Série VoIP (2) – TCP/IP for not-so dummies

J. R. Smolka

Conforme prometido... Este é o segundo artigo da série VoIP, onde vamos revisar as características gerais de funcionamento de redes TCP/IP. O nível da descrição será superficial, *pero no mucho*, e está confinada às características do TCP/IP que possuem *headiness index*[1] +5. As características do TCP/IP que foram incorporadas para atender a idéias que possuem headiness index menor que +5 vão ser objeto do próximo artigo desta série.

Não conheço nenhum outro exemplo de tecnologia que tenha conseguido sobreviver, sem mudanças essenciais no seu projeto básico, a mais de vinte anos de crescimento exponencial no número de usuários. Pessoalmente, fico profundamente impressionado com a robustez e flexibilidade demonstradas pela arquitetura TCP/IP ao longo destes anos. E acredito que esta arquitetura realmente é capaz de atender às demandas dos mais variados tipos de aplicação sem que seja necessário alterar sua filosofia básica de funcionamento.

Para quem queira uma referência definitiva sobre os conceitos que vamos falar neste artigo, recomendo o livro *Internetworking with TCP/IP – volume 1*, de Douglas Comer (apesar de antigo, ainda é imperdível). Para aprofundamento conceitual, baixe o *redbook* IBM *TCP/IP Tutorial and Technical Overview*[2]. Para os programadores de aplicação (especialmente no paradigma *client-server*), recomendo os volumes 2 e 3 do livro do Douglas Comer já mencionado. Quer mais? RTFR[3] no site do IETF[4]!

TCP/IP e OSI-RM:

Quando queremos analisar o funcionamento de uma determinada arquitetura de rede baseada em comutação de pacotes, o melhor padrão de comparação ainda é o documento ISO[5] *Open Systems Interconnection – Reference Model*, conhecido simplesmente como OSI-RM.

Cabe uma pequena discussão sobre as traduções deste título para o português. Geralmente encontra-se a tradução "interconexão de sistemas abertos". IMNSHO[6], esta tradução induz a um raciocínio errado: que os sistemas (de computação) devem ser abertos para poderem se comunicar. Creio que a tradução mais adequada para o espírito do documento seja: "interconexão aberta de sistemas", para evidenciar que o objetivo é,

---

[1] Ver a definição do *headiness index* no final do primeiro artigo desta série.

[2] www.redbooks.ibm.com/redbooks/pdfs/gg243376.pdf

[3] Read the f**king RFC

[4] www.ietf.org

[5] O significado da sigla que identifica a organização internacional de padronização, ISO vem do grego *isos* – igual, e, por extensão, padrão. Não significa *international standards organization*, como muitos pensam.

[6] In my not-so humble opinion.

apesar da heterogeneidade dos sistemas, podemos estabelecer um modo aberto de interconexão entre eles.

Outro fato que costuma ser esquecido é que uma arquitetura de rede é pensada para garantir a comunicação entre aplicações, e não entre pessoas. As pessoas são usuárias das aplicações, e não interagem diretamente com as funcionalidades da arquitetura de comunicação.

No OSI-RM, as funcionalidades críticas para garantir a interoperabilidade de aplicações em ambientes heterogêneos foram agrupadas em sete "camadas" (*layers*). O número sete não foi um consenso do comitê de trabalho, mas uma solução de compromisso entre propostas divergentes.

A arquitetura TCP/IP também adota a filosofia de dividir as funcionalidades em camadas, mas: o número de camadas não é igual ao OSI-RM; embora existam camadas com o mesmo nome nos dois modelos, as funcionalidades não são rigorosamente equivalentes.

Ainda assim, podemos fazer um paralelo funcional entre as camadas do OSI-RM e as camadas da arquitetura TCP/IP, como mostra a figura 1. À direita, são mencionados os principais protocolos utilizados em cada camada.

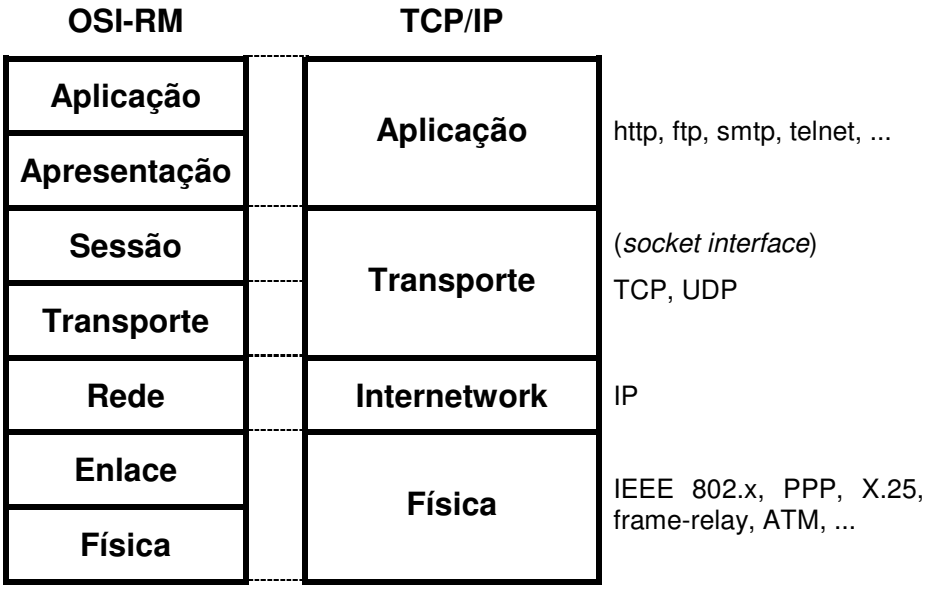

Figura 1 – OSI-RM versus TCP/IP

Seguindo o velho e honrado método "Jack, o estripador", vamos por partes... Uma camada de cada vez. Ao contrário do usual, vamos descer a pilha, e não subir. Por quê? Simples. A camada física é apenas um meio de transporte de pacotes entre sistemas adjacentes. Embora cada tipo de camada física adotada imponha determinadas restrições ao tráfego dos pacotes, o modelo conceitual da comunicação é independente disso.

Além disso, ficar fascinado demais pela camada física é uma postura com *headiness index* –5.

Camada de aplicação:

A camada de aplicação, na arquitetura TCP/IP, desempenha as funções equivalentes às designadas para as camadas de aplicação e apresentação do OSI-RM.

Cada protocolo deste nível define as primitivas necessárias para a interoperação de uma classe de aplicações (ex.: *file transfer*, *e-mail*, *remote file system*, etc.), e a implementação destas primitivas é de responsabilidade de cada programa de aplicação.

Para encaminhamento dos pacotes, é necessário que o endereço IP (da camada *internetwork*) seja determinado desde aqui. Uma possibilidade para resolver este problema é colocar o endereço IP *hard-coded* na própria aplicação. Esta solução é muito comum em redes privadas de pequeno porte, mas é totalmente inviável para uma rede privada de grande porte, ou para a Internet.

Nestes casos, as aplicações devem incluir a parte *client*, e o sistema deve ser configurado para conhecer o endereço IP do *server* (normalmente dois, primário e secundário) do serviço DNS (*domain naming system*), padronizado pelo IETF para a resolução de nomes de domínio (*domain names*) em endereços IP. Vale lembrar a distinção fundamental entre nomes e endereços: um nome é algo que diz quem você é, um endereço é algo que diz onde você está.

Camada de transporte:

Os protocolos de transporte da arquitetura isolam sessões através de portas (*ports*). Uma sessão *full-duplex* entre duas aplicações é identificada por um par de portas – origem (*source port*) e destino (*destination port*) – em cada sistema. Os números de porta de origem e destino, que fazem parte do cabeçalho (*header*) dos protocolos de transporte, são números binários com 16 bits, portanto podem assumir valores (expressos em decimal para consumo humano) de 0 a 65535.

Aplicações que funcionam no paradigma *client-server* abrem sessões através de um procedimento denominado *three-way handshake*:

a) O *server* abre uma porta para recebimento das requisições iniciais dos *clients*, e fica aguardando. Esta porta é denominada o *well-known port* do *server*. Nos protocolos padronizados pelo IETF, os *well-known ports* são administrados pelo ICANN[7] (que assumiu as funções da IANA[8]), e são designados na metade baixa da faixa de numeração (0 a 32767). Para protocolos não padronizados, recomenda-se usar apenas *well-known ports* na metade alta da faixa de numeração (32768 a 65535). Chamaremos estas faixas de numeração de porta, daqui para a frente, de *low ports* e *high ports*;

[7] www.icann.org

[8] www.iana.org

b) O primeiro pacote vindo de um *client* usa como *destination port* o *well-known port* do *server*, e como *source port* um *high port* arbitrário. Normalmente este pacote inicial contém, como carga útil (*payload*), uma primitiva HELLO do protocolo de aplicação;

c) Supondo que o *server* tenha condições de atender ao *client*, ele responde com um pacote que usa como *destination port* o mesmo *high port* designado pelo client no source port do seu pacote inicial, e como *destination port* um outro *high port* arbitrário. Tipicamente este pacote contém uma primitiva ACK (*acknowledgement*) do protocolo de aplicação como *payload*;

d) A partir daí, o *client* e o *server* prosseguem com a sessão. O *client* usando como *source port* o mesmo *high port* que ele selecionou em (b), e como *destination port* o *high port* selecionado pelo *server* em (c). E o *server* usando como *source port* o mesmo *high port* que ele selecionou em (c), e como *destination port* o *high port* selecionado pelo *client* em (b).

A figura 2 ilustra este processo.

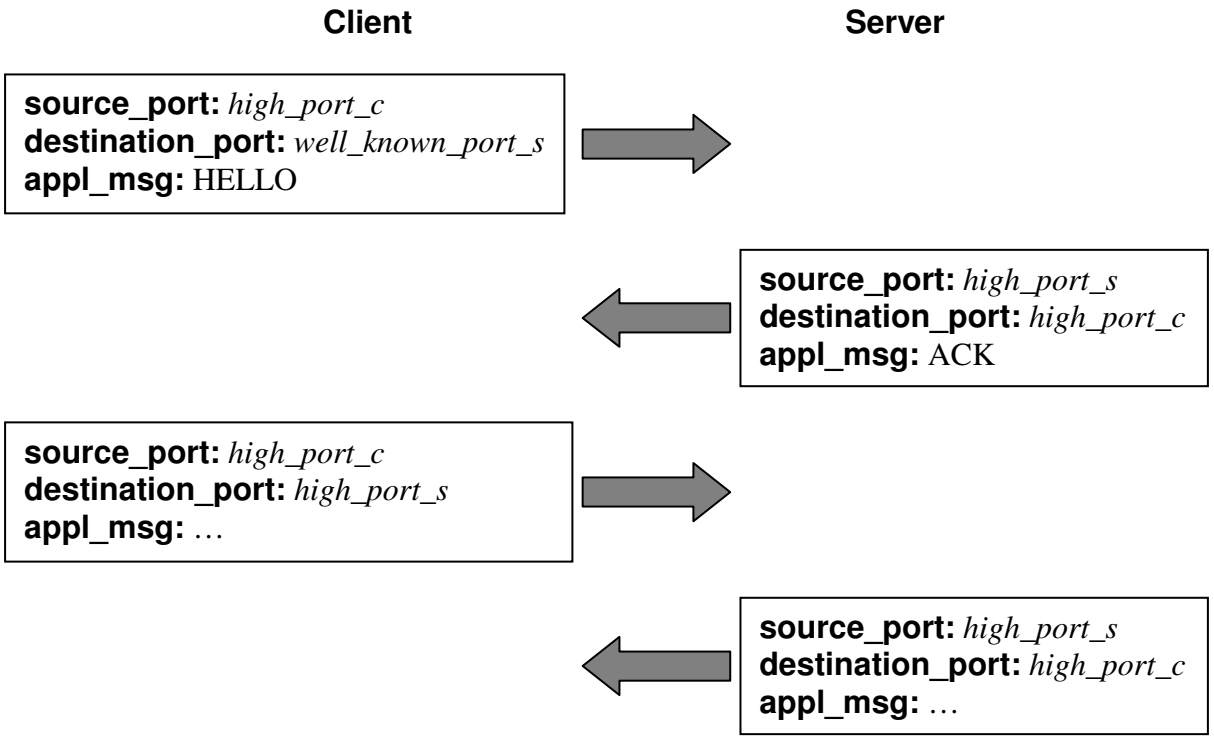

Figura 2 – *Three-way handshake*

Para facilitar a vida dos programadores de aplicação, existe uma API chamada *socket interface*, que permite ao programador implementar um serviços de sessão adequado ao seu caso. Já existem versões desta API portadas para todos os ambientes que eu já ouvi falar.

Falta ver como, exatamente, ocorre o transporte dos pacotes de aplicação. O IETF padronizou dois protocolos para esta função: *transmission control protocol* – TCP, e *user datagram protocol* – UDP.

O UDP oferece serviço de transporte na modalidade *best-effort delivery*, também conhecido como serviço de datagrama. Não existe nenhuma garantia sobre o que vai acontecer com os pacotes enquanto em trânsito, e a responsabilidade pela detecção e eventual recuperação de erros na transmissão cabe ao protocolo de aplicação.

O TCP oferece serviço de transporte confiável, com *flow control*. Ele é capaz de detectar e recuperar os seguintes tipos de erros na transmissão dos pacotes: perda, duplicação e troca de ordem.

O *flow control* implementado pelo TCP usa alguns bits no header para: numeração seqüencial dos pacotes transmitidos, confirmação de recebimento de pacotes e sinalização de sobrecarga. O algoritmo utilizado chama-se janela deslizante (*sliding window*), e funciona assim:

a) No início da sessão, as partes estabelecem o número máximo de pacotes que podem ser transmitidos por uma parte sem que haja confirmação explícita de recebimento pela outra parte. Este número é a largura da janela;

b) Cada parte estabelece um limite de transmissão igual ao número seqüencial do último pacote que teve recepção confirmada pela outra parte mais a largura da janela, e transmite até que este limite seja atingido;

c) Se o limite de transmissão é atingido, a transmissão cessa até o recebimento de uma confirmação, dentro de um limite de tempo (*timeout*);

d) Se o limite de tempo se esgotar sem recebimento de confirmação, aquela janela é considerada perdida, e retransmitida. Se a mesma janela não for confirmada por um certo número de vezes consecutivas (tipicamente três), a sessão é encerrada;

e) Quando uma confirmação é recebida, o início da janela é "deslizado" para o número do último pacote confirmado, e o processo de transmissão continua, como em (b).

Se uma das partes começa a deslizar constantemente a janela, sem que tenha sido atingido o número de pacotes limite da janela, ela assume que a outra parte pode suportar uma carga maior de transmissão, e aumenta a sua largura da janela.

Se uma das partes fica sobrecarregada na recepção, ela sinaliza no sentido reverso, ligando o bit de *source quench* do header TCP. A outra parte, ao receber pacotes com o bit de *source quench* ligado, diminui a sua largura de janela de transmissão, até passar a receber pacotes com o bit de *source quench* novamente desligado.

Muito bem... já temos um serviço de sessão e transporte que funciona. Resta saber agora como encaminhar os pacotes entre sistemas diferentes, mas isto já é assunto para a próxima seção.

Camada internetwork:

Estamos, finalmente, no reino do protocolo IP (*internetworking protocol*). Mas, antes de mais nada, alguns esclarecimentos.

Os pacotes IP são enviados entre interfaces físicas, e não entre sistemas. Desta forma, cada interface física de um sistema possui (pelo menos) um endereço IP (IP *address*) que a identifica univocamente em toda a rede.

O endereço IP sempre é tratado como sendo composto de duas partes lógicas: endereço da rede (*network address*) e endereço do host[9] (*host address*). Como as decisões de encaminhamento físico dos pacotes feitas pelo IP dependem diretamente da forma como estas duas partes do endereço são determinadas, vamos olhar isto mais de perto.

Endereços IP são números binários com 32 bits de tamanho. Normalmente (para consumo humano) ele é expresso no formato *dotted decimal*, onde cada um dos quatro bytes do endereço é representado pelo seu valor equivalente em decimal, separados por pontos. A figura 3 mostra alguns exemplos.

Dotted decimal		**Binário**
200.185.44.100	⟺	11001000 10111001 00101100 01100100
40.0.0.12	⟺	00101000 00000000 00000000 00001100
129.88.9.180	⟺	10000001 01011000 00001001 10000010

Figura 3 – Endereços IP (dotted decimal e binário)

Originalmente, a definição do *network address* e do *host address* ficavam rigidamente determinados em classes:

- *Classe A* – identificada por possuir o bit mais significativo do primeiro byte igual a zero. Neste caso, os bits do primeiro byte formavam o *network address*, e os bits do segundo, terceiro e quarto bytes formavam o *host address*. Poderiam existir 126 redes[10] classe A (1.0.0.0 a 126.0.0.0), cada uma podendo endereçar 16.777.214 *hosts*;

[9] Qualquer sistema conectado a uma rede TCP/IP, não importa a sua natureza, é chamado, genericamente, um *host*. Como iremos ver, entretanto, existem *hosts* e *hosts*.

[10] No *network address* (com exceção dos bits que definem a classe) e no *host address* vale a seguinte regra: todos os bits zero significam *this network/host*, e todos os bits um significam *all networks/hosts*. Por isso o número de *networks/hosts* possíveis é igual a 2^n–2, onde *n* é o número de bits de *network/host address*. Na classe A, a rede 127.0.0.0 (que cairia na categoria *all networks*) é reservada para *loopback addresses*.

- *Classe B* – identificada por possuir os dois bits mais significativos do primeiro byte iguais a 10. Neste caso, os bits do primeiro e do segundo bytes formavam o *network address*, e os bits do terceiro e quarto bytes formavam o *host address*. Poderiam existir 16.382 redes classe B (128.1.0.0 a 191.254.0.0), cada uma podendo endereçar 65.534 *hosts*;

- *Classe C* – identificada por possuir os três bits mais significativos do primeiro byte iguais a 110. Neste caso, os bits do primeiro, do segundo e do terceiro bytes formavam o *network address*, e os bits do quarto byte formavam o *host address*. Poderiam existir 2.097.150 redes classe C (192.0.1.0 a 223.255.254.0), cada uma podendo endereçar 254 *hosts*;

Com raciocínios análogos foram definidas também as classes D (para transmissões *multicast*) e E (para uso experimental).

Logo foi percebido que cada organização que ganhasse o direito de uso de uma dessas redes (de qualquer classe, mas especialmente para os casos das classes A e B) quase sempre iria precisar separar a rede designada para ela em sub-redes (*subnets*) menores, por razões administrativas.

Para isso foi criado o mecanismo de *subnetting*, que, quando configurado em um *host*, faz com que ele interprete a separação de um endereço IP em *network address* e *host address* de forma diferente das classes padronizadas.

Em interface física (o endereço IP é da interface, lembra?) dos *hosts* que fazem *subnetting* é configurada uma *subnet mask*, no formato *dotted decimal* (para consumo humano), com tantos bits um (começando do bit mais significativo do primeiro byte, e sempre contíguos) quantos sejam os bits que se queira interpretar como *network address*.

Nestas interfaces, sempre que tiver que ser avaliado o *network address* de um endereço IP, ele é obtido através de uma operação lógica AND entre o endereço IP e a *subnet mask*. Por exemplo: um *host* tem uma das suas interfaces configurada com endereço IP 10.30.20.55 e *subnet mask* 255.255.255.0. Pela lógica das classes, o *network address* seria interpretado como sendo 10.0.0.0 (classe A). Mas, com *subnetting*, o *network address* passa a ser interpretado como 10.30.20.0 (como se fosse uma classe C artificial).

O crescimento da Internet na sua fase acadêmica já impôs um considerável stress neste método de administração do espaço de endereçamento IP. Com a explosão de crescimento da fase comercial, ficou totalmente inviável continuar alocando endereços IP desta forma, e o fantasma do esgotamento dos endereços IP apareceu pela primeira vez.

O IETF, então, adotou uma reengenharia na forma de identificar *network addresses* e *host addresses*, chamada CIDR (*classless inter-domain routing*). Na prática, o CIDR torna o mecanismo de *subnetting* obrigatório.

A notação CIDR é a seguinte: após o endereço IP no formato *dotted decimal* coloca-se uma barra e um número variando entre zero e 32, inclusive. O número após a barra

identifica quantos bits do endereço IP (sempre começando do bit mais significativo do primeiro byte, contíguos – como na *subnet mask*) compõem o prefixo CIDR do endereço. O *network address*, é o resultado de uma operação AND entre o endereço IP e uma *subnet mask* equivalente ao prefixo CIDR.

Usando o mesmo exemplo anterior, se o endereço da interface for especificado, na notação CIDR, como 10.30.20.55/24, o *network address* será interpretado como 10.30.20.0, causando o mesmo efeito que o uso de uma *subnet mask* 255.255.255.0.

O uso de CIDR, mais a RFC 1918, que reservou algumas faixas de endereços para uso exclusivo em redes internas das organizações e estabeleceu o uso de NAT (*network address translation*) *gateways* para fazer a tradução de endereços IP dos pacotes que cruzam a fronteira entre redes privadas que usam TCP/IP e a Internet pública, permitiu (com bastante sucesso) que o problema do esgotamento dos endereços IP fosse adiado por algum tempo. Mas este esgotamento é apenas uma questão de tempo. Para contornar (quase) definitivamente este risco, foi feita a especificação da versão 6 do protocolo IP (a atual é a versão 4), conhecido como IPng (IP *next generation*) ou IPv6, que utiliza endereços com 128 bits de tamanho. Como o assunto da oportunidade e métodos para migração de IPv4 para IPv6 dá uma outra série de artigos, e como isto não é crítico para o que queremos falar sobre VoIP, não vou entrar mais fundo neste tema.

Agora que sabemos identificar qual a *subnet* e qual o *host* designados por um endereço IP, podemos entender como funciona o mecanismo de encaminhamento físico dos pacotes IP na rede, conhecido como roteamento IP.

Cada pacote que é recebido pela camada internetwork, vindo da camada de transporte do mesmo *host*, já vem com a informação de qual o endereço IP de destino final (*destination address*) que foi definido já na camada de aplicação, lembra? A *socket interface* mantém esta informação durante toda a duração da sessão. Este endereço, mais o endereço IP da interface do *host* por onde o pacote será enviado (*source address*) fazem parte do *header* do pacote IP.

A primeira coisa que o algoritmo IP neste host faz é perguntar: eu tenho adjacência física com o *host* identificado pelo *destination address*? Isto é respondido obtendo o *network address* associado ao *destination address* em todas as interfaces deste *host* (via AND com as respectivas *subnet masks* ou prefixos CIDR, o que dá na mesma). Se houver coincidência, então o *host* de destino é adjacente a este *host* através da interface onde a coincidência ocorreu.

Aqui está uma grande oportunidade para um administrador distraído "melecar" todo o processo de entrega de pacotes IP. Basta não coincidir as configurações das *subnets* com as redes físicas associadas às interfaces. Supondo que isto está ok, então o próximo passo é entregar o pacote IP à camada física associada à interface selecionada, e o pacote segue para o seu destino final diretamente.

Mas, se não ocorrer nenhuma coincidência do *network address* de destino com os *network addresses* das interfaces deste *host*? Então a entrega do pacote terá de ser feita de forma indireta, usando outro *host* como "ponte" para fazer o pacote chegar mais perto do seu destino final.

A decisão que este *host* tem de tomar, então, é: qual dos *hosts* (com os quais eu tenha adjacência física) é a melhor opção para encaminhar o pacote adiante, e fazê-lo chegar mais perto do destino pelo melhor caminho possível? Esta decisão é tomada com base em uma tabela de rotas (*routing table*), que todo *host* TCP/IP tem.

Para os *hosts* menos complicados, que tem apenas uma interface física de aceso à rede, esta tabela assume, geralmente, uma forma muito simples: apenas a declaração do endereço IP do *host* que deve ser usado para encaminhar pacotes IP para qualquer outra *subnet*, diferente desta onde este *host* está diretamente conectado. Este é o *last resort gateway*, também conhecido como *default gateway*.

Mas existem *hosts* com múltiplas interfaces, e/ou com múltiplas opções de quais poderiam ser os *hosts* usados para passar adiante os pacotes IP cujos *destination addresses* estão em *subnets* onde ele não tem conectividade física. Nestes casos a tabela de rotas terá múltiplas entradas, uma para cada *subnet* de destino (identificada pelo seu *network address* mais a *subnet mask* ou prefixo CIDR associados), e com a designação do endereço IP de um *host* (com conectividade física direta) que será usado para rotear os pacotes destinados àquela *subnet*.

Resumindo, se eu tenho conectividade física, entrego o pacote diretamente. E se eu não tenho conectividade física, entrego o pacote indiretamente, remetendo o pacote para o *host* identificado como *gateway* de acesso para a *subnet* de destino na minha tabela de rotas.

E se eu for um destes *gateways* intermediários? O que eu faço? Simples: recebo os pacotes físicos que foram endereçados a mim pelos outros *hosts*. Examino, então, o *destination address* para ver se ele coincide com algum dos meus próprios endereços IP (se eu tiver várias interfaces, então também terei vários endereços IP pelos quais eu posso ser conhecido). Se coincidir, então o pacote é para mim, e encaminho isto para cima, para a minha camada de transporte. Se não coincidir, eu também faço o processo de examinar minha tabela de rotas e passo o pacote adiante para o próximo *gateway* na rota.

Fatos importantes sobre este método de roteamento:

- Nenhum *host* conhece a rota inteira para uma determinada *subnet*. As tabelas de rotas de cada um apontam apenas quem é o próximo *gateway* no caminho.

- Em tese, qualquer *host* que tenha múltiplas interfaces pode ser usado como *gateway* entre *subnets* diferentes. O que não quer dizer que todos apresentem a mesma *performance* quando executam este papel. Na prática, usam-se *hosts* especialmente projetados para garantir o melhor desempenho possível da tarefa de encaminhar pacotes entre *subnets* – os roteadores (*routers*).

- O serviço prestado pela coletividade das camadas internetwork nos vários *hosts* conectados à rede é *best-effort delivery*, com um mínimo de sinalização de problemas através de mensagens ICMP[11] (Internet control message protocol), cuja implementação é mandatória junto com o IP.

[11] As mais comuns são: *port unreachable* (não tem ninguém esperando por este pacote na camada de transporte do *host* que emitiu a mensagem); *host unreachable* (emitida pelo último *gateway* na rota,

- Os pacotes IP são enviados de um *host* a outro "pulando carniça" (*hop-by-hop*) através de tantos *gateways* intermediários quantos sejam necessários, como mostra a figura 4. Portanto é crítico para a eficiência do processo que todas as tabelas de rotas sejam coerentes entre si.

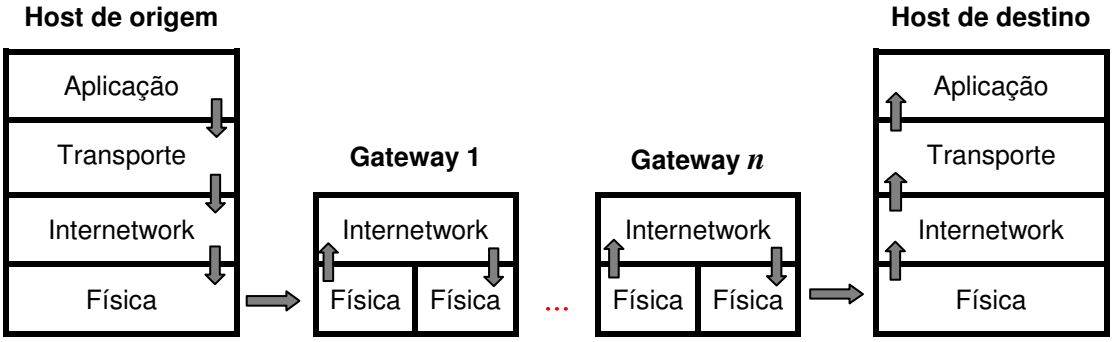

Figura 6 – Roteamento hop-by-hop

Para terminar esta seção, vamos então falar um pouco sobre como conseguir que as tabelas de rotas dos vários *gateways* fiquem coerentes.

A maneira mais simples é o(s) administrador(es) da rede configurarem manualmente as tabelas de rotas de todos os *gateways*. Esta abordagem é conhecida como roteamento estático (*static routing*). Infelizmente, a não ser que a rede seja de pequeno porte, este método é penoso e sujeito a falhas. Para redes TCP/IP de grande porte (como a Internet, ou redes privadas de grande porte) é necessário algo mais sofisticado.

Para estes casos, existe o roteamento dinâmico (*dynamic routing*), que é implementado usando protocolos de aplicação nos *gateways*, para permitir que eles negociem entre si, automaticamente, quais devem ser as entradas que irão constar na tabela de rotas de cada um.

Os protocolos para *dynamic routing* existem em dois "sabores": roteamento externo (*exterior routing*) e roteamento interno (*interior routing*), por causa da estrutura descentralizada de administração da Internet.

A maior unidade administrativa da infra-estrutura da Internet é um sistema autônomo (*autonomous system*), ou, simplesmente, AS. Para ser um AS, uma organização tem de

---

quando ele não consegue estabelecer conectividade física com o *host* de destino); *network unreachable* (emitida por um *gateway* na rota para indicar que ele não tem entrada para a *subnet* de destino na sua tabela de rotas); e *redirect* (emitida por um *gateway* na rota, para indicar que há uma opção melhor que ele para encaminhar o pacote). Além disso, o ICMP tem duas primitivas: ECHO e ECHO-REPLY, que são a base de funcionamento dos programas *ping* e *traceroute*, usados para testar a conectividade entre hosts e qual a rota que está sendo seguida pelos pacotes IP.

se registrar junto ao ICANN[12]. Se a solicitação for considerada válida, a organização ganha um AS *number* que a identifica, e um bloco CIDR de endereços IP para administrar dentro da sua área de atuação.

O objetivo dos protocolos para *exterior routing* é a negociação de rotas entre os diversos AS que compõem a Internet. Cada AS possui *gateways* de fronteira (*border gateways*) com outros AS. Os protocolos para *exterior routing* são executados nestes *border gateways*. O protocolo para *exterior routing* padronizado pelo IETF é o BGP (*border gateway protocol*). O protocolo BGP divide-se em dois aspectos: a troca de mensagens com border gateways de outros AS, eBGP (*exterior* BGP); e a troca de mensagens entre os border gateways do mesmo AS, iBGP (*interior* BGP)

Dentro da área administrativa do AS, ele tem liberdade para organizar o roteamento como achar melhor. Para este contexto são usados os protocolos de *interior routing*. Existem várias opções para *interior routing*. As mais conhecidas são: RIP (*route information protocol*), EIGRP (*enhanced interior gateway routing protocol*), OSPF (*open shortest path first*) e IS-IS (*intermediate system to intermediate system*)

Resta apenas um aspecto do *dynamic routing* a considerar: Qual critério cada *gateway* deve adotar para determinar que uma rota é melhor do que outra, e que a sua tabela de roteamento deve ser atualizada para refletir este fato?

Os protocolos para *dynamic routing*, neste aspecto, tem algumas coisas em comum e várias diferenças.

Primeiro as semelhanças. Cada *gateway*, ao iniciar sua operação, possui apenas informação sobre as *subnets* às quais ele tem acesso físico direto (isto está configurado em cada interface). Então ele faz anúncio (*advertisement*) desta informação, em todas as subnets a que esteja conectado, procurando por outros gateways adjacentes a ele – vizinhos (*neighbors*) – que falem o mesmo protocolo.

Cada *gateway* aprende, através dos *advertisements* recebidos dos seus *neighbors*, sobre a existência de novas *subnets* (um *hop* adiante), e incorpora as melhores opções de acesso à sua própria tabela de rotas. No próximo *advertisement* estas rotas aprendidas também são enviadas aos *neighbors*, de forma que, após algumas rodadas de *advertisement*, todos os *gateways* convergem suas tabelas de rotas para um conjunto coerente.

O tempo necessário para isto ocorrer é chamado de tempo de convergência do protocolo. Obtida a convergência, e na ausência de eventos que alterem a topologia da rede (ex.: quedas de *links* ou entrada em serviço de novos *gateways*), normalmente os gateways apenas trocarão periodicamente mensagens de *keepalive* com seus *neighbors*.

A falha em receber um certo número de *keepalives* consecutivos de um determinado *neighbor* faz com que ele seja considerado indisponível. O gateway, então, recalcula sua tabela de rotas e faz *advertisement* das alterações. O recebimento de um *advertisement*,

[12] Este processo é descentralizado, com organismos nacionais ou regionais "terceirizados" pelo ICANN para as tarefas burocráticas, inclusive esta. No Brasil, este papel é feito pelo registro.br, vinculado à RNP.

vindo de algum dos *neighbors* conhecidos ou de um novo *neighbor*, também faz com que o *gateway* recalcule sua tabela de rotas e faça *advertisement* das alterações.

A grande diferença entre os protocolos está no algoritmo usado para decidir se uma nova rota, recebida via *advertisement*, é melhor que alguma das rotas que já constam da tabela de rotas, e deve substituí-la.

Existem duas estratégias para que os protocolos de *dynamic routing* tomem esta decisão: vetor de distância (*distance vector*) ou estado de link (*link state*). O algoritmo mais conhecido para a estratégia *distance vector* é o *Bellman-Ford*, e para a estratégia *link state* o algoritmo mais conhecido é o SPF (*shortest path first*) de Dijkstra.

Dos protocolos que mencionamos, BGP, RIP e EIGRP seguem (com muitas variações de implementação) a estratégia *distance vector*. OSPF e IS-IS (também com variações) seguem a estratégia *link state*.

Ufa... Até que enfim, nosso pacote está pronto para remessa física para o próximo *host*.

Camada física:

Com relação à camada física, a arquitetura TCP/IP é agnóstica. Qualquer esquema de rede física (aquilo que, no modelo OSI-RM chamados de camadas de enlace e física) pode ser adaptado para transportar pacotes IP, e praticamente todos os tipos de redes físicas na praça já possuem esta implementação disponível.

Só existe um ponto que o padrão IETF exige nesta camada. Se a rede física utilizada for de meio compartilhado (ex.: redes locais IEEE 802.x), é necessário um mecanismo para associar o endereço físico da interface do *host* (no exemplo dado, conhecido como MAC[13] *address*) com o seu endereço IP.

O protocolo padrão nestes casos é o ARP (*address resolution protocol*). Cada *host* que executa o ARP mantém uma tabela associada à interface, chamada ARP cache. Cada entrada na ARP cache contém um endereço IP, o endereço físico associado a ele e um contador de *timeout*. A construção da ARP cache é feita assim:

a) Sempre que um pacote IP tem que ser transmitido na interface física, a ARP cache é consultada para saber se o endereço físico associado já é conhecido;

b) Se já existe uma entrada válida (com contador de *timeout* diferente de zero) para este endereço IP, o quadro (*frame*) físico é montado usando o endereço físico lá existente e enviado para transmissão, o contador de *timeout* desta entrada é incrementado novamente para o valor máximo, e o contador de timeout das outras entradas é decrementado;

c) Se não existe entrada para este endereço IP, ou se ela existe, mas é inválida (com contador de *timeout* igual a zero), é feito uma nova descoberta (*discovery*) do endereço físico associado, para criar ou revalidar a entrada na ARP cache.

---

[13] *Media access control*, definido pelo sub-comitê IEEE 802.2.

O *discovery* é feito pelo envio de um pacote ARP via *broadcast* físico, contendo o equivalente eletrônico da pergunta: "alguém aí é o dono do endereço IP a.b.c.d?" Todos os *hosts* na rede física recebem a pergunta e, se existir algum deles que satisfaça ao requisito, apenas ele responde, diretamente para quem perguntou. O *host* que iniciou o discovery, se receber resposta, usa o endereço físico de origem contido nela para criar uma nova entrada na sua ARP cache. Se não receber resposta, irá sinalizar erro para a camada internetwork (que gerará uma mensagem de erro ICMP).

A necessidade de um contador de timeout nas entradas da ARP cache é dupla: primeiro, *hosts* com os quais eu falo muito tendem a permanecer na tabela, sem necessidade de ficar repetindo o *discovery*. Segundo, hosts com os quais eu falo pouco não ficam "poluindo" desnecessariamente a tabela. Considerando que tanto o endereço IP quanto o endereço físico da interface do *host* de destino podem mudar, não é uma boa política manter entradas de *hosts* de baixo tráfego na ARP cache por muito tempo.

Próximos passos:

Esta é a visão geral da arquitetura TCP/IP, ainda sem a inclusão de características com *headiness index* menor que +5. Estas características adicionais apareceram em grande parte para poder suportar as características de aplicações com *headiness index* –3 ou abaixo, mas antes de chegar nelas, nosso próximo artigo vai dar uma olhada nos roteadores (*gateways* especialistas).

Ainda não falamos nada sobre como as características específicas desta ou daquela rede física irão afetar as aplicações de VoIP. Como já disse antes, ainda não é hora de ficar indevidamente fascinado pela camada física. Isto terá seu tempo, mais tarde.