pacotes em direção à fila mais curta, em vez de em direção ao destino, uma situação bastante familiar para aqueles que pulam de fila em fila no supermercado. Em outras palavras, o mecanismo de

roteamento original da ARPANET sofria por não levar em consideração nem a largura de banda nem a latência do enlace.

Uma segunda versão do algoritmo de roteamento da ARPANET levou em consideração a largura de banda do link e a latência e utilizou o atraso, em vez de apenas o comprimento da fila, como medida de carga. Isso foi feito da seguinte forma. Primeiro, cada pacote de entrada recebeu um registro de data e hora com seu horário de chegada ao roteador (`ArrivalTime`); seu horário de saída do roteador (`DepartTime`) também foi registrado. Em segundo lugar, quando o ACK em nível de link era recebido do outro lado, o nó calculava o atraso para aquele pacote como

```
Delay = (DepartTime - ArrivalTime) + TransmissionTime + Latency
```

onde `TransmissionTime`e `Latency`foram definidos estaticamente para o link e capturaram a largura de banda e a latência do link, respectivamente. Observe que, neste caso, representa o tempo que o pacote ficou atrasado (enfileirado) no nó devido à carga. Se o ACK não chegou, mas o pacote atingiu o tempo limite, então foi redefinido para o momento em que o pacote foi *retransmitido* . Neste caso, captura a confiabilidade do link — quanto mais frequente a retransmissão de pacotes, menos confiável o link e mais queremos evitá-lo. Por fim, o peso atribuído a cada link foi derivado do atraso médio experimentado pelos pacotes enviados recentemente por esse link.`DepartTime - ArrivalTimeDepartTimeDepartTime - ArrivalTime`

Embora representasse uma melhoria em relação ao mecanismo original, essa abordagem também apresentava muitos problemas. Sob carga leve, funcionava razoavelmente bem, já que os dois fatores estáticos de atraso dominavam o custo. Sob carga pesada, no entanto, um link congestionado começava a anunciar um custo muito alto. Isso fazia com que todo o tráfego fosse transferido daquele link, deixando-o ocioso. Em seguida, ele anunciava um custo baixo, atraindo todo o tráfego de volta, e assim por diante. O efeito dessa instabilidade era que, sob carga pesada, muitos links passavam muito tempo ociosos, o que é a última coisa que se deseja sob carga pesada.

Outro problema era que a faixa de valores de link era muito grande. Por exemplo, um link de 9,6 kbps com alta carga poderia parecer 127 vezes mais caro do que um link de 56 kbps com carga leve. (Lembre-se de que estamos falando da ARPANET por volta de 1975.) Isso significa que o algoritmo de roteamento escolheria um caminho com 126 saltos de links de 56 kbps com carga leve em vez de um caminho de 9,6 kbps com 1 salto. Embora eliminar parte do tráfego de uma linha sobrecarregada seja uma boa ideia, torná-la tão pouco atraente a ponto de perder todo o tráfego é excessivo. Usar 126 saltos quando 1 salto basta é, em geral, um mau uso dos recursos da rede. Além disso, os links de satélite foram indevidamente penalizados, de modo que um link de satélite ocioso de 56 kbps parecia consideravelmente mais caro do que um link terrestre ocioso de 9,6 kbps, embora o primeiro oferecesse melhor desempenho para aplicações de alta largura de banda.

Uma terceira abordagem abordou esses problemas. As principais mudanças foram comprimir consideravelmente a faixa dinâmica da métrica, levar em conta o tipo de link e suavizar a variação da métrica ao longo do tempo.

A suavização foi alcançada por vários mecanismos. Primeiro, a medição do atraso foi transformada em uma utilização do link, e esse valor foi calculado com base na última utilização relatada para suprimir mudanças repentinas. Segundo, havia um limite rígido para o quanto a métrica poderia mudar de um ciclo de medição para o próximo. Ao suavizar as mudanças no custo, a probabilidade de todos os nós abandonarem uma rota ao mesmo tempo é bastante reduzida.

A compressão da faixa dinâmica foi obtida alimentando a utilização medida, o tipo de link e a velocidade do link em uma função mostrada graficamente na Figura 92 abaixo. Observe o seguinte:

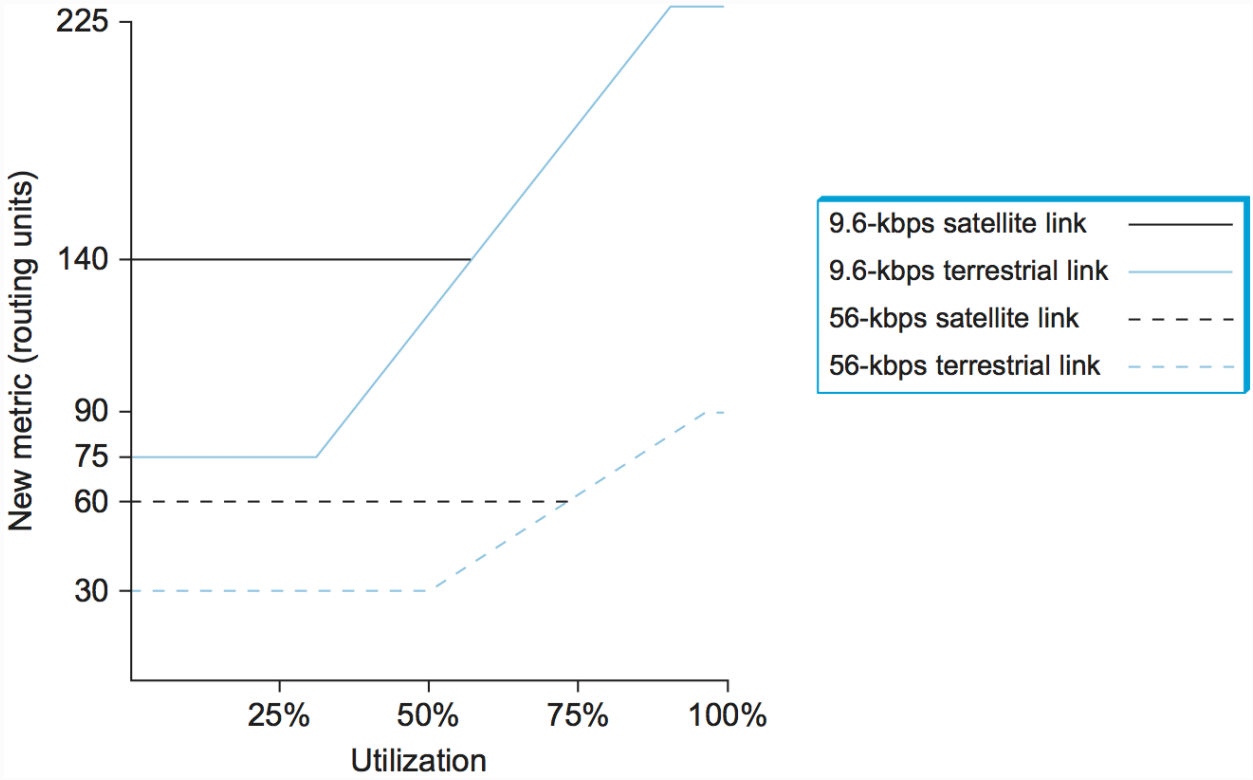


Figura 92. *Métrica de roteamento ARPANET revisada versus utilização de link.*

- Um link altamente carregado nunca mostra um custo maior que três vezes maior que seu custo quando ocioso.
- O link mais caro custa apenas sete vezes mais que o mais barato.
- Um link de satélite de alta velocidade é mais atraente do que um link terrestre de baixa velocidade.
- O custo é uma função da utilização do link somente em cargas moderadas a altas.

Todos esses fatores significam que uma ligação tem muito menos probabilidade de ser abandonada universalmente, visto que um aumento de três vezes no custo provavelmente tornará a ligação pouco atraente para alguns caminhos, enquanto a manterá como a melhor opção para outros. As inclinações, os deslocamentos e os pontos de interrupção das curvas na Figura 92 foram obtidos por meio de muita tentativa e erro e foram cuidadosamente ajustados para proporcionar um bom desempenho.

Apesar de todas essas melhorias, verifica-se que, na maioria das implantações de rede no mundo real, as métricas raramente mudam, ou nunca mudam, e apenas sob o controle de um administrador de rede, e não automaticamente, como descrito acima. A razão para isso se deve, em parte, ao fato de que a sabedoria popular agora defende que métricas que mudam dinamicamente são muito instáveis, embora isso provavelmente não precise ser verdade. Talvez mais significativamente, muitas redes hoje não apresentam a grande disparidade de velocidades e latências de link que prevalecia na ARPANET. Portanto, métricas estáticas são a norma. Uma abordagem comum para definir métricas é usar uma constante multiplicada por (1/largura_de_banda_do_link).

Conclusão principal

Por que ainda contamos a história de um algoritmo com décadas de existência que não é mais usado? Porque ele ilustra perfeitamente duas lições valiosas. A primeira é que os sistemas computacionais são frequentemente *projetados iterativamente com base na experiência.* Raramente acertamos na primeira vez, por isso é importante implementar uma solução simples o mais cedo possível e esperar aprimorá-la com o tempo. Ficar preso na fase de projeto indefinidamente geralmente não é um bom plano. A segunda é o conhecido princípio KISS: *Keep it Simple, Stupid (Mantenha a simplicidade, estúpido).* Ao construir um sistema complexo, menos costuma ser mais. Oportunidades para inventar otimizações sofisticadas são abundantes e é uma oportunidade tentadora a ser aproveitada. Embora tais otimizações às vezes tenham valor a curto prazo, é chocante a frequência com que uma abordagem simples se mostra a melhor ao longo do tempo. Isso ocorre porque, quando um sistema tem muitas partes móveis, como a Internet certamente tem, manter cada parte o mais simples possível geralmente é a melhor abordagem. [Próximo]

3.5 Implementation

So far, we have talked about what switches and routers must do without describing how they do it. There is a straightforward way to build a switch or router: Buy a general-purpose processor and equip it with multiple network interfaces. Such a device, running suitable software, can receive packets on one of its interfaces, perform any of the switching or forwarding functions described in this chapter, and send packets out another of its interfaces. This so called *software switch* is not too far removed from the architecture of many commercial mid- to low-end network devices.[1] Implementations that deliver high-end performance typically take advantage of additional hardware acceleration. We refer to these as *hardware switches*, although both approaches obviously include a combination of hardware and software.

[1]

This is also how the very first Internet routers, often called *gateways* at the time, were implemented in the early days of the Internet.

This section gives an overview of both software-centric and hardware-centric designs, but it is worth noting that on the question of switches versus routers, the distinction isn't such a big deal. It turns out that the implementation of switches and routers have so much in common that a network administrator typically buys a single forwarding box and then configures it to be an L2 switch, an L3 router, or some combination of the two. Since their internal designs are so similar, we'll use the word *switch* to cover both variants throughout this section, avoiding the tedium of saying "switch or router" all the time. We'll call out the differences between the two when appropriate.

3.5.1 Software Switch

Figure 93 shows a software switch built using a general-purpose processor with four network interface cards (NICs). The path for a typical packet that arrives on, say, NIC 1 and is forwarded out on NIC 2 is straightforward: as NIC 1 receives the packet it copies

its bytes directly into the main memory over the I/O bus (PCIe in this example) using a technique called *direct memory access* (DMA). Once the packet is in memory, the CPU examines its header to determine which interface the packet should be sent out on, and instructs NIC 2 to transmit the packet, again directly out of main memory using DMA. The important take-away is that the packet is buffered in main memory (this is the "store" half of store-and-forward), with the CPU reading only the necessary header fields into its internal registers for processing.

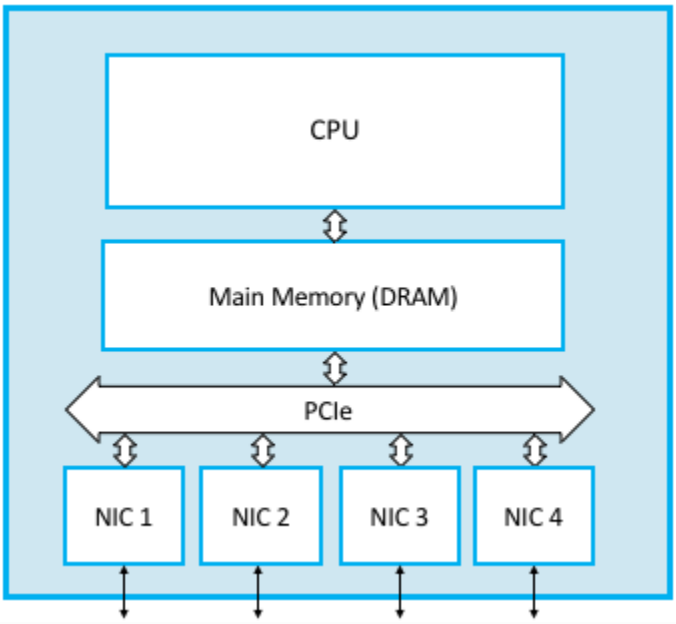


Figure 93. *A general-purpose processor used as a software switch.*

There are two potential bottlenecks with this approach, one or both of which limits the aggregate packet forwarding capacity of the software switch.

The first problem is that performance is limited by the fact that all packets must pass into and out of main memory. Your mileage will vary based on how much you are willing to pay for hardware, but as an example, a machine limited by a 1333-MHz, 64-bit-wide memory bus can transmit data at a peak rate of a little over 100 Gbps—enough to build a switch with a handful of 10-Gbps Ethernet ports, but hardly enough for a high-end router in the core of the Internet.

Moreover, this upper bound assumes that moving data is the only problem. This is a fair approximation for long packets but a bad one when packets are short, which is the worst-case situation switch designers have to plan for. With minimum-sized packets, the cost of processing each packet—parsing its header and deciding which output link to transmit it on—is likely to dominate, and potentially become a bottleneck. Suppose, for example, that a processor can perform all the necessary processing to switch 40 million packets each second. This is sometimes called the packet per second (pps) rate. If the average packet is 64 bytes, this would imply

$$\text{Throughput} = \text{pps} \times \text{BitsPerPacket}$$

$$= 40 \times 10^6 \times 64 \times 8$$

$$= 2048 \times 10^7$$

that is, a throughput of about 20 Gbps—fast, but substantially below the range users are demanding from their switches today. Bear in mind that this 20 Gbps would be shared by all users connected to the switch, just as the bandwidth of a single (unswitched) Ethernet segment is shared among all users connected to the shared medium. Thus, for example, a 16-port switch with this aggregate throughput would only be able to cope with an average data rate of about 1 Gbps on each port.[2]

[2]

These example performance numbers do not represent the absolute maximum throughput rate that highly tuned software running on a high-end server could achieve, but they are indicative of limits one ultimately faces in pursuing this approach.

One final consideration is important to understand when evaluating switch implementations. The non-trivial algorithms discussed in this chapter—the spanning tree algorithm used by learning bridges, the distance-vector algorithm used by RIP, and the link-state algorithm used by OSPF—are *not* directly part of the per-packet

forwarding decision. They run periodically in the background, but switches do not have to execute, say, OSPF code for every packet it forwards. The most costly routine the CPU is likely to execute on a per-packet basis is a table lookup, for example, looking up a VCI number in a VC table, an IP address in an L3 forwarding table, or an Ethernet address in an L2 forwarding table.

Key Takeaway

The distinction between these two kinds of processing is important enough to give it a name: the *control plane* corresponds to the background processing required to "control" the network (e.g., running OSPF, RIP, or the BGP protocol described in the next chapter) and the *data plane* corresponds to the per-packet processing required to move packets from input port to output port. For historical reasons, this distinction is called *control plane* and *user plane* in cellular access networks, but the idea is the same, and in fact, the 3GPP standard defines CUPS (Control/User Plane Separation) as an architectural principle.

These two kinds of processing are easy to conflate when both run on the same CPU, as is the case in software switch depicted in Figure 93, but performance can be dramatically improved by optimizing how the data plane is implemented, and correspondingly, specifying a well-defined interface between the control and data planes. [Next]

3.5.2 Hardware Switch

Throughout much of the Internet's history, high-performance switches and routers have been specialized devices, built with Application-Specific Integrated Circuits (ASICs). While it was possible to build low-end routers and switches using commodity servers running C programs, ASICs were required to achieve the required throughput rates.

The problem with ASICs is that hardware takes a long time to design and fabricate, meaning the delay for adding new features to a switch is usually measured in years, not the days or weeks today's software industry is accustomed to. Ideally, we'd like to benefit from the performance of ASICs and the agility of software.

Fortunately, recent advances in domain specific processors (and other commodity components) have made this possible. Just as importantly, the full architectural specification for switches that take advantage of these new processors is now available online—the hardware equivalent of *open source software*. This means anyone can build a high-performance switch by pulling the blueprint off the web (see the Open Compute Project, OCP, for examples) in the same way it is possible to build your own PC. In both cases you still need software to run on the hardware, but just as Linux is available to run on your home-built PC, there are now open source L2 and L3 stacks available on GitHub to run on your home-built switch. Alternatively, you can simply buy a pre-built switch from a commodity switch manufacturer and then load your own software onto it. The following describes these open *bare-metal switches*, so called to contrast them with closed devices, in which hardware and software are tightly bundled, that have historically dominated the industry.

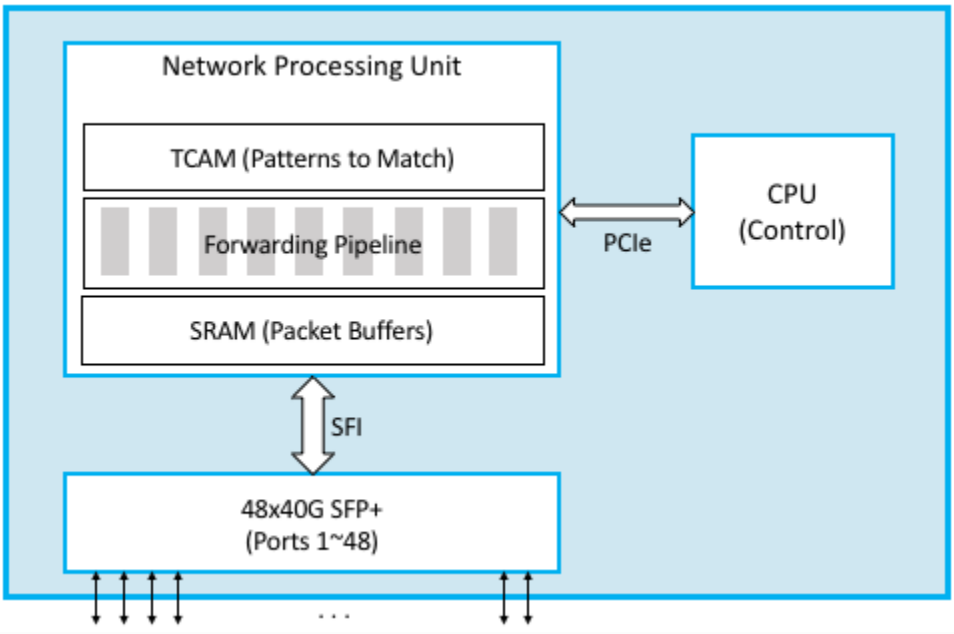


Figure 94. *Bare-metal switch using a Network Processing Unit.*

Figure 94 is a simplified depiction of a bare-metal switch. The key difference from the earlier implementation on a general-purpose processor is the addition of a Network Processor Unit (NPU), a domain-specific processor with an architecture and instruction set that has been optimized for processing packet headers (i.e., for implementing the data plane). NPUs are similar in spirit to GPUs that have an architecture optimized for rendering computer graphics, but in this case, the NPU is optimized for parsing packet headers and making a forwarding decision. NPUs are able to process packets (input, make a forwarding decision, and output) at rates measured in Terabits per second (Tbps), easily fast enough to keep up with 32x100-Gbps ports, or the 48x40-Gbps ports shown in the diagram.

Network Processing Units

> Our use of the term NPU is a bit non-standard. Historically, NPU was the name given more narrowly-defined network processing chips used, for example, to implement intelligent firewalls or deep packet inspection. They were not as general-purpose as the NPUs we're discussing here; nor were they as high-performance. It seems likely that the current approach will make purpose-built network processors obsolete, but in any case, we prefer the NPU nomenclature because it is consistent with the trend to build programmable domain-specific processors, including GPUs for graphics and TPUs (Tensor Processing Units) for AI.

The beauty of this new switch design is that a given bare-metal switch can now be programmed to be an L2 switch, an L3 router, or a combination of both, just by a matter of programming. The exact same control plane software stack used in a software switch still runs on the control CPU, but in addition, data plane "programs" are loaded onto the NPU to reflect the forwarding decisions made by the control plane software. Exactly how one "programs" the NPU depends on the chip vendor, of which there are currently several. In some cases, the forwarding pipeline is fixed and the control processor merely loads the forwarding table into the NPU (by fixed we mean the NPU only knows how to process certain headers, like Ethernet and IP), but in other cases, the forwarding

pipeline is itself programmable. P4 is a new programming language that can be used to program such NPU-based forwarding pipelines. Among other things, P4 tries to hide many of the differences in the underlying NPU instruction sets.

Internally, an NPU takes advantage of three technologies. First, a fast SRAM-based memory buffers packets while they are being processed. SRAM (Static Random Access Memory), is roughly an order of magnitude faster than the DRAM (Dynamic Random Access Memory) that is used by main memory. Second, a TCAM-based memory stores bit patterns to be matched in the packets being processed. The “CAM” in TCAM stands for “Content Addressable Memory,” which means that the key you want to look up in a table can effectively be used as the address into the memory that implements the table. The “T” stands for “Ternary” which is a fancy way to say the key you want to look up can have wildcards in it (e.g, key `10*1` matches both `1001` and `1011`). Finally, the processing involved to forward each packet is implemented by a forwarding pipeline. This pipeline is implemented by an ASIC, but when well-designed, the pipeline’s forwarding behavior can be modified by changing the program it runs. At a high level, this program is expressed as a collection of *(Match, Action)* pairs: if you match such-and-such field in the header, then execute this-or-that action.

The relevance of packet processing being implemented by a multi-stage pipeline rather than a single-stage processor is that forwarding a single packet likely involves looking at multiple header fields. Each stage can be programmed to look at a different combination of fields. A multi-stage pipeline adds a little end-to-end latency to each packet (measured in nanoseconds), but also means that multiple packets can be processed at the same time. For example, Stage 2 can be making a second lookup on packet A while Stage 1 is doing an initial lookup on packet B, and so on. This means the NPU as a whole is able to keep up with line speeds. As of this writing, the state of the art is 25.6 Tbps.

Finally, Figure 94 includes other commodity components that make this all practical. In particular, it is now possible to buy pluggable *transceiver* modules that take care of all the media access details—be it Gigabit Ethernet, 10-Gigabit Ethernet, or SONET—as

well as the optics. These transceivers all conform to standardized form factors, such as SFP+, that can in turn be connected to other components over a standardized bus (e.g., SFI). Again, the key takeaway is that the networking industry is just now entering into the same commoditized world that the computing industry has enjoyed for the last two decades.

3.5.3 Software Defined Networks

With switches becoming increasingly commoditized, attention is rightfully shifting to the software that controls them. This puts us squarely in the middle of a trend to build *Software Defined Networks* (SDN), an idea that started to germinate about ten years ago. In fact, it was the early stages of SDN that triggered the networking industry to move towards bare-metal switches.

The fundamental idea of SDN is one we've already discussed: to decouple the network control plane (i.e., where routing algorithms like RIP, OSPF, and BGP run) from the network data plane (i.e., where packet forwarding decisions get made), with the former moved into software running on commodity servers and the latter implemented by bare-metal switches. The key enabling idea behind SDN was to take this decoupling a step further, and to define a standard interface between the control plane and the data plane. Doing so allows any implementation of the control plane to talk to any implementation of the data plane; this breaks the dependency on any one vendor's bundled solution. The original interface is called *OpenFlow*, and this idea of decoupling the control and data planes came to be known as disaggregation. (The P4 language mentioned in the previous subsection is a second-generation attempt to define this interface by generalizing OpenFlow.)

Another important aspect of disaggregation is that a logically centralized control plane can be used to control a distributed network data plane. We say logically centralized because while the state collected by the control plane is maintained in a global data structure, such as a Network Map, the implementation of this data structure could still be distributed over multiple servers. For example, it could run in a cloud. This is

important for both scalability and availability, where the key is that the two planes are configured and scaled independent of each other. This idea took off quickly in the cloud, where today's cloud providers run SDN-based solutions both within their datacenters and across the backbone networks that interconnect their datacenters.

One consequence of this design that isn't immediately obvious is that a logically centralized control plane doesn't just manage a network of physical (hardware) switches that interconnects physical servers, but it also manages a network of virtual (software) switches that interconnect virtual servers (e.g., Virtual Machines and containers). If you're counting "switch ports" (a good measure of all the devices connected to your network) then the number of virtual ports in the Internet rocketed past the number of physical ports in 2012.

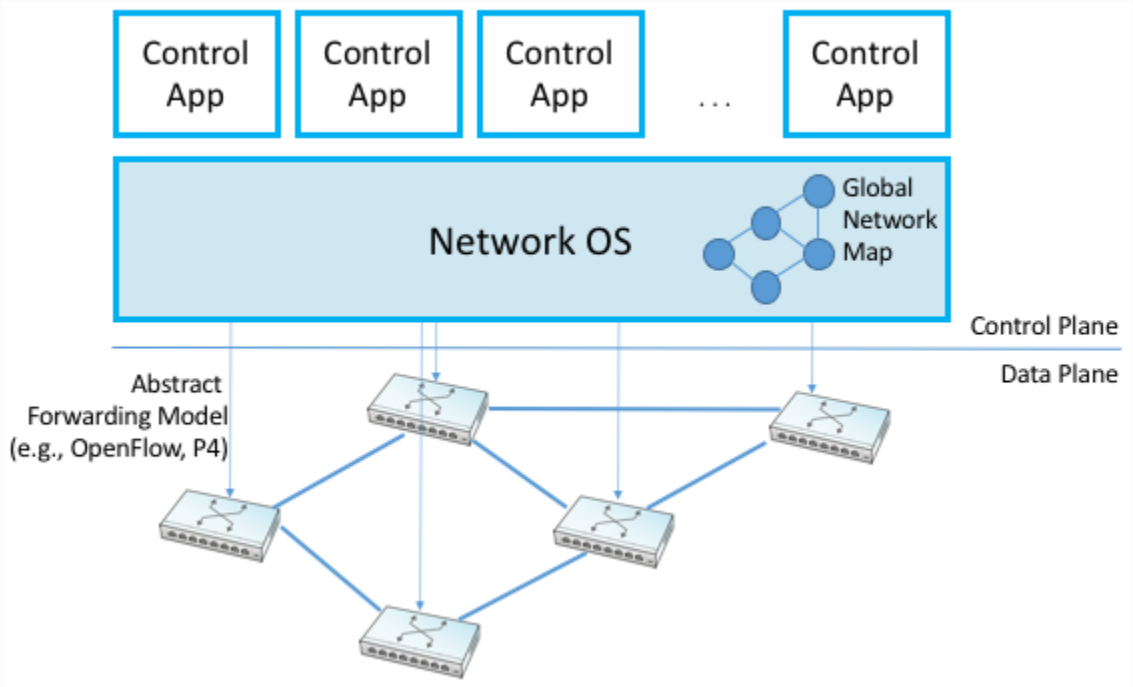


Figure 95. *Network Operating System (NOS) hosting a set of control applications and providing a logically centralized point of control for an underlying network data plane.*

One of other key enablers for SDN's success, as depicted in Figure 95, is the Network Operating System (NOS). Like a server operating system (e.g., Linux, iOS, Android, Windows) that provides a set of high-level abstractions that make it easier to implement applications (e.g., you can read and write files instead of directly accessing disk drives),

a NOS makes it easier to implement network control functionality, otherwise known as *Control Apps*. A good NOS abstracts the details of the network switches and provides a *Network Map* abstraction to the application developer. The NOS detects changes in the underlying network (e.g., switches, ports, and links going up-and-down) and the control application simply implements the behavior it wants on this abstract graph. This means the NOS takes on the burden of collecting network state (the hard part of distributed algorithms like Link-State and Distance-Vector algorithms) and the app is free to simply implement the shortest path algorithm and load the forwarding rules into the underlying switches. By centralizing this logic, the goal is to come up with a globally optimized solution. The published evidence from cloud providers that have embraced this approach confirms this advantage.

Key Takeaway

It is important to understand that SDN is an implementation strategy. It does not magically make fundamental problems like needing to compute a forwarding table go away. But instead of burdening the switches with having to exchange messages with each other as part of a distributed routing algorithm, the logically centralized SDN controller is charged with collecting link and port status information from the individual switches, constructing a global view of the network graph, and making that graph available to the control apps. From the control application's perspective, all the information it needs to compute the forwarding table is locally available. Keeping in mind that the SDN Controller is logically centralized but physically replicated on multiple servers—for both scalable performance and high availability—it is still a hotly contested question whether the centralized or distributed approach is best. [Next]

As much of an advantage as the cloud providers have been able to get out of SDN, its adoption in enterprises and Telcos has been much slower. This is partly about the ability of different markets to manage their networks. The Googles, Microsofts, and Amazons of the world have the engineers and DevOps skills needed to take advantage of this

technology, whereas others still prefer pre-packaged and integrated solutions that support the management and command line interfaces they are familiar with.

Perspectiva: Redes virtuais até o fim

Desde que existem redes comutadas por pacotes, surgiram ideias sobre como virtualizá-las, começando com circuitos virtuais. Mas o que exatamente significa virtualizar uma rede?

A memória virtual é um exemplo útil. A memória virtual cria uma abstração de um conjunto grande e privado de memória, mesmo que a memória física subjacente possa ser compartilhada por muitos aplicativos e seja consideravelmente menor que o conjunto aparente de memória virtual. Essa abstração permite que os programadores operem sob a ilusão de que há bastante memória e que ninguém mais a está usando, enquanto, internamente, o sistema de gerenciamento de memória cuida de coisas como mapear a memória virtual para recursos físicos e evitar conflitos entre usuários.

Da mesma forma, a virtualização de servidores apresenta a abstração de uma máquina virtual (VM), que possui todos os recursos de uma máquina física. Novamente, pode haver muitas VMs suportadas em um único servidor físico, e o sistema operacional e os usuários na máquina virtual felizmente não sabem que a VM está sendo mapeada para recursos físicos.

Um ponto fundamental é que a virtualização de recursos computacionais preserva as abstrações e interfaces que existiam antes de serem virtualizadas. Isso é importante porque significa que os usuários dessas abstrações não precisam mudar — eles veem uma reprodução fiel do recurso que está sendo virtualizado. A virtualização também significa que os diferentes usuários (às vezes chamados de *locatários*) não podem interferir uns nos outros. Então, o que acontece quando tentamos virtualizar uma rede?

VPNs, conforme descrito na Seção 3.3 , foram um dos primeiros sucessos das redes virtuais. Elas permitiram que as operadoras apresentassem aos clientes corporativos a ilusão de que tinham sua própria rede privada, mesmo que, na realidade, estivessem compartilhando links e switches subjacentes com muitos outros usuários. As VPNs, no entanto, virtualizam apenas alguns recursos, principalmente tabelas de endereçamento e roteamento. A virtualização de rede, como comumente entendida hoje, vai além, virtualizando todos os aspectos da rede. Isso significa que uma rede virtual deve suportar todas as abstrações básicas de uma rede física. Nesse sentido, elas são análogas à máquina virtual, com seu suporte a todos os recursos de um servidor: CPU, armazenamento, E/S e assim por diante.

Para esse fim, VLANs, conforme descrito na Seção 3.2 , são como normalmente virtualizamos uma rede L2. VLANs provaram ser bastante úteis para empresas que queriam isolar diferentes grupos internos (por exemplo, departamentos, laboratórios), dando a cada um deles a aparência de ter sua própria LAN privada. VLANs também foram vistas como uma maneira promissora de virtualizar redes L2 em datacenters em nuvem, tornando possível dar a cada locatário sua própria rede L2 para isolar seu tráfego do tráfego de todos os outros locatários. Mas havia um problema: as 4096 VLANs possíveis não eram suficientes para contabilizar todos os locatários que uma nuvem poderia hospedar e, para complicar as coisas, em uma nuvem a rede precisa conectar *máquinas virtuais* em vez das máquinas físicas nas quais essas VMs são executadas.

Para resolver esse problema, outro padrão chamado *Virtual Extensible LAN* (VXLAN) foi introduzido. Diferentemente da abordagem original, que encapsulava efetivamente um quadro Ethernet virtualizado dentro de outro quadro Ethernet, a VXLAN encapsula um quadro Ethernet virtual dentro de um pacote UDP. Isso significa que uma rede virtual baseada em VXLAN (que é frequentemente chamada de *rede sobreposta*) é executada sobre uma rede baseada em IP, que por sua vez é executada em uma

Ethernet subjacente (ou talvez em apenas uma VLAN da Ethernet subjacente). A VXLAN também possibilita que um locatário de nuvem tenha várias VLANs próprias, o que permite que ele segregue seu próprio tráfego interno. Isso significa que, em última análise, é possível ter uma VLAN encapsulada em uma sobreposição de VXLAN encapsulada em uma VLAN.

O ponto forte da virtualização é que, quando feita corretamente, deve ser possível aninhar um recurso virtualizado dentro de outro recurso virtualizado, já que, afinal, um recurso virtual deve se comportar exatamente como um recurso físico, e sabemos como virtualizar recursos físicos! Em outras palavras, ser capaz de virtualizar um recurso virtual é a melhor prova de que você fez um bom trabalho virtualizando o recurso físico original. Para retomar a mitologia da Tartaruga Mundial: são redes virtuais em todos os níveis.

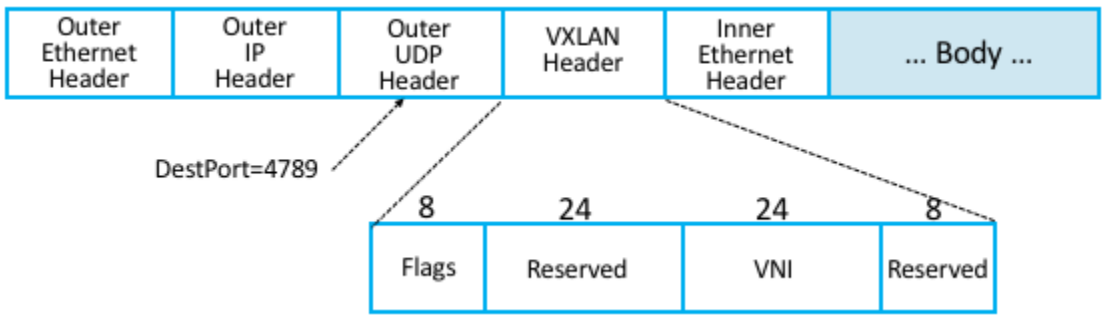


Figura 96. *Cabeçalho VXLAN encapsulado em um pacote UDP/IP.*

O cabeçalho VXLAN real é simples, como mostrado na Figura 96. *Ele inclui um ID de Rede Virtual* (VNI) de 24 bits , além de alguns bits de sinalização e reservados. Ele também implica uma configuração específica dos campos de porta de origem e destino UDP (consulte a Seção 5.1), com a porta de destino 4789 oficialmente reservada para VXLANs. Descobrir como identificar exclusivamente LANs virtuais (etiquetas VLAN) e redes virtuais (VIDs VXLAN) é a parte fácil. Isso ocorre porque o encapsulamento é a base fundamental da virtualização; tudo o que você precisa adicionar é um identificador que informe a qual dos muitos usuários possíveis esse pacote encapsulado pertence.

A parte difícil é lidar com a ideia de redes virtuais aninhadas (encapsuladas) dentro de redes virtuais, que é a versão de recursão das redes. O outro desafio é entender como automatizar a criação, o gerenciamento, a migração e a exclusão de redes virtuais, e nesse aspecto ainda há muito espaço para melhorias. Dominar esse desafio estará no cerne das redes na próxima década e, embora parte desse trabalho, sem dúvida, aconteça em ambientes proprietários, existem plataformas de virtualização de redes de código aberto (por exemplo, o projeto *Tungsten Fabric* da Linux Foundation) liderando o caminho.

Perspectiva mais ampla

Para continuar lendo sobre a cloudificação da Internet, consulte Perspectiva: a nuvem está devorando a Internet .

Para saber mais sobre a maturação das redes virtuais, recomendamos:

- Virtualização de rede revisitada , 2023.
- Tecido de tungstênio , 2018.