Capítulo 6: Controle de Congestionamento

Problema: Alocação de Recursos

Até agora, vimos camadas suficientes da hierarquia do protocolo de rede para entender como os dados podem ser transferidos entre processos em redes heterogêneas. Agora, nos voltamos para um problema que abrange toda a pilha de protocolos: como alocar recursos de forma eficaz e justa entre um conjunto de usuários concorrentes. Os recursos compartilhados incluem a largura de banda dos links e os buffers nos roteadores ou switches onde os pacotes são enfileirados aguardando transmissão. Os pacotes *competem* em um roteador pelo uso de um link, com cada pacote em disputa colocado em uma fila aguardando sua vez de ser transmitido pelo link. Quando muitos pacotes competem pelo mesmo link, a fila fica cheia e duas coisas indesejáveis acontecem: os pacotes sofrem um atraso de ponta a ponta maior e, no pior caso, a fila transborda e os pacotes precisam ser descartados. Quando longas filas persistem e as quedas se tornam comuns, diz-se que a rede está *congestionada* . A maioria das redes fornece um mecanismo *de controle de congestionamento* para lidar com essa situação.

Controle de congestionamento e alocação de recursos são dois lados da mesma moeda. Por um lado, se a rede assume um papel ativo na alocação de recursos — por exemplo, agendando qual circuito virtual usará um determinado enlace físico durante um determinado período —, o congestionamento pode ser evitado, tornando o controle de congestionamento desnecessário. Alocar recursos de rede com precisão é difícil, no entanto, porque os recursos em questão são distribuídos por toda a rede; múltiplos enlaces conectando uma série de roteadores precisam ser agendados. Por outro lado, você sempre pode deixar as fontes de pacotes enviarem quantos dados desejarem e, em seguida, se recuperarem do congestionamento, caso ele ocorra. Essa é a

abordagem mais fácil, mas pode ser disruptiva, pois muitos pacotes podem ser descartados pela rede antes que o congestionamento possa ser controlado. Além disso, é precisamente nos momentos em que a rede está congestionada — ou seja, os recursos se tornaram escassos em relação à demanda — que a necessidade de alocação de recursos entre usuários concorrentes é mais sentida. Existem também soluções intermediárias, nas quais decisões de alocação imprecisas são tomadas, mas o congestionamento ainda pode ocorrer e, portanto, algum mecanismo ainda é necessário para se recuperar dele. Não importa se você chama essa solução mista de controle de congestionamento ou alocação de recursos. Em certo sentido, são ambos.

O controle de congestionamento e a alocação de recursos envolvem tanto hosts quanto elementos de rede, como roteadores. Em elementos de rede, diversas disciplinas de enfileiramento podem ser usadas para controlar a ordem em que os pacotes são transmitidos e quais pacotes são descartados. A disciplina de enfileiramento também pode segregar o tráfego para evitar que os pacotes de um usuário afetem indevidamente os pacotes de outro usuário. Nos hosts finais, o mecanismo de controle de congestionamento determina a velocidade com que as fontes podem enviar pacotes. Isso é feito para evitar que o congestionamento ocorra em primeiro lugar e, caso ocorra, para ajudar a eliminá-lo.

Este capítulo começa com uma visão geral do controle de congestionamento e da alocação de recursos. Em seguida, discutimos diferentes disciplinas de enfileiramento que podem ser implementadas nos roteadores dentro da rede, seguidas por uma descrição do algoritmo de controle de congestionamento fornecido pelo TCP nos hosts. A quarta seção explora diversas técnicas envolvendo roteadores e hosts que visam evitar o congestionamento antes que ele se torne um problema. Por fim, examinamos a ampla área da *qualidade de serviço* . Consideramos as necessidades das aplicações para receber diferentes níveis de alocação de recursos na rede e descrevemos

diversas maneiras pelas quais elas podem solicitar esses recursos e a rede pode atender às solicitações.

6.1 Problemas na alocação de recursos

A alocação de recursos e o controle de congestionamento são questões complexas que têm sido objeto de muitos estudos desde o projeto da primeira rede. Essas áreas ainda são áreas de pesquisa ativas. Um fator que torna essas questões complexas é que elas não estão isoladas em um único nível da hierarquia de protocolos. A alocação de recursos é parcialmente implementada nos roteadores, switches e links dentro da rede e parcialmente no protocolo de transporte executado nos hosts finais. Os sistemas finais podem usar protocolos de sinalização para transmitir seus requisitos de recursos aos nós da rede, que respondem com informações sobre a disponibilidade dos recursos. Um dos principais objetivos deste capítulo é definir uma estrutura na qual esses mecanismos possam ser compreendidos, bem como fornecer os detalhes relevantes sobre uma amostra representativa de mecanismos.

Devemos esclarecer nossa terminologia antes de prosseguir. Por *alocação de recursos* , entendemos o processo pelo qual os elementos de rede tentam atender às demandas concorrentes que os aplicativos têm por recursos de rede — principalmente largura de banda de link e espaço de buffer em roteadores ou switches. É claro que muitas vezes não será possível atender a todas as demandas, o que significa que alguns usuários ou aplicativos podem receber menos recursos de rede do que desejam. Parte do problema de alocação de recursos é decidir quando dizer não e a quem.

Usamos o termo *controle de congestionamento* para descrever os esforços feitos pelos nós da rede para prevenir ou responder a condições de sobrecarga. Como o congestionamento geralmente é ruim para todos, a primeira coisa a fazer é diminuir o congestionamento, ou preveni-lo. Isso pode ser alcançado simplesmente persuadindo

alguns hosts a interromper o envio, melhorando assim a situação para todos os outros. No entanto, é mais comum que mecanismos de controle de congestionamento tenham algum aspecto de justiça — ou seja, eles tentam compartilhar o sofrimento entre todos os usuários, em vez de causar grande sofrimento a alguns. Assim, vemos que muitos mecanismos de controle de congestionamento possuem algum tipo de alocação de recursos incorporada.

Também é importante entender a diferença entre controle de fluxo e controle de congestionamento. O controle de fluxo envolve impedir que um remetente rápido sobrecarregue um destinatário lento. O controle de congestionamento, por outro lado, visa impedir que um conjunto de remetentes envie dados em excesso *para a rede* devido à falta de recursos em algum momento. Esses dois conceitos são frequentemente confundidos; como veremos, eles também compartilham alguns mecanismos.

6.1.1 Modelo de Rede

Começamos definindo três características principais da arquitetura de rede. Em sua maior parte, este é um resumo do material apresentado nos capítulos anteriores e relevante para o problema da alocação de recursos.

Rede comutada por pacotes

Consideramos a alocação de recursos em uma rede comutada por pacotes (ou internet) composta por múltiplos enlaces e switches (ou roteadores). Como a maioria dos mecanismos descritos neste capítulo foram projetados para uso na internet e, portanto, originalmente definidos em termos de roteadores em vez de switches, usamos o termo *roteador* em toda a nossa discussão. O problema é essencialmente o mesmo, seja em uma rede ou em uma interconexão de redes.

Em tal ambiente, uma determinada fonte pode ter capacidade mais do que suficiente no link de saída imediato para enviar um pacote, mas em algum lugar no meio de uma rede seus pacotes encontram um link que está sendo usado por muitas fontes de tráfego diferentes. A Figura 152 ilustra essa situação — dois links de alta velocidade alimentam um link de baixa velocidade. Isso contrasta com redes de acesso compartilhado, como Ethernet e redes sem fio, onde a fonte pode observar diretamente o tráfego na rede e decidir, de acordo, se envia ou não um pacote. Já vimos os algoritmos usados para alocar largura de banda em redes de acesso compartilhado (por exemplo, Ethernet e Wi-Fi). Esses algoritmos de controle de acesso são, em certo sentido, análogos aos algoritmos de controle de congestionamento em uma rede comutada.

Conclusão principal

Observe que o controle de congestionamento é um problema diferente do roteamento. Embora seja verdade que um link congestionado possa receber um grande peso de borda pelo protocolo de roteamento e, como consequência, os roteadores o contornariam, "rotear ao redor" de um link congestionado geralmente não resolve o problema de congestionamento. Para ver isso, basta olhar para a rede simples representada na Figura 152 , onde todo o tráfego precisa fluir pelo mesmo roteador para chegar ao destino. Embora este seja um exemplo extremo, é comum haver um determinado roteador que não é possível contornar. Esse roteador pode ficar congestionado e não há nada que o mecanismo de roteamento possa fazer a respeito. Esse roteador congestionado às vezes é chamado de roteador *gargalo* . [Próximo]

Fluxos sem conexão

Em grande parte da nossa discussão, assumimos que a rede é essencialmente sem conexão, com qualquer serviço orientado a conexão implementado no protocolo de transporte em execução nos hosts finais. (Explicaremos a qualificação "essencialmente" em breve.) Este é precisamente o modelo da Internet, onde o IP

fornece um serviço de entrega de datagramas sem conexão e o TCP implementa uma abstração de conexão ponta a ponta. Observe que essa suposição não se aplica a redes de circuitos virtuais, como ATM e X.25. Nessas redes, uma mensagem de configuração de conexão atravessa a rede quando um circuito é estabelecido. Essa mensagem de configuração reserva um conjunto de buffers para a conexão em cada roteador, fornecendo assim uma forma de controle de congestionamento — uma conexão só é estabelecida se buffers suficientes puderem ser alocados a ela em cada roteador. A principal deficiência dessa abordagem é que ela leva à subutilização de recursos — buffers reservados para um circuito específico não estão disponíveis para uso por outro tráfego, mesmo que não estejam sendo usados por esse circuito. O foco deste capítulo são as abordagens de alocação de recursos que se aplicam a uma interconexão de redes e, portanto, nos concentramos principalmente em redes sem conexão.

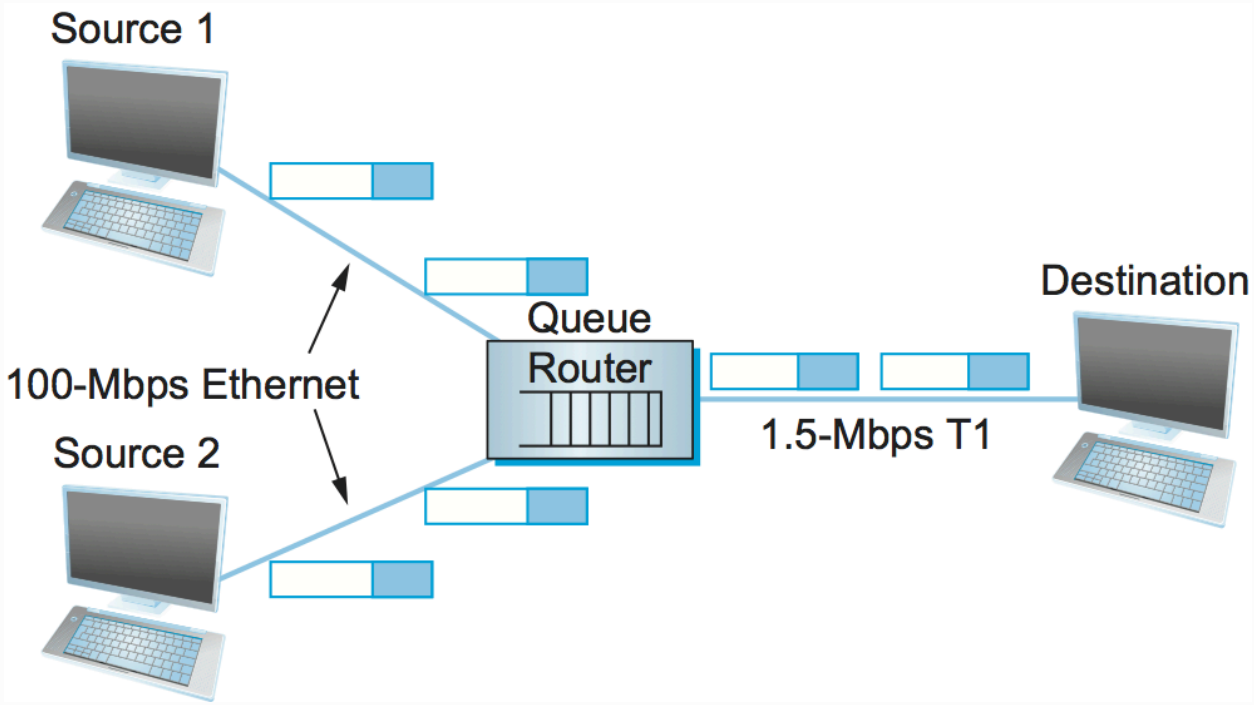

Figura 152. *Um possível roteador de gargalo.*

Precisamos qualificar o termo *"sem conexão"* porque nossa classificação de redes como sem conexão ou orientadas à conexão é um pouco restritiva demais; há uma área cinzenta entre elas. Em particular, a suposição de que todos os datagramas são completamente independentes em uma rede sem conexão é muito forte. Os datagramas certamente são comutados de forma independente, mas geralmente ocorre que um fluxo de datagramas entre um par específico de hosts flui através de um conjunto específico de roteadores. Essa ideia de *fluxo* — uma sequência de pacotes enviados entre um par origem/destino e seguindo a mesma rota pela rede — é uma abstração importante no contexto da alocação de recursos; é uma que usaremos neste capítulo.

Um dos benefícios da abstração de fluxo é que os fluxos podem ser definidos em diferentes granularidades. Por exemplo, um fluxo pode ser de host para host (ou seja, ter os mesmos endereços de host de origem/destino) ou de processo para processo (ou seja, ter os mesmos pares de host/porta de origem/destino). Neste último caso, um fluxo é essencialmente o mesmo que um canal, como temos usado esse termo ao longo deste livro. A razão pela qual introduzimos um novo termo é que um fluxo é visível para os roteadores dentro da rede, enquanto um canal é uma abstração de ponta a ponta. A Figura 153 ilustra vários fluxos passando por uma série de roteadores.

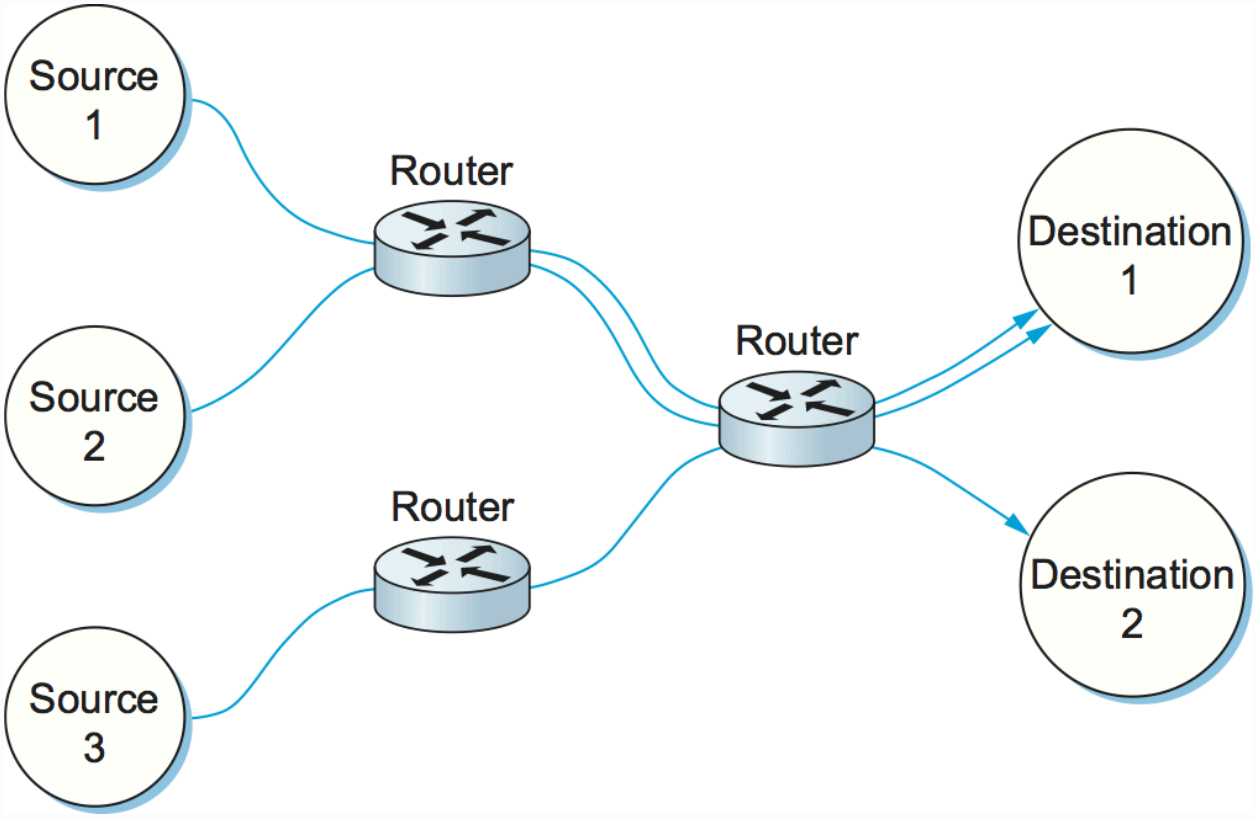

Figura 153. *Vários fluxos passando por um conjunto de roteadores.*

Como vários pacotes relacionados fluem por cada roteador, às vezes faz sentido manter algumas informações de estado para cada fluxo, informações que podem ser usadas para tomar decisões de alocação de recursos sobre os pacotes que pertencem ao fluxo. Esse estado às vezes é chamado de *estado suave* . A principal diferença entre o estado suave e o estado rígido é que o estado suave nem sempre precisa ser explicitamente criado e removido pela sinalização. O estado suave representa um meio-termo entre uma rede puramente sem conexão que não mantém *nenhum* estado nos roteadores e uma rede puramente orientada à conexão que mantém o estado rígido nos roteadores. Em geral, a operação correta da rede não depende da presença do estado suave (cada pacote ainda é roteado corretamente sem levar em conta esse estado), mas quando um pacote pertence a um fluxo para o qual o roteador está atualmente mantendo o estado suave, o roteador está mais apto a lidar com o pacote.

Observe que um fluxo pode ser definido implicitamente ou explicitamente estabelecido. No primeiro caso, cada roteador monitora os pacotes que trafegam entre o mesmo par origem/destino — o roteador faz isso inspecionando os endereços no cabeçalho — e trata esses pacotes como pertencentes ao mesmo fluxo para fins de controle de congestionamento. No segundo caso, a origem envia uma mensagem de configuração de fluxo pela rede, declarando que um fluxo de pacotes está prestes a iniciar. Embora fluxos explícitos não sejam diferentes de uma conexão em uma rede orientada a conexões, chamamos a atenção para este caso porque, mesmo quando explicitamente estabelecido, um fluxo não implica nenhuma semântica de ponta a ponta e, em particular, não implica a entrega confiável e ordenada de um circuito virtual. Ele existe simplesmente para fins de alocação de recursos. Veremos exemplos de fluxos implícitos e explícitos neste capítulo.

Modelo de Serviço

Na parte inicial deste capítulo, vamos nos concentrar nos mecanismos que pressupõem o modelo de serviço de melhor esforço da Internet. Com o serviço de melhor esforço, todos os pacotes recebem tratamento essencialmente igual, sem que os hosts finais tenham a oportunidade de solicitar à rede que alguns pacotes ou fluxos recebam certas garantias ou serviço preferencial. Definir um modelo de serviço que suporte algum tipo de serviço ou garantia preferencial — por exemplo, garantir a largura de banda necessária para um fluxo de vídeo — é o assunto de uma seção posterior. Diz-se que tal modelo de serviço fornece múltiplas *qualidades de serviço* (QoS). Como veremos, existe, na verdade, um espectro de possibilidades, que vão desde um modelo de serviço puramente de melhor esforço até um em que fluxos individuais recebem garantias quantitativas de QoS. Um dos maiores desafios é definir um modelo de serviço que atenda às necessidades de uma ampla gama de aplicações e até mesmo permita as aplicações que serão inventadas no futuro.

6.1.2 Taxonomia

Existem inúmeras diferenças entre os mecanismos de alocação de recursos, portanto, criar uma taxonomia completa é uma tarefa difícil. Por enquanto, descrevemos três dimensões pelas quais os mecanismos de alocação de recursos podem ser caracterizados; distinções mais sutis serão abordadas ao longo deste capítulo.

Centrado no roteador versus centrado no host

Os mecanismos de alocação de recursos podem ser classificados em dois grandes grupos: aqueles que abordam o problema de dentro da rede (ou seja, nos roteadores ou switches) e aqueles que o abordam das bordas da rede (ou seja, nos hosts, talvez dentro do protocolo de transporte). Como tanto os roteadores dentro da rede quanto os hosts nas bordas da rede participam da alocação de recursos, a verdadeira questão é onde recai a maior parte da carga.

Em um projeto centrado em roteador, cada roteador é responsável por decidir quando os pacotes são encaminhados e selecionar quais pacotes devem ser descartados, bem como por informar aos hosts que estão gerando o tráfego de rede quantos pacotes eles têm permissão para enviar. Em um projeto centrado em host, os hosts finais observam as condições da rede (por exemplo, quantos pacotes estão recebendo com sucesso pela rede) e ajustam seu comportamento de acordo. Observe que esses dois grupos não são mutuamente exclusivos. Por exemplo, uma rede que coloca a responsabilidade principal pelo gerenciamento do congestionamento nos roteadores ainda espera que os hosts finais sigam quaisquer mensagens de aviso enviadas pelos roteadores, enquanto os roteadores em redes que usam controle de congestionamento ponta a ponta ainda têm alguma política, não importa quão simples, para decidir quais pacotes descartar quando suas filas transbordam.

Baseado em reserva versus baseado em feedback

Uma segunda maneira pela qual os mecanismos de alocação de recursos são às vezes classificados é de acordo com se eles usam *reservas* ou *feedback* . Em um sistema baseado em reservas, alguma entidade (por exemplo, o host final) solicita à rede uma

certa quantidade de capacidade a ser alocada para um fluxo. Cada roteador então aloca recursos suficientes (buffers e/ou porcentagem da largura de banda do link) para atender a essa solicitação. Se a solicitação não puder ser atendida em algum roteador, porque isso comprometeria demais seus recursos, o roteador rejeita a reserva. Isso é análogo a receber um sinal de ocupado ao tentar fazer uma chamada telefônica. Em uma abordagem baseada em feedback, os hosts finais começam a enviar dados sem primeiro reservar qualquer capacidade e, em seguida, ajustam sua taxa de envio de acordo com o feedback que recebem. Esse feedback pode ser *explícito* (por exemplo, um roteador congestionado envia uma mensagem de "por favor, diminua a velocidade" para o host) ou *implícito* (por exemplo, o host final ajusta sua taxa de envio de acordo com o comportamento observável externamente da rede, como perdas de pacotes).

Observe que um sistema baseado em reservas sempre implica um mecanismo de alocação de recursos centrado no roteador. Isso ocorre porque cada roteador é responsável por monitorar quanta de sua capacidade está disponível no momento e decidir se novas reservas podem ser admitidas. Os roteadores também podem ter que garantir que cada host cumpra a reserva que fez. Se um host envia dados mais rápido do que alegou que enviaria quando fez a reserva, os pacotes desse host são bons candidatos para descarte, caso o roteador fique congestionado. Por outro lado, um sistema baseado em feedback pode implicar um mecanismo centrado no roteador ou no host. Normalmente, se o feedback for explícito, o roteador está envolvido, pelo menos em algum grau, no esquema de alocação de recursos. Se o feedback for implícito, quase toda a responsabilidade recai sobre o host final; os roteadores descartam silenciosamente os pacotes quando eles ficam congestionados.

As reservas não precisam ser feitas pelos hosts finais. É possível que um administrador de rede aloque recursos para fluxos ou para agregados maiores de tráfego, como veremos em uma seção posterior.

Baseado em janela versus baseado em taxa

Uma terceira maneira de caracterizar mecanismos de alocação de recursos é de acordo com se eles são *baseados em janela* ou *baseados em taxa* . Esta é uma das áreas, observadas acima, onde mecanismos e terminologia semelhantes são usados para controle de fluxo e controle de congestionamento. Ambos os mecanismos de controle de fluxo e alocação de recursos precisam de uma maneira de expressar, para o remetente, quantos dados ele tem permissão para transmitir. Há duas maneiras gerais de fazer isso: com uma *janela* ou com uma *taxa* . Já vimos protocolos de transporte baseados em janela, como o TCP, nos quais o receptor anuncia uma janela para o remetente. Esta janela corresponde a quanto espaço de buffer o receptor tem e limita a quantidade de dados que o remetente pode transmitir; ou seja, ela suporta controle de fluxo. Um mecanismo semelhante — anúncio de janela — pode ser usado dentro da rede para reservar espaço de buffer (ou seja, para suportar alocação de recursos). Os mecanismos de controle de congestionamento do TCP são baseados em janela.

Também é possível controlar o comportamento de um remetente usando uma taxa — ou seja, quantos bits por segundo o destinatário ou a rede são capazes de absorver. O controle baseado em taxa faz sentido para muitas aplicações multimídia, que tendem a gerar dados a uma taxa média e que precisam de pelo menos uma taxa de transferência mínima para serem úteis. Por exemplo, um codec de vídeo pode gerar vídeo a uma taxa média de 1 Mbps com uma taxa de pico de 2 Mbps. Como veremos mais adiante neste capítulo, a caracterização de fluxos baseada em taxa é uma escolha lógica em um sistema baseado em reserva que suporta diferentes qualidades de serviço — o remetente faz uma reserva para uma quantidade específica de bits por segundo, e cada roteador ao longo do caminho determina se pode suportar essa taxa, considerando os outros fluxos com os quais se comprometeu.

Resumo da Taxonomia de Alocação de Recursos

Classificar as abordagens de alocação de recursos em dois pontos diferentes ao longo de cada uma das três dimensões, como acabamos de fazer, parece sugerir até oito estratégias distintas. Embora oito abordagens diferentes sejam certamente possíveis,

observamos que, na prática, duas estratégias gerais parecem ser as mais prevalentes; essas duas estratégias estão vinculadas ao modelo de serviço subjacente da rede.

Por um lado, um modelo de serviço de melhor esforço geralmente implica que o feedback está sendo utilizado, uma vez que tal modelo não permite que os usuários reservem capacidade de rede. Isso, por sua vez, significa que a maior parte da responsabilidade pelo controle de congestionamento recai sobre os hosts finais, talvez com alguma assistência dos roteadores. Na prática, essas redes utilizam informações baseadas em janelas. Essa é a estratégia geral adotada na Internet.

Por outro lado, um modelo de serviço baseado em QoS provavelmente implica alguma forma de reserva. O suporte a essas reservas provavelmente exigirá um envolvimento significativo do roteador, como o enfileiramento de pacotes de forma diferente dependendo do nível de recursos reservados que eles exigem. Além disso, é natural expressar essas reservas em termos de taxa, uma vez que as janelas estão apenas indiretamente relacionadas à quantidade de largura de banda que um usuário precisa da rede. Discutiremos esse tópico em uma seção posterior.

6.1.3 Critérios de Avaliação

A questão final é saber se um mecanismo de alocação de recursos é bom ou não. Lembre-se de que, na declaração do problema no início deste capítulo, levantamos a questão de como uma rede aloca seus recursos *de forma eficaz* e *justa* . Isso sugere pelo menos duas medidas amplas pelas quais um esquema de alocação de recursos pode ser avaliado. Consideraremos cada uma delas separadamente.

Alocação eficaz de recursos

Um bom ponto de partida para avaliar a eficácia de um esquema de alocação de recursos é considerar as duas principais métricas de rede: taxa de transferência e atraso. Claramente, queremos a maior taxa de transferência e o menor atraso possível. Infelizmente, esses objetivos costumam ser um tanto conflitantes. Uma maneira segura

de um algoritmo de alocação de recursos aumentar a taxa de transferência é permitir a entrada do maior número possível de pacotes na rede, de modo a elevar a utilização de todos os links a até 100%. Faríamos isso para evitar a possibilidade de um link ficar ocioso, pois um link ocioso necessariamente prejudica a taxa de transferência. O problema com essa estratégia é que aumentar o número de pacotes na rede também aumenta o tamanho das filas em cada roteador. Filas maiores, por sua vez, significam que os pacotes ficam mais atrasados na rede.

Para descrever essa relação, alguns projetistas de rede propuseram o uso da razão entre taxa de transferência e atraso como métrica para avaliar a eficácia de um esquema de alocação de recursos. Essa razão é às vezes chamada de *potência* da rede:

```
Power = Throughput / Delay
```

Observe que não é óbvio que a potência seja a métrica correta para julgar a eficácia da alocação de recursos. Por um lado, a teoria por trás da potência baseia-se em uma rede de enfileiramento M/M/1 que pressupõe filas infinitas; redes reais têm buffers finitos e, às vezes, precisam descartar pacotes. Por outro lado, a potência é normalmente definida em relação a uma única conexão (fluxo); não está claro como ela se estende a múltiplas conexões concorrentes. Apesar dessas limitações bastante severas, no entanto, nenhuma alternativa obteve ampla aceitação e, portanto, a potência continua a ser usada.

1

Como este não é um livro sobre teoria de filas, fornecemos apenas esta breve descrição de uma fila M/M/1. O 1 significa que ela tem um único servidor, e os Ms significam que a distribuição dos tempos de chegada e de serviço dos pacotes é *markoviana,* ou seja, exponencial.

O objetivo é maximizar essa relação, que é uma função da carga que você coloca na rede. A carga, por sua vez, é definida pelo mecanismo de alocação de recursos. A Figura 154 apresenta uma curva de potência representativa, onde, idealmente, o mecanismo de alocação de recursos operaria no pico dessa curva. À esquerda do pico, o mecanismo está sendo muito conservador; ou seja, não está permitindo o envio de pacotes suficientes para manter os links ocupados. À direita do pico, tantos pacotes estão sendo permitidos na rede que o aumento do atraso devido ao enfileiramento está começando a dominar quaisquer pequenos ganhos de throughput.

Curiosamente, essa curva de potência se assemelha muito à curva de rendimento de um sistema de computador com compartilhamento de tempo. O rendimento do sistema melhora à medida que mais tarefas são admitidas no sistema, até atingir um ponto em que há tantas tarefas em execução que o sistema começa a apresentar sobrecarga (gasta todo o tempo trocando páginas de memória) e o rendimento começa a cair.

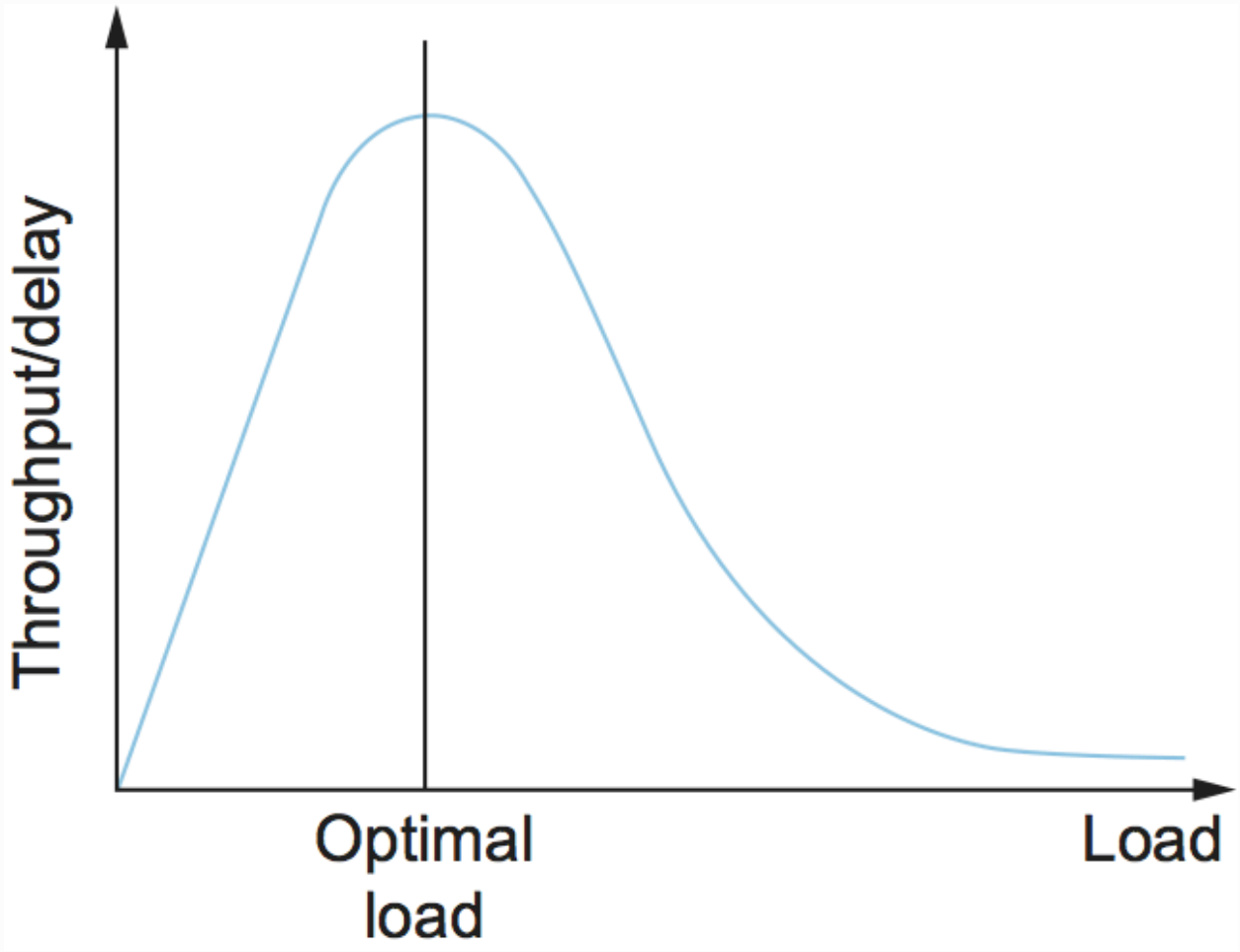

Figura 154. *Razão entre taxa de transferência e atraso em função da carga.*

Como veremos em seções posteriores deste capítulo, muitos esquemas de controle de congestionamento são capazes de controlar a carga apenas de maneiras muito rudimentares; ou seja, simplesmente não é possível girar o "botão" um pouco e permitir apenas um pequeno número de pacotes adicionais na rede. Como consequência, os projetistas de rede precisam se preocupar com o que acontece mesmo quando o sistema está operando sob carga extremamente pesada — isto é, na extremidade mais à direita da curva na Figura 154. Idealmente, gostaríamos de evitar a situação em que a taxa de transferência do sistema chega a zero porque o sistema está sofrendo uma sobrecarga. Na terminologia de rede, queremos um sistema estável — onde os pacotes continuam a passar pela rede mesmo quando a rede está operando sob carga pesada.

Se um mecanismo não for estável, a rede pode sofrer *um colapso de congestionamento* .

Alocação justa de recursos

A utilização eficaz dos recursos de rede não é o único critério para julgar um esquema de alocação de recursos. Devemos também considerar a questão da justiça. No entanto, rapidamente entramos em águas turvas quando tentamos definir o que exatamente constitui uma alocação justa de recursos. Por exemplo, um esquema de alocação de recursos baseado em reservas fornece uma maneira explícita de criar injustiça controlada. Com tal esquema, poderíamos usar reservas para permitir que um fluxo de vídeo receba 1 Mbps através de um link, enquanto uma transferência de arquivo recebe apenas 10 kbps pelo mesmo link.

Na ausência de informações explícitas em contrário, quando vários fluxos compartilham um link específico, gostaríamos que cada fluxo recebesse uma parcela igual da largura de banda. Essa definição pressupõe que uma parcela *justa* da largura de banda significa uma parcela *igual* da largura de banda. Mas, mesmo na ausência de reservas, parcelas iguais podem não ser equivalentes a parcelas justas. Devemos também considerar o comprimento dos caminhos comparados? Por exemplo, como ilustrado na Figura 155 , o que é justo quando um fluxo de quatro saltos está competindo com três fluxos de um salto?

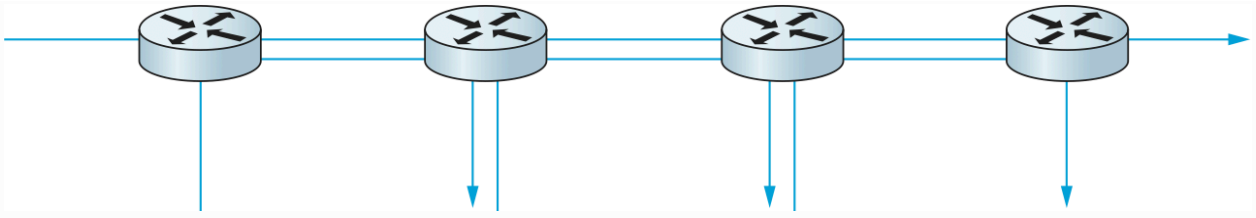

Figura 155. *Um fluxo de quatro saltos competindo com três fluxos de um salto.*

Partindo do princípio de que justo implica igual e que todos os caminhos têm o mesmo comprimento, o pesquisador de redes Raj Jain propôs uma métrica que pode ser usada para quantificar a imparcialidade de um mecanismo de controle de congestionamento. O índice de imparcialidade de Jain é definido da seguinte forma: Dado um conjunto de vazões de fluxo

$$(x1,x2,\ldots,xn)$$

(medido em unidades consistentes, como bits/segundo), a função a seguir atribui um índice de justiça aos fluxos:

$$f(x1,x2,\ldots,xn)=(\sum i=1nxi)2n\sum i=1nxi2$$

O índice de imparcialidade sempre resulta em um número entre 0 e 1, com 1 representando a maior imparcialidade. Para entender a intuição por trás dessa métrica, considere o caso em que todos *os n* fluxos recebem uma taxa de transferência de 1 unidade de dados por segundo. Podemos ver que o índice de imparcialidade neste caso é

$$n2n\times n=1$$

Agora, suponha que um fluxo receba uma taxa de transferência de

$$1+\Delta$$

. Agora o índice de justiça é

$$((n-1)+1+\Delta)2n(n-1+(1+\Delta)2)=n2+2n\Delta+\Delta2n2+2n\Delta+n\Delta2$$

Observe que o denominador excede o numerador em

$$(n-1)\Delta2$$

. Assim, se o fluxo ímpar estava ficando maior ou menor do que todos os outros fluxos (positivos ou negativos

$$\Delta$$

), o índice de justiça caiu abaixo de um. Outro caso simples a ser considerado é quando apenas *k* dos *n* fluxos recebem a mesma taxa de transferência, e os *nk* usuários restantes recebem taxa de transferência zero, caso em que o índice de justiça cai para k/n .

6.2 Disciplinas de Filas

Independentemente de quão simples ou sofisticado seja o restante do mecanismo de alocação de recursos, cada roteador deve implementar alguma disciplina de enfileiramento que governe como os pacotes são armazenados em buffer enquanto aguardam para serem transmitidos. O algoritmo de enfileiramento pode ser considerado como alocando tanto largura de banda (quais pacotes são transmitidos) quanto espaço em buffer (quais pacotes são descartados). Ele também afeta diretamente a latência experimentada por um pacote, determinando quanto tempo ele aguarda para ser transmitido. Esta seção apresenta dois algoritmos de enfileiramento comuns — primeiro a entrar, primeiro a sair (FIFO) e enfileiramento justo (FQ) — e identifica diversas variações que foram propostas.

6.2.1 FIFO

A ideia do enfileiramento FIFO, também chamado de enfileiramento por primeiro a chegar, primeiro a ser atendido (FCFS), é simples: o primeiro pacote que chega a um roteador é o primeiro pacote a ser transmitido. Isso é ilustrado na Figura 156(a) , que

mostra um FIFO com "slots" para armazenar até oito pacotes. Dado que a quantidade de espaço de buffer em cada roteador é finita, se um pacote chega e a fila (espaço de buffer) está cheia, o roteador descarta esse pacote, como mostrado na Figura 156(b) . Isso é feito sem considerar a qual fluxo o pacote pertence ou quão importante o pacote é. Isso às vezes é chamado de *descarte final* , pois os pacotes que chegam no final do FIFO são descartados.

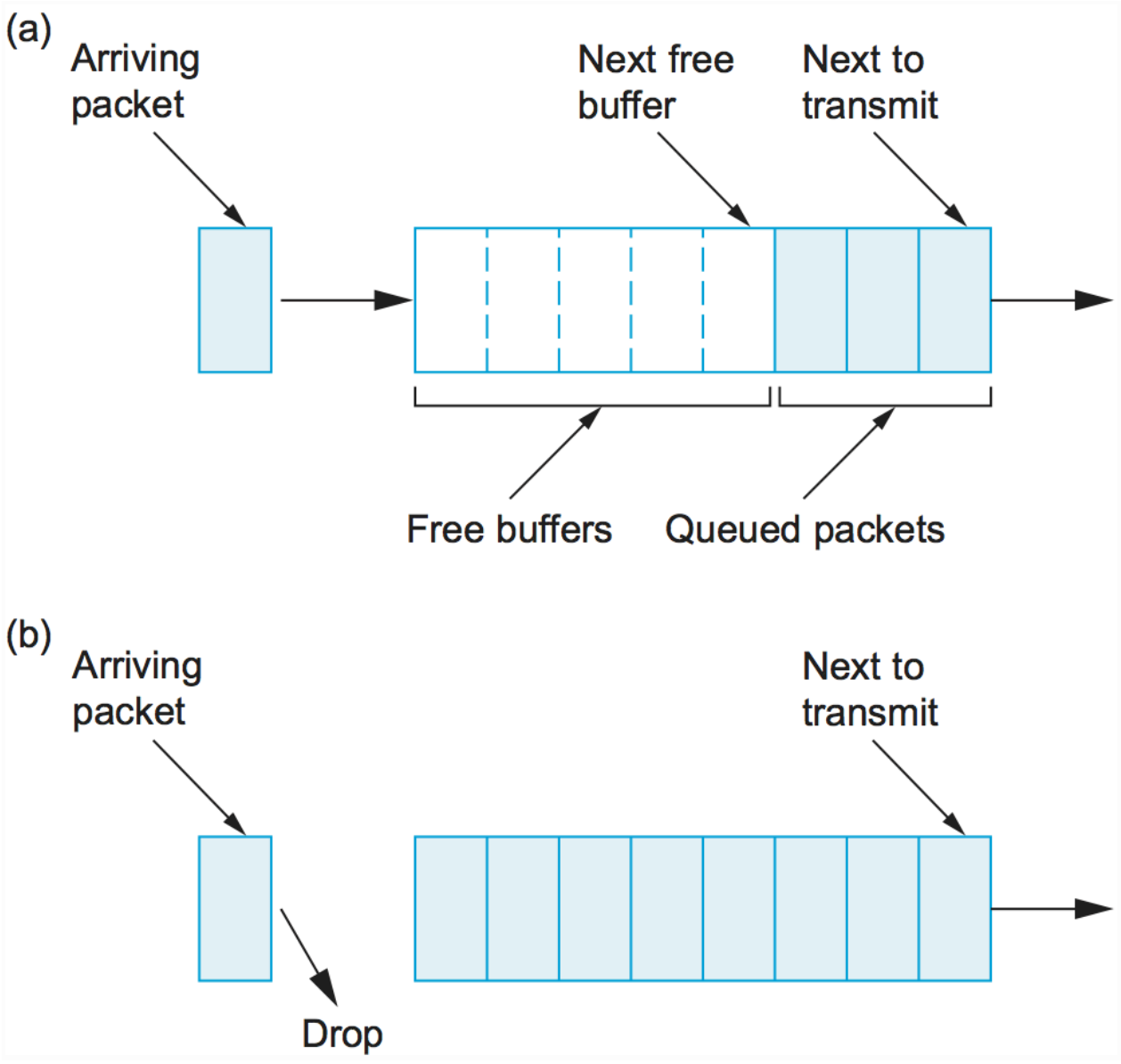

Figura 156. *Enfileiramento FIFO (a) e queda de cauda em uma fila FIFO (b).*

Note que tail drop e FIFO são duas ideias separáveis. FIFO é uma *disciplina de escalonamento* — ela determina a ordem em que os pacotes são transmitidos. Tail drop é uma *política de descarte* — ela determina quais pacotes são descartados. Como FIFO e tail drop são as instâncias mais simples de disciplina de escalonamento e política de descarte, respectivamente, eles são às vezes vistos como um pacote — a implementação de enfileiramento vanilla. Infelizmente, o pacote é frequentemente chamado simplesmente de *enfileiramento FIFO* , quando deveria ser chamado mais precisamente de *FIFO com tail drop* . Uma seção posterior fornece um exemplo de outra política de descarte, que usa um algoritmo mais complexo do que "Existe um buffer livre?" para decidir quando descartar pacotes. Tal política de descarte pode ser usada com FIFO ou com disciplinas de escalonamento mais complexas.

O FIFO com queda de cauda, por ser o mais simples de todos os algoritmos de enfileiramento, é o mais amplamente utilizado em roteadores de internet no momento em que este texto foi escrito. Essa abordagem simples de enfileiramento transfere toda a responsabilidade pelo controle de congestionamento e alocação de recursos para as bordas da rede. Assim, a forma predominante de controle de congestionamento na internet atualmente não exige ajuda dos roteadores: o TCP assume a responsabilidade de detectar e responder ao congestionamento. Veremos como isso funciona na próxima seção.

Uma variação simples do enfileiramento FIFO básico é o enfileiramento por prioridade. A ideia é marcar cada pacote com uma prioridade; a marca pode ser carregada, por exemplo, no cabeçalho IP, como discutiremos em uma seção posterior. Os roteadores então implementam múltiplas filas FIFO, uma para cada classe de prioridade. O roteador sempre transmite os pacotes da fila de maior prioridade se esta não estiver vazia antes de passar para a próxima fila de prioridade. Dentro de cada prioridade, os pacotes ainda são gerenciados de forma FIFO. Essa ideia é um pequeno desvio do modelo de entrega de melhor esforço, mas não chega a garantir nenhuma classe de prioridade específica. Ela apenas permite que os pacotes de alta prioridade passem para o início da fila.

O problema com o enfileiramento prioritário, é claro, é que a fila de alta prioridade pode sobrecarregar todas as outras filas; ou seja, enquanto houver pelo menos um pacote de alta prioridade na fila de alta prioridade, as filas de menor prioridade não serão atendidas. Para que isso seja viável, é necessário haver limites rígidos para a quantidade de tráfego de alta prioridade inserida na fila. Deve ficar imediatamente claro que não podemos permitir que os usuários definam seus próprios pacotes como alta prioridade de forma descontrolada; devemos impedi-los de fazer isso completamente ou fornecer alguma forma de "rejeição" aos usuários. Uma maneira óbvia de fazer isso é usar a economia — a rede poderia cobrar mais para entregar pacotes de alta prioridade do que pacotes de baixa prioridade. No entanto, existem desafios significativos para implementar tal esquema em um ambiente descentralizado como a Internet.

Uma situação em que o enfileiramento prioritário é usado na internet é para proteger os pacotes mais importantes — normalmente, as atualizações de roteamento necessárias para estabilizar as tabelas de roteamento após uma mudança de topologia. Frequentemente, há uma fila especial para esses pacotes, que pode ser identificada pelo Ponto de Código de Serviços Diferenciados (antigo campo TOS) no cabeçalho IP. Este é, na verdade, um caso simples da ideia de "Serviços Diferenciados".

6.2.2 Fila justa

O principal problema com o enfileiramento FIFO é que ele não discrimina entre diferentes fontes de tráfego ou, na linguagem apresentada na seção anterior, não separa os pacotes de acordo com o fluxo ao qual pertencem. Este é um problema em dois níveis diferentes. Em um nível, não está claro se qualquer algoritmo de controle de congestionamento implementado inteiramente na fonte será capaz de controlar adequadamente o congestionamento com tão pouca ajuda dos roteadores. Suspenderemos o julgamento sobre este ponto até a próxima seção, quando discutiremos o controle de congestionamento TCP. Em outro nível, como todo o mecanismo de controle de congestionamento é implementado nas fontes e o enfileiramento FIFO não fornece meios para monitorar a aderência das fontes a esse

mecanismo, é possível que uma fonte (fluxo) com mau comportamento capture uma fração arbitrariamente grande da capacidade da rede. Considerando a Internet novamente, é certamente possível que uma determinada aplicação não utilize TCP e, como consequência, ignore seu mecanismo de controle de congestionamento de ponta a ponta. (Aplicações como a telefonia pela Internet fazem isso hoje.) Tal aplicação é capaz de inundar os roteadores da Internet com seus próprios pacotes, fazendo com que os pacotes de outras aplicações sejam descartados.

O Fair Queuing (FQ) é um algoritmo desenvolvido para resolver esse problema. A ideia do FQ é manter uma fila separada para cada fluxo atualmente sendo processado pelo roteador. O roteador então atende essas filas em uma espécie de rodízio, conforme ilustrado na Figura 157. Quando um fluxo envia pacotes muito rápido, sua fila fica cheia. Quando uma fila atinge um determinado comprimento, os pacotes adicionais pertencentes à fila desse fluxo são descartados. Dessa forma, uma determinada fonte não pode aumentar arbitrariamente sua participação na capacidade da rede em detrimento de outros fluxos.

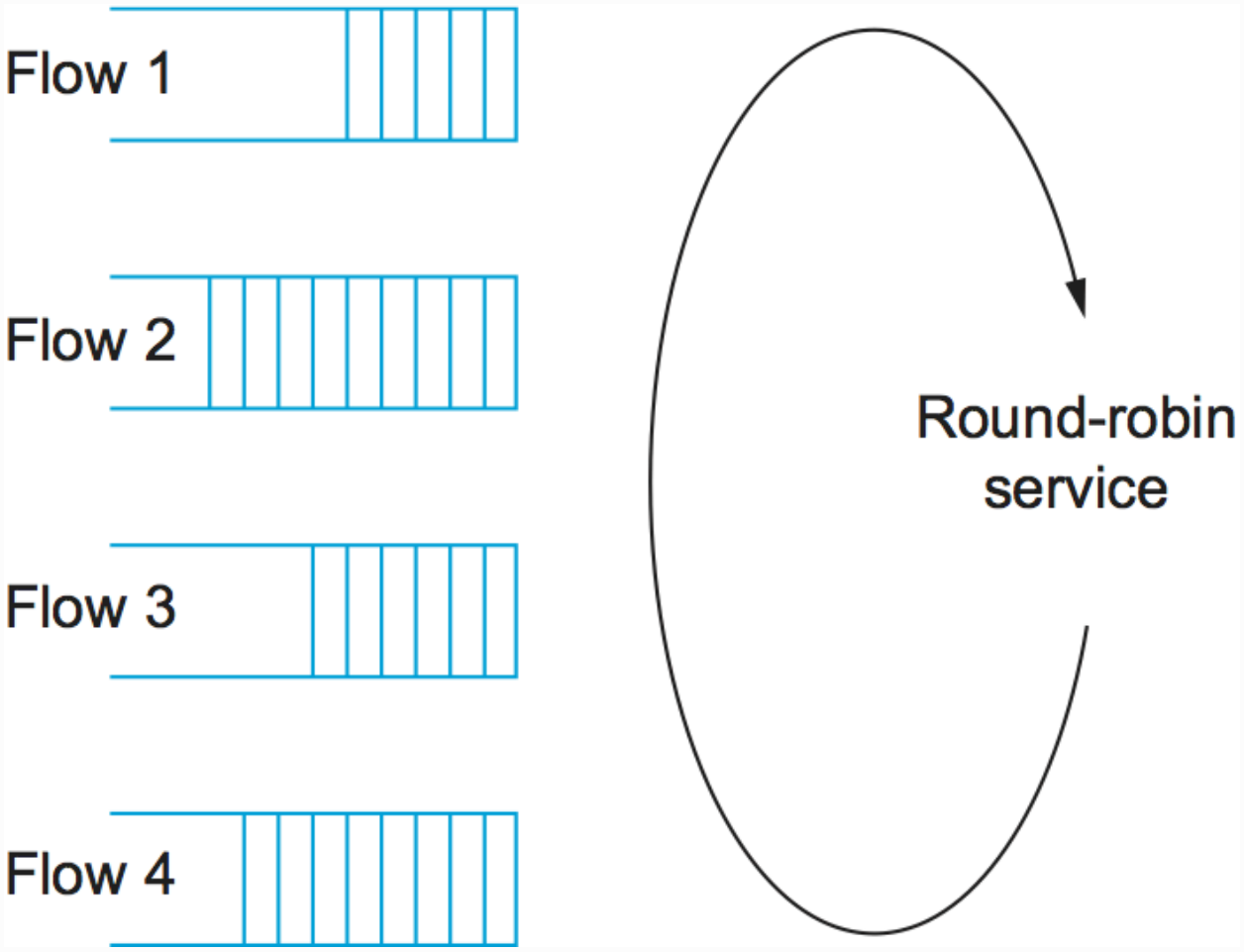

Figura 157. *Serviço round-robin de quatro fluxos em um roteador.*

Observe que o FQ não envolve o roteador informando às fontes de tráfego nada sobre o estado do roteador ou limitando de alguma forma a velocidade de envio de pacotes por uma determinada fonte. Em outras palavras, o FQ ainda é projetado para ser usado em conjunto com um mecanismo de controle de congestionamento de ponta a ponta. Ele simplesmente segrega o tráfego para que fontes de tráfego com mau comportamento não interfiram com aquelas que estão implementando fielmente o algoritmo de ponta a ponta. O FQ também impõe justiça entre um conjunto de fluxos gerenciados por um algoritmo de controle de congestionamento bem comportado.

Por mais simples que seja a ideia básica, ainda há um número modesto de detalhes que você precisa acertar. A principal complicação é que os pacotes processados em um roteador não têm necessariamente o mesmo comprimento. Para alocar verdadeiramente a largura de banda do link de saída de forma justa, é necessário levar em consideração o comprimento do pacote. Por exemplo, se um roteador estiver gerenciando dois fluxos, um com pacotes de 1.000 bytes e o outro com pacotes de 500 bytes (talvez devido à fragmentação a montante deste roteador), então um simples serviço round-robin de pacotes da fila de cada fluxo dará ao primeiro fluxo dois terços da largura de banda do link e ao segundo fluxo apenas um terço de sua largura de banda.

O que realmente queremos é um round-robin bit a bit, em que o roteador transmite um bit do fluxo 1, depois um bit do fluxo 2 e assim por diante. Claramente, não é viável intercalar os bits de pacotes diferentes. O mecanismo FQ, portanto, simula esse comportamento, determinando primeiro quando um determinado pacote terminaria de ser transmitido se estivesse sendo enviado usando round-robin bit a bit e, em seguida, usando esse tempo de término para sequenciar os pacotes para transmissão.

Para entender o algoritmo de aproximação do round-robin bit a bit, considere o comportamento de um único fluxo e imagine um relógio que marca uma vez cada vez que um bit é transmitido por todos os fluxos ativos. (Um fluxo está ativo quando possui dados na fila.) Para este fluxo, seja

P_i

denota o comprimento do pacote i , deixe

S_i

denota o momento em que o roteador começa a transmitir o pacote i e deixa

F_i

denota o momento em que o roteador termina de transmitir o pacote *i* . Se

$$P_i$$

é expresso em termos de quantos tiques de relógio são necessários para transmitir o pacote *i* (tendo em mente que o tempo avança 1 tique cada vez que esse fluxo recebe 1 bit de serviço), então é fácil ver que

$$F_i = S_i + P_i$$

.

Quando começamos a transmitir o pacote *i* ? A resposta a esta pergunta depende se o pacote *i* chegou antes ou depois de o roteador terminar de transmitir o pacote *i-1* deste fluxo. Se chegou antes, então logicamente o primeiro bit do pacote *i* é transmitido imediatamente após o último bit do pacote *i-1* . Por outro lado, é possível que o roteador tenha terminado de transmitir o pacote *i-1* muito antes de *i* chegar, o que significa que houve um período de tempo durante o qual a fila para este fluxo estava vazia, de modo que o mecanismo round-robin não conseguiu transmitir nenhum pacote deste fluxo. Se deixarmos

$$A_i$$

denota o momento em que o pacote *i* chega ao roteador, então

$$S_i = \max(F_i - 1, A_i)$$

. Assim, podemos calcular

$$F_i = \max(F_i - 1, A_i) + P_i$$

Agora passamos para a situação em que há mais de um fluxo e descobrimos que há um problema em determinar

Ai

Não podemos simplesmente ler o relógio quando o pacote chega. Como observado acima, queremos que o tempo avance um tique cada vez que todos os fluxos ativos recebem um bit de serviço no round-robin bit a bit, portanto, precisamos de um relógio que avance mais lentamente quando há mais fluxos. Especificamente, o relógio deve avançar um tique quando *n* bits são transmitidos se houver *n* fluxos ativos. Este relógio será usado para calcular

Ai

.

Agora, para cada fluxo, calculamos

Fi

para cada pacote que chega usando a fórmula acima. Em seguida, tratamos todos os

Fi

como registros de data e hora, e o próximo pacote a ser transmitido é sempre o pacote que tem o registro de data e hora mais baixo — o pacote que, com base no raciocínio acima, deve terminar a transmissão antes de todos os outros.

Observe que isso significa que um pacote pode chegar em um fluxo e, por ser mais curto do que um pacote de outro fluxo que já esteja na fila aguardando para ser transmitido, pode ser inserido na fila antes desse pacote mais longo. No entanto, isso não significa que um pacote recém-chegado possa preempcionar um pacote que está

sendo transmitido. É essa falta de preempção que impede a implementação do FQ recém-descrita de simular exatamente o esquema round-robin bit a bit que estamos tentando aproximar.

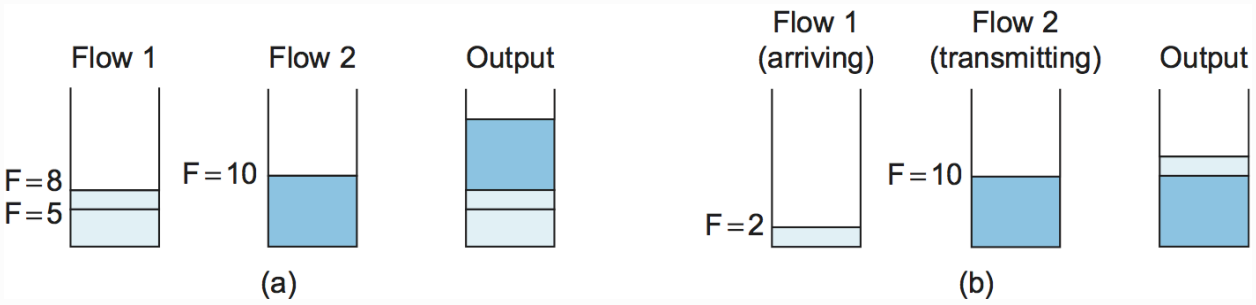

Figura 158. *Exemplo de enfileiramento justo em ação: (a) Pacotes com tempos de término anteriores são enviados primeiro; (b) o envio de um pacote já em andamento é concluído.*

Para entender melhor como essa implementação do enfileiramento justo funciona, considere o exemplo da Figura 158. A parte (a) mostra as filas para dois fluxos; o algoritmo seleciona ambos os pacotes do fluxo 1 para serem transmitidos antes do pacote na fila do fluxo 2, devido aos seus tempos de término mais curtos. Em (b), o roteador já começou a enviar um pacote do fluxo 2 quando o pacote do fluxo 1 chega. Embora o pacote que chega no fluxo 1 tivesse terminado antes do fluxo 2 se estivéssemos usando o enfileiramento justo bit a bit perfeito, a implementação não interrompe o pacote do fluxo 2.

Há duas coisas a serem observadas sobre o enfileiramento justo. Primeiro, o enlace nunca fica ocioso enquanto houver pelo menos um pacote na fila. Qualquer esquema de enfileiramento com essa característica é considerado *conservador de trabalho* . Um efeito de ser conservador de trabalho é que, se eu estiver compartilhando um enlace com muitos fluxos que não estão enviando dados, posso usar toda a capacidade do enlace para o meu fluxo. Assim que os outros fluxos começarem a enviar, no entanto, eles começarão a usar sua parte e a capacidade disponível para o meu fluxo diminuirá.

A segunda coisa a observar é que, se o link estiver totalmente carregado e houver *n* fluxos enviando dados, não posso usar mais do que 1/n- ésimo da largura de banda do link. Se eu tentar enviar mais do que isso, meus pacotes receberão carimbos de data/hora cada vez maiores, fazendo com que fiquem na fila por mais tempo, aguardando transmissão. Eventualmente, a fila transbordará — embora a decisão de descartar meus pacotes ou os de outra pessoa não seja determinada pelo fato de estarmos usando o enfileiramento justo. Isso é determinado pela política de descarte; FQ é um algoritmo de escalonamento que, como FIFO, pode ser combinado com várias políticas de descarte.

Como o FQ conserva o trabalho, qualquer largura de banda não utilizada por um fluxo fica automaticamente disponível para outros fluxos. Por exemplo, se tivermos quatro fluxos passando por um roteador e todos estiverem enviando pacotes, cada um receberá um quarto da largura de banda. No entanto, se um deles ficar ocioso por tempo suficiente para que todos os seus pacotes sejam drenados da fila do roteador, a largura de banda disponível será compartilhada entre os três fluxos restantes, que agora receberão um terço da largura de banda cada. Assim, podemos pensar no FQ como um provedor de uma parcela mínima garantida de largura de banda para cada fluxo, com a possibilidade de obter mais do que a garantia se outros fluxos não estiverem usando suas parcelas.

É possível implementar uma variação do FQ, chamada de *enfileiramento justo ponderado* (WFQ), que permite atribuir um peso a cada fluxo (fila). Esse peso especifica logicamente quantos bits transmitir a cada vez que o roteador atende a fila, o que controla efetivamente a porcentagem da largura de banda do link que esse fluxo receberá. O FQ simples atribui a cada fila o peso 1, o que significa que, logicamente, apenas 1 bit é transmitido de cada fila a cada vez. Isso resulta em cada fluxo recebendo

1/nth

da largura de banda quando há *n* fluxos. Com o WFQ, no entanto, uma fila pode ter peso 2, uma segunda fila pode ter peso 1 e uma terceira fila pode ter peso 3. Supondo que cada fila sempre contenha um pacote aguardando para ser transmitido, o primeiro fluxo receberá um terço da largura de banda disponível, o segundo receberá um sexto da largura de banda disponível e o terceiro receberá metade da largura de banda disponível.

Embora tenhamos descrito o WFQ em termos de fluxos, observe que ele poderia ser implementado em *classes* de tráfego, onde as classes são definidas de alguma forma diferente dos fluxos simples apresentados no início deste capítulo. Por exemplo, poderíamos usar alguns bits no cabeçalho IP para identificar classes e alocar uma fila e um peso para cada classe. Isso é exatamente o que é proposto como parte da arquitetura de Serviços Diferenciados descrita em uma seção posterior.

Observe que um roteador que executa WFQ precisa aprender quais pesos atribuir a cada fila de algum lugar, seja por configuração manual ou por algum tipo de sinalização das fontes. Neste último caso, estamos caminhando para um modelo baseado em reserva. A simples atribuição de um peso a uma fila fornece uma forma bastante fraca de reserva, pois esses pesos estão apenas indiretamente relacionados à largura de banda que o fluxo recebe. (A largura de banda disponível para um fluxo também depende, por exemplo, de quantos outros fluxos compartilham o link.) Veremos em uma seção posterior como o WFQ pode ser usado como um componente de um mecanismo de alocação de recursos baseado em reserva.

Conclusão principal

Por fim, observamos que toda essa discussão sobre gerenciamento de filas ilustra um importante princípio de design de sistemas conhecido como *separação entre política e mecanismo* . A ideia é visualizar cada mecanismo como uma caixa opaca que fornece um serviço multifacetado que pode ser controlado por um conjunto de botões. Uma política especifica uma configuração específica desses botões, mas não sabe (ou se

importa) com a forma como a política é implementada. Nesse caso, o mecanismo em questão é a disciplina de enfileiramento, e a política é uma configuração específica de qual fluxo obtém qual nível de serviço (por exemplo, prioridade ou peso). Discutiremos algumas políticas que podem ser usadas com o mecanismo WFQ em uma seção posterior. [Próximo]

6.3 Controle de Congestionamento TCP

Esta seção descreve o exemplo predominante de controle de congestionamento ponta a ponta em uso atualmente, implementado pelo TCP. A estratégia essencial do TCP é enviar pacotes para a rede sem reserva e, em seguida, reagir a eventos observáveis que ocorram. O TCP pressupõe apenas o enfileiramento FIFO nos roteadores da rede, mas também funciona com outras estratégias de enfileiramento.

O controle de congestionamento TCP foi introduzido na internet no final da década de 1980 por Van Jacobson, aproximadamente oito anos após a pilha de protocolos TCP/IP entrar em operação. Imediatamente antes dessa época, a internet sofria de um colapso de congestionamento — os hosts enviavam seus pacotes para a internet tão rápido quanto a janela anunciada permitia, ocorria congestionamento em algum roteador (causando a perda de pacotes) e os hosts atingiam o tempo limite e retransmitiam seus pacotes, resultando em ainda mais congestionamento.

Em termos gerais, a ideia do controle de congestionamento do TCP é que cada fonte determine quanta capacidade está disponível na rede, para que saiba quantos pacotes pode ter em trânsito com segurança. Uma vez que uma determinada fonte tenha essa quantidade de pacotes em trânsito, ela usa a chegada de um ACK como um sinal de

que um de seus pacotes deixou a rede e que, portanto, é seguro inserir um novo pacote na rede sem aumentar o nível de congestionamento. Ao usar ACKs para controlar o ritmo da transmissão de pacotes, o TCP é considerado *autocronometrado* . É claro que determinar a capacidade disponível em primeiro lugar não é uma tarefa fácil. Para piorar a situação, como outras conexões vêm e vão, a largura de banda disponível muda ao longo do tempo, o que significa que qualquer fonte deve ser capaz de ajustar o número de pacotes que tem em trânsito. Esta seção descreve os algoritmos usados pelo TCP para resolver esses e outros problemas.

Observe que, embora descrevamos os mecanismos de controle de congestionamento do TCP individualmente, dando a impressão de que estamos falando de três mecanismos independentes, é somente quando considerados em conjunto que temos o controle de congestionamento do TCP. Além disso, embora comecemos aqui com a variante de controle de congestionamento do TCP mais frequentemente chamada de *TCP padrão* , veremos que, na verdade, existem diversas variantes de controle de congestionamento do TCP em uso atualmente, e os pesquisadores continuam a explorar novas abordagens para lidar com esse problema. Algumas dessas novas abordagens são discutidas a seguir.

6.3.1 Aumento aditivo/diminuição multiplicativa

O TCP mantém uma nova variável de estado para cada conexão, chamada `CongestionWindow`, que é usada pela fonte para limitar a quantidade de dados que ela pode ter em trânsito em um determinado momento. A janela de congestionamento é a contrapartida do controle de congestionamento para a janela anunciada do controle de fluxo. O TCP é modificado de forma que o número máximo de bytes de dados não confirmados permitidos seja agora o mínimo entre a janela de congestionamento e a janela anunciada. Assim, usando as variáveis definidas no capítulo anterior, a janela efetiva do TCP é revisada da seguinte forma:

```
MaxWindow = MIN(CongestionWindow, AdvertisedWindow)

EffectiveWindow = MaxWindow -  (LastByteSent - LastByteAcked)
```

Ou seja, `MaxWindow`substitui `AdvertisedWindow`no cálculo de `EffectiveWindow`. Assim, uma origem TCP não pode enviar mais rápido do que o componente mais lento — a rede ou o host de destino — pode acomodar.

O problema, claro, é como o TCP aprende um valor apropriado para `CongestionWindow`. Ao contrário do `AdvertisedWindow`, que é enviado pelo lado receptor da conexão, não há ninguém para enviar um valor adequado `CongestionWindow`para o lado emissor do TCP. A resposta é que a fonte TCP define o `CongestionWindow`com base no nível de congestionamento que percebe existir na rede. Isso envolve diminuir a janela de congestionamento quando o nível de congestionamento aumenta e aumentar a janela de congestionamento quando o nível de congestionamento diminui. Em conjunto, o mecanismo é comumente chamado de *aumento aditivo/diminuição multiplicativa* (AIMD); a razão para esse nome complicado ficará clara a seguir.

A questão-chave, então, é como a fonte determina que a rede está congestionada e que deve diminuir a janela de congestionamento? A resposta se baseia na observação de que o principal motivo pelo qual os pacotes não são entregues, resultando em um timeout, é que um pacote foi descartado devido a congestionamento. É raro que um pacote seja descartado devido a um erro durante a transmissão. Portanto, o TCP interpreta timeouts como um sinal de congestionamento e reduz a taxa de transmissão. Especificamente, cada vez que ocorre um timeout, a fonte define o `CongestionWindow`valor anterior pela metade. Essa redução pela metade do tempo `CongestionWindow`limite para cada timeout corresponde à parte de "redução multiplicativa" do AIMD.

Embora `CongestionWindow`seja definido em termos de bytes, é mais fácil entender a diminuição multiplicativa se pensarmos em termos de pacotes inteiros. Por exemplo, suponha que o `CongestionWindow`tamanho atual seja de 16 pacotes. Se uma perda for detectada, `CongestionWindow` o tamanho é definido como 8. (Normalmente, uma perda é detectada quando ocorre um timeout, mas, como vemos abaixo, o TCP possui outro mecanismo para detectar pacotes perdidos.) Perdas adicionais fazem com `CongestionWindow`que o tamanho seja reduzido para 4, depois para 2 e, finalmente, para 1 pacote. `CongestionWindow`Não é permitido que o tamanho do segmento seja inferior ao de um único pacote ou, na terminologia TCP, ao *tamanho máximo do segmento* .

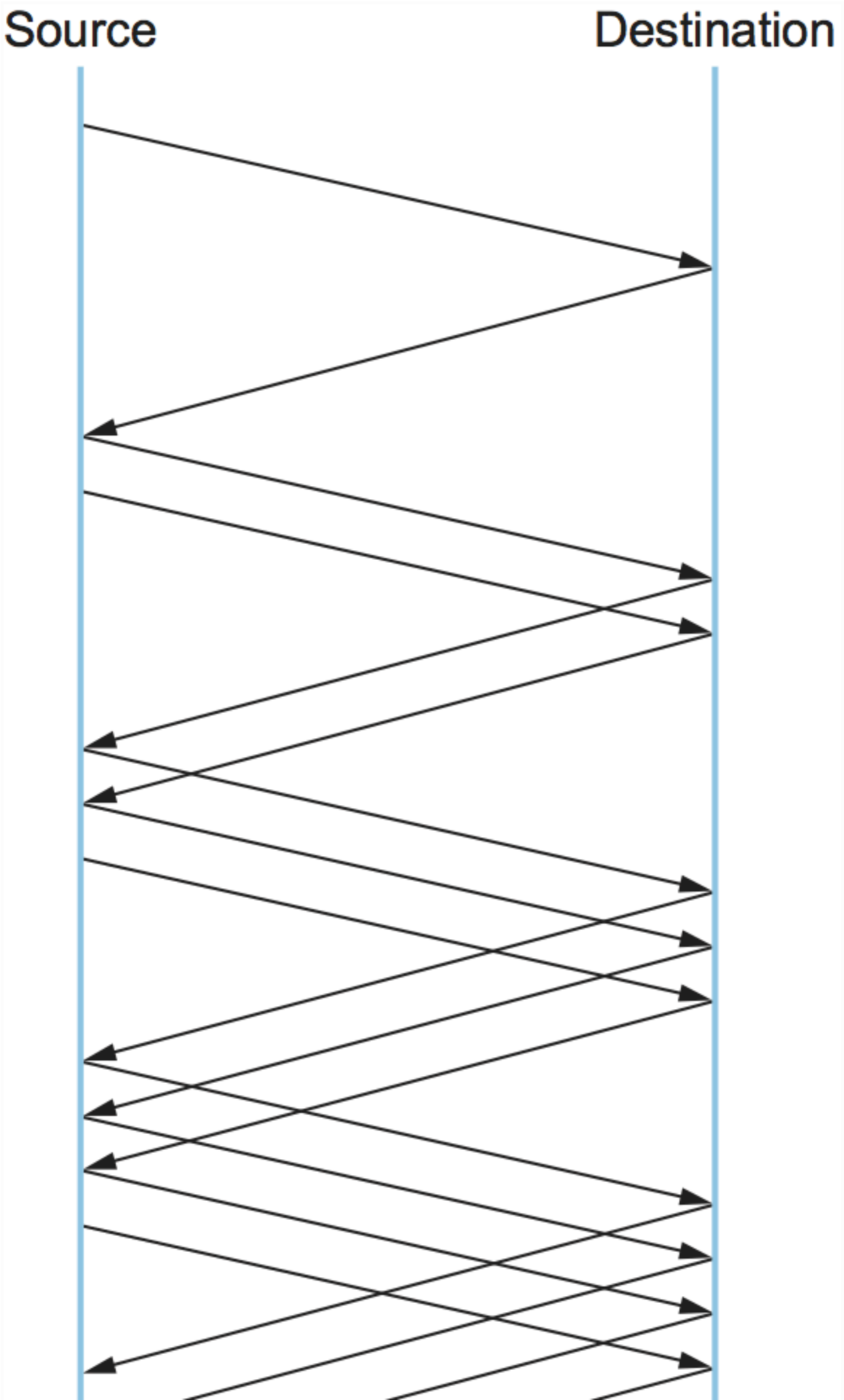
Source
Destination

Figura 159. *Pacotes em trânsito durante aumento aditivo, com um pacote sendo adicionado a cada RTT.*

Uma estratégia de controle de congestionamento que apenas reduz o tamanho da janela é obviamente muito conservadora. Também precisamos ser capazes de aumentar a janela de congestionamento para aproveitar a capacidade recém-disponível na rede. Esta é a parte de "aumento aditivo" do AIMD e funciona da seguinte maneira. Toda vez que a origem envia com sucesso um pacote equivalente`CongestionWindow` a _____ — ou seja, cada pacote enviado durante o último tempo de ida e volta (RTT) foi ACKed — ela adiciona o equivalente a 1 pacote a _ ...`CongestionWindowCongestionWindow`

```
Increment = MSS x (MSS/CongestionWindow)
```

```
CongestionWindow += Increment
```

Ou seja, em vez de incrementar `CongestionWindow`em um byte inteiro `MSS`cada RTT, incrementamos em uma fração de `MSS`cada vez que um ACK é recebido. Supondo que cada ACK confirme o recebimento de `MSS`bytes, essa fração é `MSS/CongestionWindow`.

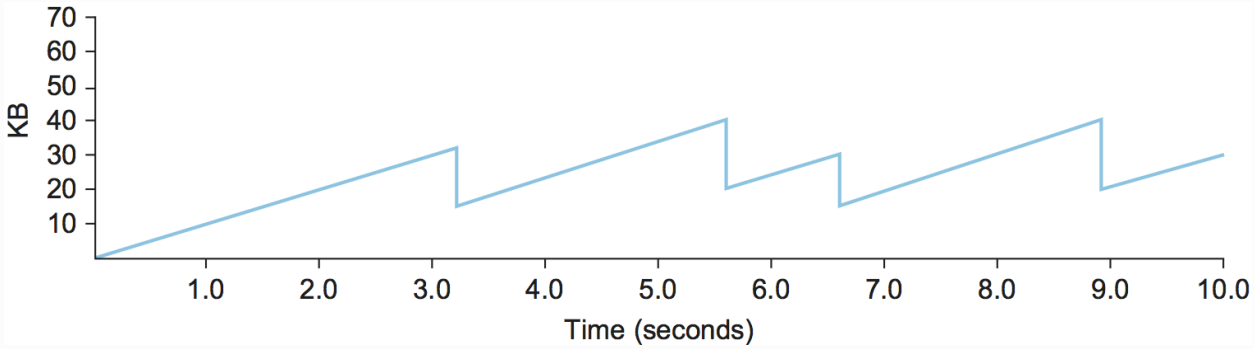

Figura 160. *Padrão dente de serra típico do TCP.*

Esse padrão de aumento e diminuição contínuos da janela de congestionamento continua ao longo da vida útil da conexão. De fato, se você plotar o valor atual de `CongestionWindow`como uma função do tempo, obterá um padrão dente de serra, conforme ilustrado na Figura 160. O conceito importante a ser entendido sobre AIMD é que a fonte está disposta a reduzir sua janela de congestionamento a uma taxa muito mais rápida do que está disposta a aumentá-la. Isso contrasta com uma estratégia de aumento/diminuição aditivos, na qual a janela seria aumentada em 1 pacote quando um ACK chega e diminuída em 1 quando ocorre um timeout. Foi demonstrado que AIMD é uma condição necessária para que um mecanismo de controle de congestionamento seja estável.

Uma explicação intuitiva para o motivo pelo qual o TCP diminui a janela agressivamente e a aumenta de forma conservadora é que as consequências de ter uma janela muito grande são agravadas. Isso ocorre porque, quando a janela é muito grande, os pacotes descartados serão retransmitidos, agravando ainda mais o congestionamento. É importante sair dessa situação rapidamente.

Por fim, como um tempo limite é uma indicação de congestionamento que desencadeia uma diminuição multiplicativa, o TCP precisa do mecanismo de tempo limite mais preciso possível. Já abordamos o mecanismo de tempo limite do TCP em um capítulo anterior, portanto, não o repetiremos aqui. Os dois principais pontos a serem lembrados sobre esse mecanismo são que (1) os tempos limite são definidos em função tanto do RTT médio quanto do desvio padrão dessa média e (2) devido ao custo de medir cada transmissão com um relógio preciso, o TCP amostra o tempo de ida e volta apenas uma vez por RTT (em vez de uma vez por pacote) usando um relógio de granulação grossa (500 ms).

6.3.2 Início lento

O mecanismo de aumento aditivo que acabamos de descrever é a abordagem correta a ser usada quando a fonte está operando próxima à capacidade disponível da rede, mas demora muito para aumentar a velocidade de uma conexão quando ela está

começando do zero. O TCP, portanto, fornece um segundo mecanismo, ironicamente chamado de *início lento* , que é usado para aumentar rapidamente a janela de congestionamento a partir de um início a frio. O início lento efetivamente aumenta a janela de congestionamento exponencialmente, em vez de linearmente.

Especificamente, a origem começa definindo `CongestionWindow`um pacote. Quando o ACK para esse pacote chega, o TCP adiciona 1 `CongestionWindow`e envia dois pacotes. Ao receber os dois ACKs correspondentes, o TCP incrementa `CongestionWindow`em 2 — um para cada ACK — e, em seguida, envia quatro pacotes. O resultado final é que o TCP efetivamente dobra o número de pacotes em trânsito a cada RTT. A Figura 161 mostra o crescimento do número de pacotes em trânsito durante o início lento. Compare isso com o crescimento linear do aumento aditivo ilustrado na Figura 159 .

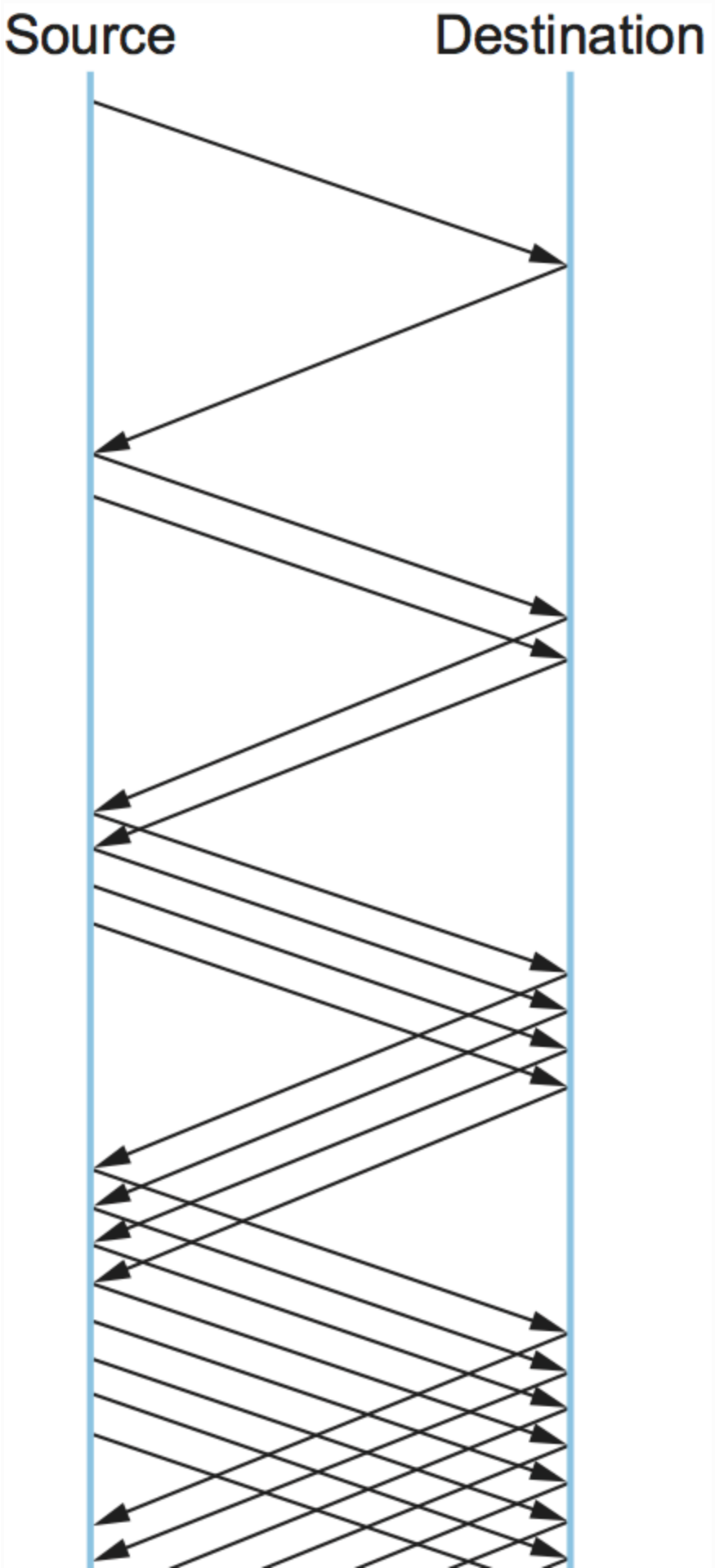
Source
Destination

Figura 161. *Pacotes em trânsito durante inicialização lenta.*

Por que qualquer mecanismo exponencial seria chamado de "lento" é intrigante à primeira vista, mas pode ser explicado se colocado no contexto histórico adequado. Precisamos comparar o início lento não com o mecanismo linear da subseção anterior, mas com o comportamento original do TCP. Considere o que acontece quando uma conexão é estabelecida e a origem começa a enviar pacotes — ou seja, quando não há pacotes em trânsito. Se a origem enviar tantos pacotes quanto a janela anunciada permitir — que é exatamente o que o TCP fazia antes do início lento ser desenvolvido —, mesmo que haja uma quantidade razoavelmente grande de largura de banda disponível na rede, os roteadores podem não ser capazes de consumir essa rajada de pacotes. Tudo depende de quanto espaço de buffer está disponível nos roteadores. O início lento foi, portanto, projetado para espaçar os pacotes de forma que essa rajada não ocorra. Em outras palavras, embora seu crescimento exponencial seja mais rápido do que o crescimento linear, o início lento é muito "mais lento" do que enviar dados de uma janela anunciada inteira de uma só vez.

Na verdade, existem duas situações diferentes em que o início lento ocorre. A primeira é logo no início de uma conexão, momento em que a origem não tem ideia de quantos pacotes poderá ter em trânsito em um determinado momento. (Lembre-se de que hoje o TCP opera em tudo, desde links de 1 Mbps até links de 40 Gbps, portanto, não há como a origem saber a capacidade da rede.) Nessa situação, o início lento continua a dobrar `CongestionWindow`cada RTT até que haja uma perda, momento em que um tempo limite causa a diminuição multiplicativa, que se divide `CongestionWindow`por 2.

A segunda situação em que o início lento é usado é um pouco mais sutil; ocorre quando a conexão cai enquanto aguarda a ocorrência de um tempo limite. Lembre-se de como funciona o algoritmo de janela deslizante do TCP — quando um pacote é perdido, a origem eventualmente atinge um ponto em que enviou a quantidade de dados permitida pela janela anunciada e, portanto, bloqueia enquanto aguarda um ACK que não chegará. Eventualmente, ocorre um tempo limite, mas nesse momento não há pacotes em trânsito, o que significa que a origem não receberá ACKs para

"cronometrar" a transmissão de novos pacotes. Em vez disso, a origem receberá um único ACK cumulativo que reabre toda a janela anunciada, mas, como explicado acima, a origem usa o início lento para reiniciar o fluxo de dados em vez de despejar os dados de uma janela inteira na rede de uma só vez.

Embora a origem esteja usando o início lento novamente, ela agora sabe mais informações do que no início de uma conexão. Especificamente, a origem tem um valor atual (e útil) de `CongestionWindow`; este é o valor de `CongestionWindow`que existia antes da última perda de pacotes, dividido por 2 como resultado da perda. Podemos pensar nisso como a janela de congestionamento *de destino* . O início lento é usado para aumentar rapidamente a taxa de envio até esse valor e, em seguida, o aumento aditivo é usado além desse ponto. Observe que temos um pequeno problema de contabilidade para cuidar, pois queremos lembrar a janela de congestionamento de destino resultante da diminuição multiplicativa, bem como a janela de congestionamento *real*`CongestionThreshold` que está sendo usada pelo início lento. Para resolver esse problema, o TCP introduz uma variável temporária para armazenar a janela de destino, normalmente chamada de , que é definida como igual ao `CongestionWindow`valor que resulta da diminuição multiplicativa. A variável `CongestionWindow`é então redefinida para um pacote e é incrementada em um pacote para cada ACK recebido até atingir `CongestionThreshold`, ponto em que é incrementada em um pacote por RTT.

Em outras palavras, o TCP aumenta a janela de congestionamento, conforme definido pelo seguinte fragmento de código:

```
{


    u_int    cw = state->CongestionWindow;


    u_int    incr = state->maxseg;
```

```
    if (cw > state->CongestionThreshold)

        incr = incr * incr / cw;

    state->CongestionWindow = MIN(cw + incr, TCP_MAXWIN);

}
```

onde `state` representa o estado de uma conexão TCP específica e define um limite superior sobre o tamanho que a janela de congestionamento pode atingir.

A Figura 162 mostra como o TCP `CongestionWindow` aumenta e diminui ao longo do tempo e serve para ilustrar a interação entre o início lento e o aumento aditivo/diminuição multiplicativa. Este rastreamento foi obtido de uma conexão TCP real e mostra o valor atual da `CongestionWindow`—linha colorida— ao longo do tempo.

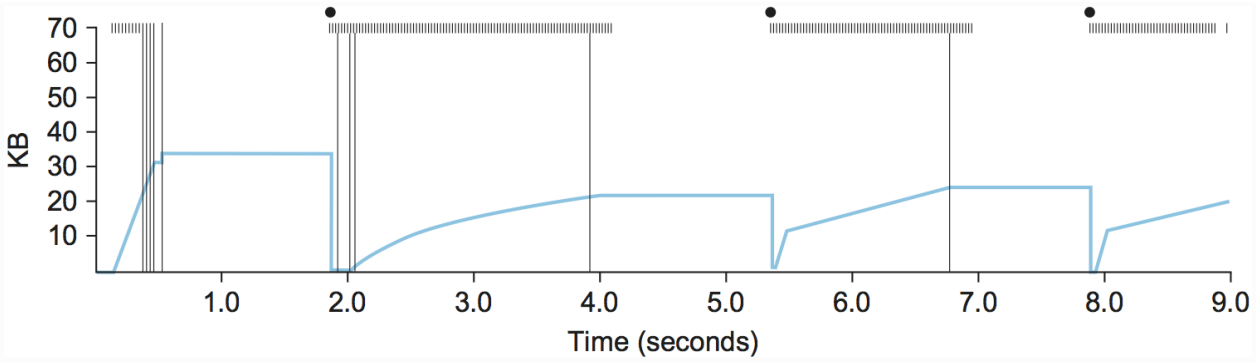

Figura 162. *Comportamento do controle de congestionamento TCP. Linha colorida = valor de CongestionWindow ao longo do tempo; marcadores sólidos no topo do gráfico = tempos limite; marcas de hash no topo do gráfico = horário em que cada pacote é transmitido; barras verticais = horário em que um pacote que foi eventualmente retransmitido foi transmitido pela primeira vez.*

Há vários pontos a serem observados sobre esse rastreamento. O primeiro é o rápido aumento na janela de congestionamento no início da conexão. Isso corresponde à fase inicial de início lento. A fase de início lento continua até que vários pacotes sejam perdidos em cerca de 0,4 segundos de conexão, tempo em que `CongestionWindow`se estabiliza em cerca de 34 KB. (O motivo pelo qual tantos pacotes são perdidos durante o início lento é discutido abaixo.) O motivo pelo qual a janela de congestionamento se estabiliza é que não há ACKs chegando, devido ao fato de que vários pacotes foram perdidos. De fato, nenhum novo pacote é enviado durante esse tempo, como denotado pela ausência de marcas de hash na parte superior do gráfico. Um tempo limite eventualmente ocorre em aproximadamente 2 segundos, momento em que a janela de congestionamento é dividida por 2 (ou seja, reduzida de aproximadamente 34 KB para cerca de 17 KB) e `CongestionThreshold`é definida para esse valor. O início lento então faz com `CongestionWindow`que seja redefinido para um pacote e comece a aumentar a partir daí.

Não há detalhes suficientes no rastreamento para ver exatamente o que acontece quando alguns pacotes são perdidos logo após 2 segundos, então avançamos para o aumento linear na janela de congestionamento que ocorre entre 2 e 4 segundos. Isso corresponde a um aumento aditivo. Em cerca de 4 segundos, o `CongestionWindow`aumento se estabiliza, novamente devido à perda de um pacote. Agora, em cerca de 5,5 segundos:

1. Ocorre um tempo limite, fazendo com que a janela de congestionamento seja dividida por 2, caindo de aproximadamente 22 KB para 11 KB, e `CongestionThreshold`é definida para esse valor.
2. `CongestionWindow`é redefinido para um pacote, quando o remetente entra em início lento.

3. O início lento faz com que `CongestionWindow`cresça exponencialmente até atingir `CongestionThreshold`.
4. `CongestionWindow`então cresce linearmente.

O mesmo padrão se repete por volta dos 8 segundos, quando ocorre outro tempo limite.

Voltamos agora à questão de por que tantos pacotes são perdidos durante o período inicial de inicialização lenta. Nesse ponto, o TCP está tentando descobrir quanta largura de banda está disponível na rede. Essa é uma tarefa difícil. Se a fonte não for agressiva nesse estágio — por exemplo, se ela apenas aumentar a janela de congestionamento linearmente —, levará muito tempo para descobrir quanta largura de banda está disponível. Isso pode ter um impacto drástico na taxa de transferência alcançada para essa conexão. Por outro lado, se a fonte for agressiva nesse estágio, como o TCP é durante o crescimento exponencial, a fonte corre o risco de ter metade da janela de pacotes descartada pela rede.

Para ver o que pode acontecer durante o crescimento exponencial, considere a situação em que a origem conseguiu enviar com sucesso 16 pacotes pela rede, fazendo com que sua janela de congestionamento dobrasse para 32. Suponha, no entanto, que a rede tenha capacidade suficiente para suportar 16 pacotes dessa origem. O resultado provável é que 16 dos 32 pacotes enviados sob a nova janela de congestionamento serão descartados pela rede; na verdade, este é o pior cenário, já que alguns dos pacotes serão armazenados em buffer em algum roteador. Esse problema se tornará cada vez mais grave à medida que o produto atraso x largura de banda das redes aumenta. Por exemplo, um produto atraso x largura de banda de 500 KB significa que cada conexão tem o potencial de perder até 500 KB de dados no início de cada conexão. Obviamente, isso pressupõe que tanto a origem quanto o destino implementem a extensão "janelas grandes".

Alternativas ao início lento, em que a fonte tenta estimar a largura de banda disponível por meios mais sofisticados, também foram exploradas. Um exemplo é chamado de

início rápido . A ideia básica é que um remetente TCP pode solicitar uma taxa de envio inicial maior do que o início lento permitiria, colocando uma taxa solicitada em seu pacote SYN como uma opção IP. Os roteadores ao longo do caminho podem examinar a opção, avaliar o nível atual de congestionamento no link de saída para esse fluxo e decidir se essa taxa é aceitável, se uma taxa menor seria aceitável ou se o início lento padrão deve ser usado. Quando o SYN chegar ao receptor, ele conterá uma taxa que foi aceitável para todos os roteadores no caminho ou uma indicação de que um ou mais roteadores no caminho não puderam suportar a solicitação de início rápido. No primeiro caso, o remetente TCP usa essa taxa para iniciar a transmissão; no último caso, ele retorna ao início lento padrão. Se o TCP puder iniciar o envio a uma taxa mais alta, uma sessão poderá atingir mais rapidamente o ponto de preenchimento do canal, em vez de levar muitos tempos de ida e volta para isso.

Claramente, um dos desafios desse tipo de aprimoramento do TCP é que ele exige uma cooperação substancialmente maior dos roteadores do que o TCP padrão. Se um único roteador no caminho não suportar inicialização rápida, o sistema retorna à inicialização lenta padrão. Portanto, pode levar muito tempo até que esses tipos de aprimoramentos cheguem à internet; por enquanto, é mais provável que sejam usados em ambientes de rede controlados (por exemplo, redes de pesquisa).

6.3.3 Retransmissão rápida e recuperação rápida

Os mecanismos descritos até agora faziam parte da proposta original de adicionar controle de congestionamento ao TCP. No entanto, logo se descobriu que a implementação granular dos timeouts do TCP resultava em longos períodos de inatividade da conexão enquanto aguardava a expiração de um timer. Por esse motivo, um novo mecanismo chamado *retransmissão rápida* foi adicionado ao TCP. A retransmissão rápida é uma heurística que, às vezes, aciona a retransmissão de um pacote descartado antes do mecanismo de timeout regular. O mecanismo de retransmissão rápida não substitui os timeouts regulares; apenas aprimora essa funcionalidade.

A ideia de retransmissão rápida é simples. Toda vez que um pacote de dados chega ao lado receptor, o receptor responde com uma confirmação, mesmo que esse número de sequência já tenha sido confirmado. Assim, quando um pacote chega fora de ordem — quando o TCP ainda não consegue confirmar os dados que o pacote contém porque os dados anteriores ainda não chegaram — o TCP reenvia a mesma confirmação que enviou da última vez. Essa segunda transmissão da mesma confirmação é chamada de *ACK duplicado* . Quando o lado remetente vê um ACK duplicado, ele sabe que o outro lado deve ter recebido um pacote fora de ordem, o que sugere que um pacote anterior pode ter sido perdido. Como também é possível que o pacote anterior tenha sido apenas atrasado em vez de perdido, o remetente espera até ver um certo número de ACKs duplicados e então retransmite o pacote perdido. Na prática, o TCP espera até ver três ACKs duplicados antes de retransmitir o pacote.

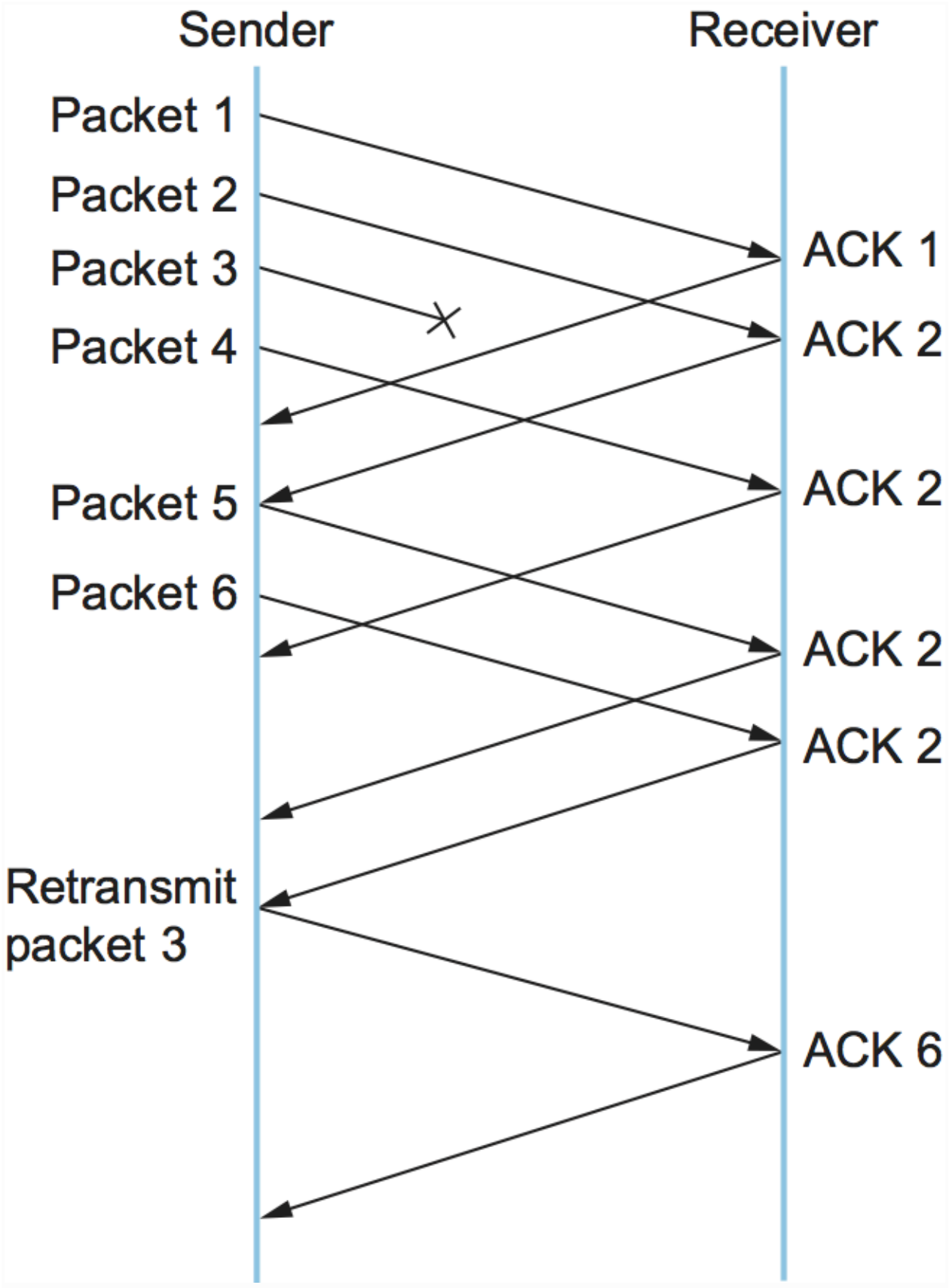
Sender
Receiver
Packet 1
Packet 2
Packet 3
Packet 4
Packet 5
Packet 6
ACK 1
ACK 2
ACK 2
ACK 2
ACK 2
Retransmit
packet 3
ACK 6

Figura 163. *Retransmissão rápida baseada em ACKs duplicados.*

A Figura 163 ilustra como ACKs duplicados levam a uma retransmissão rápida. Neste exemplo, o destino recebe os pacotes 1 e 2, mas o pacote 3 é perdido na rede. Assim, o destino enviará um ACK duplicado para o pacote 2 quando o pacote 4 chegar, novamente quando o pacote 5 chegar e assim por diante. (Para simplificar este exemplo, pensamos em termos dos pacotes 1, 2, 3 e assim por diante, em vez de nos preocuparmos com os números de sequência de cada byte.) Quando o remetente vê o terceiro ACK duplicado para o pacote 2 — aquele enviado porque o receptor recebeu o pacote 6 — ele retransmite o pacote 3. Observe que, quando a cópia retransmitida do pacote 3 chega ao destino, o receptor envia um ACK cumulativo para tudo, até e incluindo o pacote 6, de volta para a origem.

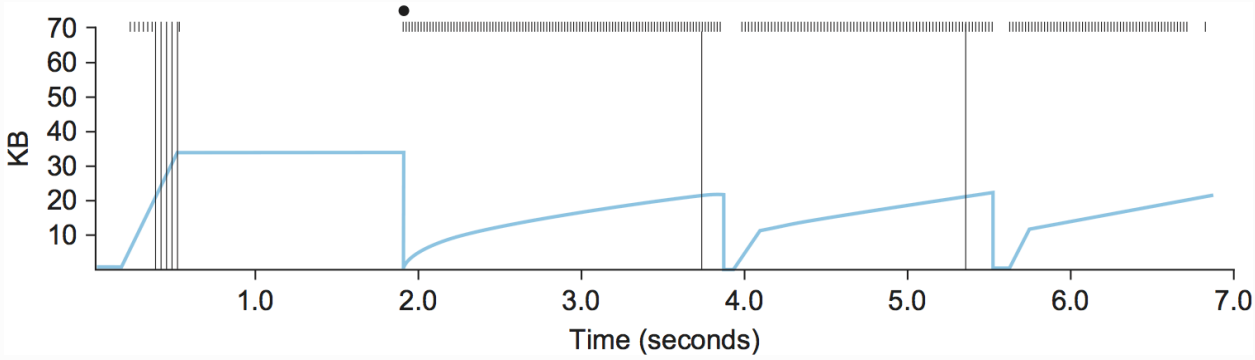

Figura 164. *Rastreamento de TCP com retransmissão rápida. Linha colorida = CongestionWindow; marcador sólido = tempo limite; marcas de hash = horário em que cada pacote é transmitido; barras verticais = horário em que um pacote que foi eventualmente retransmitido foi transmitido pela primeira vez.*

A Figura 164 ilustra o comportamento de uma versão do TCP com o mecanismo de retransmissão rápida. É interessante comparar este rastreamento com o apresentado na Figura 162 , onde a retransmissão rápida não foi implementada — os longos períodos durante os quais a janela de congestionamento permanece plana e nenhum pacote é enviado foram eliminados. Em geral, essa técnica é capaz de eliminar cerca de metade dos timeouts de granularidade grossa em uma conexão TCP típica, resultando em uma melhoria de aproximadamente 20% na taxa de transferência em

relação ao que poderia ter sido alcançado de outra forma. Observe, no entanto, que a estratégia de retransmissão rápida não elimina todos os timeouts de granularidade grossa. Isso ocorre porque, para um tamanho de janela pequeno, não haverá pacotes em trânsito suficientes para causar a entrega de ACKs duplicados suficientes. Dado um número suficiente de pacotes perdidos — por exemplo, como acontece durante a fase inicial de inicialização lenta — o algoritmo de janela deslizante eventualmente bloqueia o remetente até que ocorra um timeout. Na prática, o mecanismo de retransmissão rápida do TCP pode detectar até três pacotes descartados por janela.

Finalmente, há uma última melhoria que podemos fazer. Quando o mecanismo de retransmissão rápida sinaliza congestionamento, em vez de reduzir a janela de congestionamento para um pacote e executar o início lento, é possível usar os ACKs que ainda estão no pipe para cronometrar o envio de pacotes. Esse mecanismo, chamado de *recuperação rápida* , remove efetivamente a fase de início lento que ocorre entre o momento em que a retransmissão rápida detecta um pacote perdido e o início do aumento aditivo. Por exemplo, a recuperação rápida evita o período de início lento entre 3,8 e 4 segundos na Figura 164 e, em vez disso, simplesmente corta a janela de congestionamento pela metade (de 22 KB para 11 KB) e retoma o aumento aditivo. Em outras palavras, o início lento é usado apenas no início de uma conexão e sempre que ocorre um tempo limite de granularidade grossa. Em todos os outros momentos, a janela de congestionamento segue um padrão puro de aumento aditivo/diminuição multiplicativa.

6.3.4 TCP CÚBICO

Uma variante do algoritmo TCP padrão recém-descrito, chamada CUBIC, é o algoritmo de controle de congestionamento padrão distribuído com o Linux. O objetivo principal do CUBIC é oferecer suporte a redes com grandes produtos de atraso × largura de banda, às vezes chamadas de *redes long-fat* . Essas redes sofrem com o fato de o algoritmo TCP original exigir muitas viagens de ida e volta para atingir a capacidade disponível do caminho de ponta a ponta. O CUBIC faz isso sendo mais agressivo na

forma como aumenta o tamanho da janela, mas, claro, o segredo é ser mais agressivo sem ser tão agressivo a ponto de afetar negativamente outros fluxos.

Um aspecto importante da abordagem do CUBIC é ajustar sua janela de congestionamento em intervalos regulares, com base no tempo decorrido desde o último evento de congestionamento (por exemplo, a chegada de um ACK duplicado), em vez de apenas quando os ACKs chegam (sendo este último uma função do RTT). Isso permite que o CUBIC se comporte de forma justa ao competir com fluxos de RTT curto, que terão ACKs chegando com mais frequência.

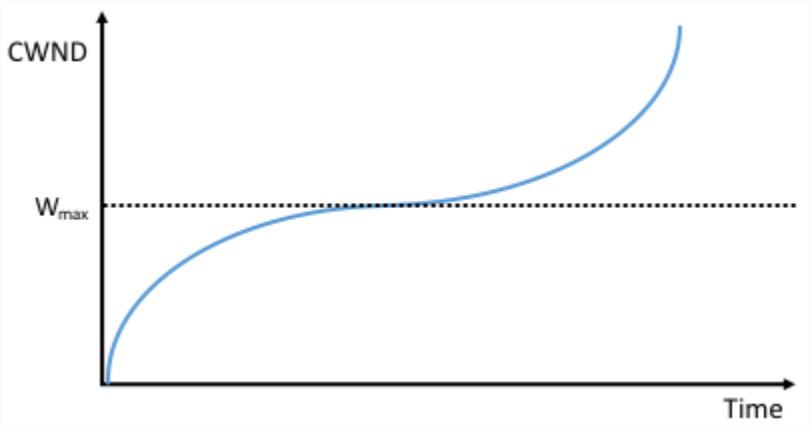

Figura 165. *Função cúbica genérica ilustrando a mudança na janela de congestionamento em função do tempo.*

O segundo aspecto importante do CUBIC é o uso de uma função cúbica para ajustar a janela de congestionamento. A ideia básica é mais fácil de entender observando a forma geral de uma função cúbica, que possui três fases: crescimento lento, platô achatado e crescimento crescente. Um exemplo genérico é mostrado na Figura 165 , que anotamos com uma informação extra: o tamanho máximo da janela de congestionamento atingido imediatamente antes do último evento de congestionamento como alvo (denotado

Wmax

). A ideia é começar rápido, mas diminuir a taxa de crescimento à medida que você se aproxima

$$W_{max}$$

, seja cauteloso e tenha crescimento próximo de zero quando estiver perto de

$$W_{max}$$

, e então aumente a taxa de crescimento à medida que você se afasta

$$W_{max}$$

. A última fase consiste essencialmente em sondar um novo objetivo alcançável

$$W_{max}$$

.

Especificamente, o CUBIC calcula a janela de congestionamento como uma função do tempo (t) desde o último evento de congestionamento

$$CWND(t) = C \times (t - K)3 + W_{max}$$

onde

$$K = W_{max} \times (1 - \beta)/C3$$

C é uma constante de escala e

$$\beta$$

é o fator de diminuição multiplicativo. O CUBIC define este último como 0,7 em vez do 0,5 usado pelo TCP padrão. Observando a Figura 165 , o CUBIC é frequentemente descrito como uma mudança de uma função côncava para convexa (enquanto a função aditiva do TCP padrão é apenas convexa).

6.4 Controle Avançado de Congestionamento

Esta seção explora o controle de congestionamento mais profundamente. Ao fazê-lo, é importante entender que a estratégia padrão do TCP é controlar o congestionamento assim que ele acontece, em vez de tentar evitá-lo em primeiro lugar. De fato, o TCP aumenta repetidamente a carga que impõe à rede em um esforço para encontrar o ponto em que o congestionamento ocorre e, em seguida, recua a partir desse ponto. Dito de outra forma, o TCP *precisa* criar perdas para encontrar a largura de banda disponível da conexão. Uma alternativa atraente é prever quando o congestionamento está prestes a acontecer e, em seguida, reduzir a taxa na qual os hosts enviam dados pouco antes dos pacotes começarem a ser descartados. Chamamos essa estratégia *de prevenção de congestionamento* para distingui-la do *controle de congestionamento* , mas provavelmente é mais preciso pensar em "prevenção" como um subconjunto de "controle".

Descrevemos duas abordagens diferentes para evitar congestionamentos. A primeira adiciona uma pequena funcionalidade adicional ao roteador para auxiliar o nó final na antecipação de congestionamentos. Essa abordagem é frequentemente chamada de *Gerenciamento Ativo de Filas* (AQM). A segunda abordagem tenta evitar congestionamentos exclusivamente dos hosts finais. Essa abordagem é implementada no TCP, tornando-a uma variante dos mecanismos de controle de congestionamento descritos na seção anterior.

6.4.1 Gerenciamento de filas ativas (DECbit, RED, ECN)

A primeira abordagem requer mudanças nos roteadores, o que nunca foi a forma preferida da internet para introduzir novos recursos, mas, ainda assim, tem sido uma fonte constante de consternação nos últimos 20 anos. O problema é que, embora seja geralmente aceito que os roteadores estão em uma posição ideal para detectar o início do congestionamento — ou seja, quando suas filas começam a ficar cheias —, não há consenso sobre qual é exatamente o melhor algoritmo. A seguir, descrevemos dois dos mecanismos clássicos e concluímos com uma breve discussão sobre a situação atual.

DECbit

O primeiro mecanismo foi desenvolvido para uso na Arquitetura de Rede Digital (DNA), uma rede sem conexão com um protocolo de transporte orientado à conexão. Esse mecanismo poderia, portanto, também ser aplicado ao TCP e ao IP. Como observado acima, a ideia aqui é dividir mais uniformemente a responsabilidade pelo controle de congestionamento entre os roteadores e os nós finais. Cada roteador monitora a carga que está enfrentando e notifica explicitamente os nós finais quando o congestionamento está prestes a ocorrer. Essa notificação é implementada definindo um bit de congestionamento binário nos pacotes que fluem pelo roteador, daí o nome *DECbit* . O host de destino então copia esse bit de congestionamento para o ACK que envia de volta para a origem. Finalmente, a origem ajusta sua taxa de envio para evitar congestionamento. A discussão a seguir descreve o algoritmo em mais detalhes, começando com o que acontece no roteador.

Um único bit de congestionamento é adicionado ao cabeçalho do pacote. Um roteador define esse bit em um pacote se o comprimento médio da fila for maior ou igual a 1 no momento em que o pacote chega. Esse comprimento médio da fila é medido em um intervalo de tempo que abrange o último ciclo ocupado+ocioso, mais o ciclo ocupado atual. (O roteador está *ocupado* quando está transmitindo e *ocioso* quando não está.) A Figura 166 mostra o comprimento da fila em um roteador em função do tempo.

Essencialmente, o roteador calcula a área sob a curva e divide esse valor pelo intervalo de tempo para calcular o comprimento médio da fila. Usar um comprimento de fila de 1 como gatilho para definir o bit de congestionamento é uma compensação entre enfileiramento significativo (e, portanto, maior taxa de transferência) e maior tempo ocioso (e, portanto, menor atraso). Em outras palavras, um comprimento de fila de 1 parece otimizar a função de potência.

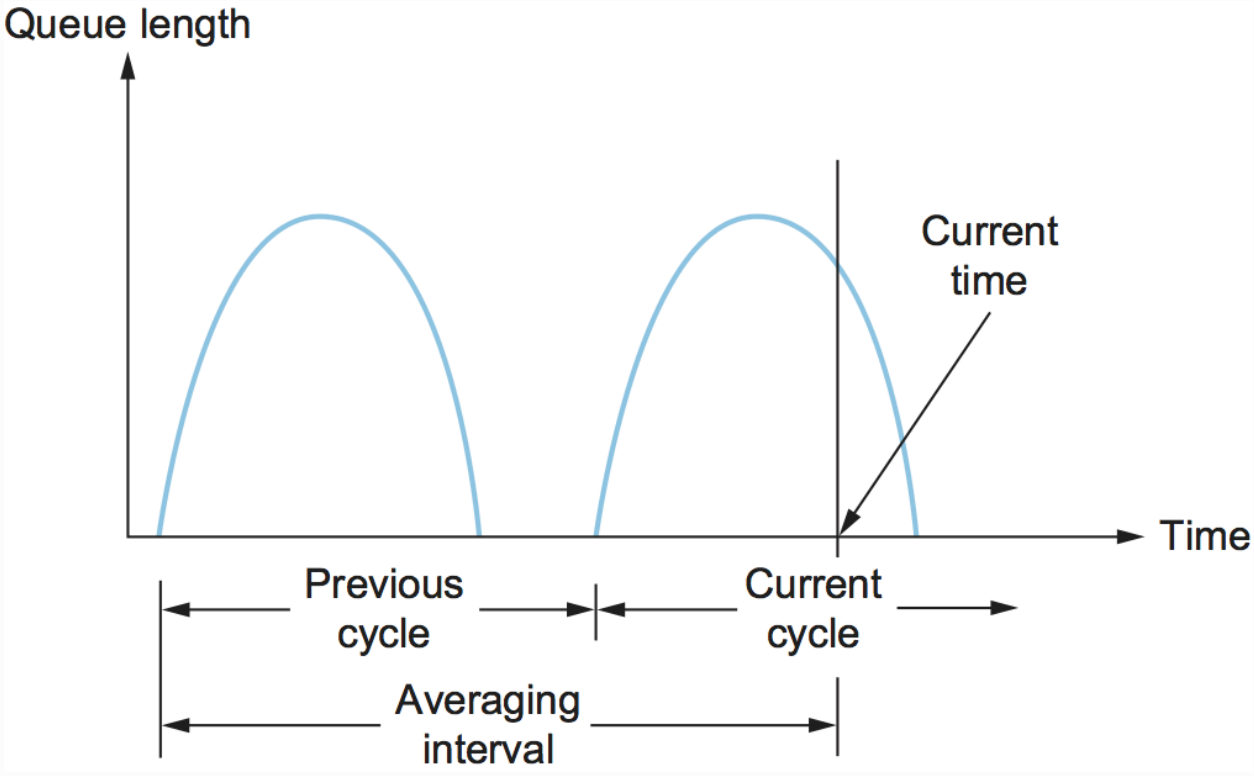

Figura 166. *Cálculo do comprimento médio da fila em um roteador.*

Voltando agora nossa atenção para a metade do mecanismo referente ao host, a origem registra quantos de seus pacotes resultaram na configuração do bit de congestionamento por algum roteador. Em particular, a origem mantém uma janela de congestionamento, assim como no TCP, e observa qual fração dos pacotes da última janela resultou na configuração do bit. Se menos de 50% dos pacotes tiveram o bit definido, a origem aumenta sua janela de congestionamento em um pacote. Se 50% ou mais dos pacotes da última janela tiveram o bit de congestionamento definido, a origem

diminui sua janela de congestionamento para 0,875 vezes o valor anterior. O valor de 50% foi escolhido como limite com base na análise que mostrou que ele corresponde ao pico da curva de potência. A regra "aumentar em 1, diminuir em 0,875" foi selecionada porque o aumento aditivo/diminuição multiplicativa torna o mecanismo estável.

Detecção precoce aleatória

Um segundo mecanismo, chamado *detecção precoce aleatória* (RED), é semelhante ao esquema DECbit, no sentido de que cada roteador é programado para monitorar seu próprio comprimento de fila e, quando detecta congestionamento iminente, notificar a fonte para ajustar sua janela de congestionamento. O RED, inventado por Sally Floyd e Van Jacobson no início da década de 1990, difere do esquema DECbit em dois aspectos principais.

A primeira é que, em vez de enviar explicitamente uma mensagem de notificação de congestionamento para a fonte, o RED é mais comumente implementado de forma a notificar *implicitamente* a fonte sobre o congestionamento, descartando um de seus pacotes. A fonte é, portanto, efetivamente notificada pelo tempo limite subsequente ou ACK duplicado. Caso você ainda não tenha adivinhado, o RED foi projetado para ser usado em conjunto com o TCP, que atualmente detecta congestionamento por meio de tempos limite (ou algum outro meio de detectar perda de pacotes, como ACKs duplicados). Como a parte "early" da sigla RED sugere, o gateway descarta o pacote mais cedo do que deveria, para notificar a fonte de que ela deve diminuir sua janela de congestionamento mais cedo do que normalmente faria. Em outras palavras, o roteador descarta alguns pacotes antes de esgotar completamente seu espaço de buffer, de modo a fazer com que a fonte fique mais lenta, na esperança de que isso signifique que ela não precise descartar muitos pacotes posteriormente.

A segunda diferença entre RED e DECbit está nos detalhes de como o RED decide quando descartar um pacote e qual pacote ele decide descartar. Para entender a ideia básica, considere uma fila FIFO simples. Em vez de esperar que a fila fique

completamente cheia e então ser forçado a descartar cada pacote que chega (a política de descarte de cauda da seção anterior), poderíamos decidir descartar cada pacote que chega com alguma *probabilidade de descarte* sempre que o comprimento da fila exceder algum *nível de descarte* . Essa ideia é chamada de *descarte aleatório antecipado* . O algoritmo RED define os detalhes de como monitorar o comprimento da fila e quando descartar um pacote.

Nos parágrafos seguintes, descrevemos o algoritmo RED conforme proposto originalmente por Floyd e Jacobson. Observamos que diversas modificações foram propostas desde então, tanto pelos inventores quanto por outros pesquisadores. No entanto, as ideias-chave são as mesmas apresentadas a seguir, e a maioria das implementações atuais se aproxima do algoritmo a seguir.

Primeiro, o RED calcula o comprimento médio da fila usando uma média ponderada de execução semelhante à usada no cálculo original do tempo limite do TCP. Ou seja, `AvgLen`é calculado como

```
AvgLen = (1 - Weight) x AvgLen + Weight x SampleLen
```

onde 0 < `Weight`< 1 e `SampleLen`é o comprimento da fila quando uma medição de amostra é realizada. Na maioria das implementações de software, o comprimento da fila é medido sempre que um novo pacote chega ao gateway. Em hardware, ele pode ser calculado em algum intervalo de amostragem fixo.

A razão para usar um comprimento médio de fila em vez de um instantâneo é que ele captura com mais precisão a noção de congestionamento. Devido à natureza intermitente do tráfego da Internet, as filas podem ficar cheias muito rapidamente e depois ficar vazias novamente. Se uma fila está passando a maior parte do tempo vazia, então provavelmente não é apropriado concluir que o roteador está congestionado e dizer aos hosts para desacelerarem. Assim, o cálculo da média

ponderada de execução tenta detectar congestionamentos de longa duração, conforme indicado na parte direita da Figura 167 , filtrando mudanças de curto prazo no comprimento da fila. Você pode pensar na média de execução como um filtro passa-baixa, onde `Weight`determina a constante de tempo do filtro. A questão de como escolhemos essa constante de tempo é discutida abaixo.

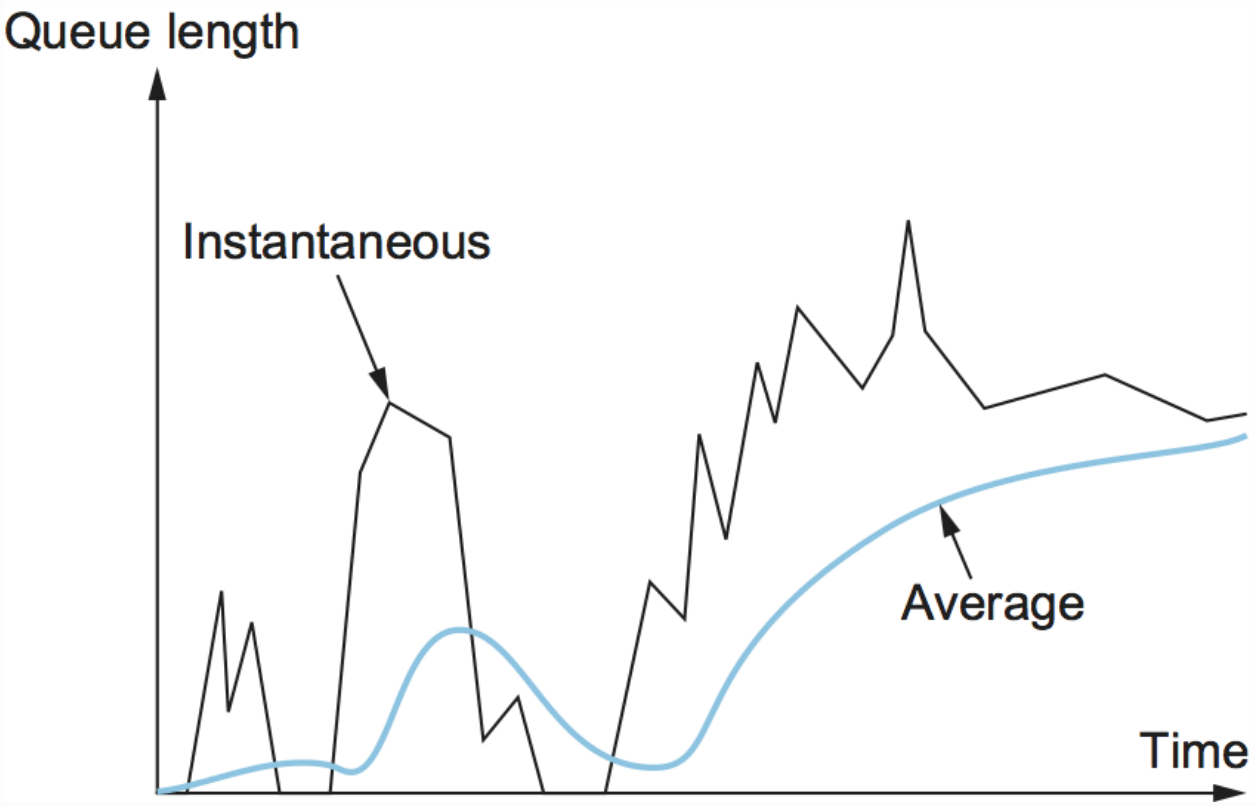

Figura 167. *Comprimento médio ponderado da fila.*

Em segundo lugar, o RED possui dois limites de comprimento de fila que acionam determinada atividade: `MinThreshold`e `MaxThreshold`. Quando um pacote chega ao gateway, o RED compara o atual `AvgLen`com esses dois limites, de acordo com as seguintes regras:

```
if AvgLen <= MinThreshold
```

```
    queue the packet

if MinThreshold < AvgLen < MaxThreshold

    calculate probability P

    drop the arriving packet with probability P

if MaxThreshold <= AvgLen

    drop the arriving packet
```

Se o comprimento médio da fila for menor que o limite inferior, nenhuma ação será tomada, e se o comprimento médio da fila for maior que o limite superior, então o pacote será sempre descartado. Se o comprimento médio da fila estiver entre os dois limites, então o pacote recém-chegado será descartado com alguma probabilidade `P`. Esta situação é ilustrada na Figura 168. A relação aproximada entre `P`e `AvgLen`é mostrada na Figura 169. Observe que a probabilidade de descarte aumenta lentamente quando `AvgLen`está entre os dois limites, atingindo `MaxP`o limite superior, ponto em que salta para a unidade. A lógica por trás disso é que, se `AvgLen`atingir o limite superior, então a abordagem suave (descartar alguns pacotes) não está funcionando e medidas drásticas são necessárias: descartar todos os pacotes que chegam. Algumas pesquisas sugeriram que uma transição mais suave da eliminação aleatória para a eliminação completa, em vez da abordagem descontínua mostrada aqui, pode ser apropriada.

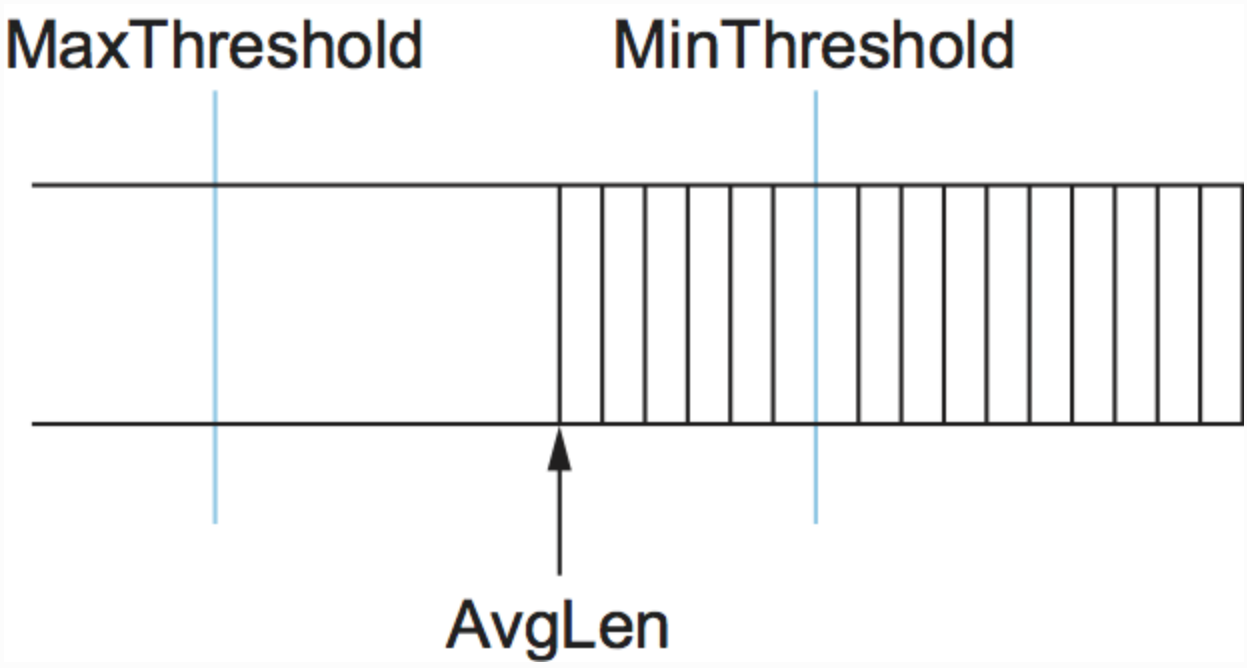

Figura 168. *Limites RED em uma fila FIFO.*

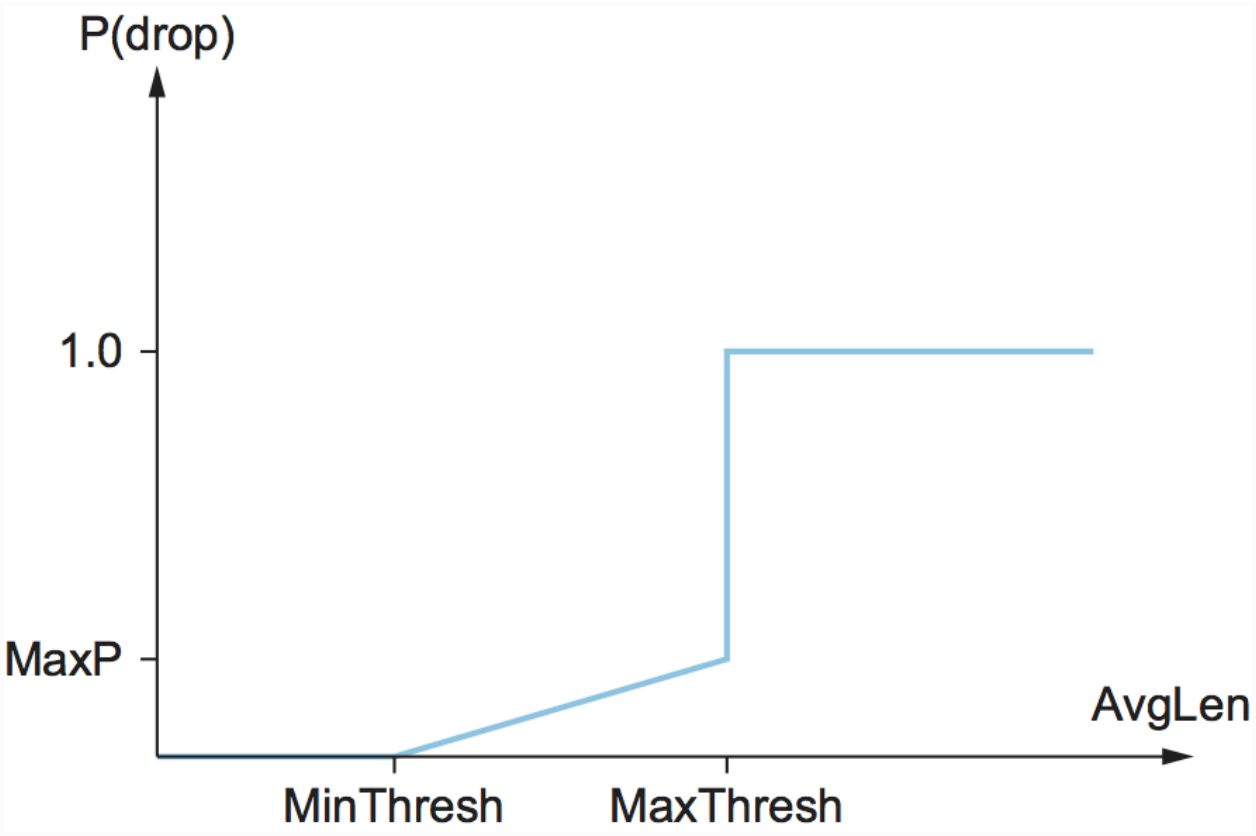

Figura 169. *Função de probabilidade de queda para RED.*

Embora a Figura 169 mostre a probabilidade de queda como uma função apenas de `AvgLen`, a situação é, na verdade, um pouco mais complicada. Na verdade, `P`é uma função de ambos `AvgLen`e de quanto tempo se passou desde que o último pacote foi descartado. Especificamente, é calculado da seguinte forma:

```
TempP = MaxP x (AvgLen - MinThreshold) / (MaxThreshold - MinThreshold)
```

```
P = TempP/(1 - count x TempP)
```

`TempP`é a variável plotada no eixo y na Figura 169 , `count`mantém o controle de quantos pacotes recém-chegados foram enfileirados (não descartados) e `AvgLen`estiveram entre os dois limites. `P`aumenta lentamente à medida que `count`aumenta, tornando uma queda cada vez mais provável à medida que o tempo desde a última queda aumenta. Isso torna quedas espaçadas próximas relativamente menos prováveis do que quedas amplamente espaçadas. Essa etapa extra no cálculo `P`foi introduzida pelos inventores do RED quando observaram que, sem ela, as quedas de pacotes não eram bem distribuídas no tempo, mas tendiam a ocorrer em clusters. Como as chegadas de pacotes de uma determinada conexão provavelmente chegam em rajadas, esse agrupamento de quedas provavelmente causará várias quedas em uma única conexão. Isso não é desejável, pois apenas uma queda por tempo de ida e volta é suficiente para fazer com que uma conexão reduza seu tamanho de janela, enquanto várias quedas podem enviá-la de volta ao início lento.

Por exemplo, suponha que definimos `MaxP`como 0,02 e `count`é inicializado como zero. Se o comprimento médio da fila estivesse na metade entre os dois limites, então `TempP`, e o valor inicial de `P`, seria metade de `MaxP`, ou 0,01. Um pacote que chega, é claro, tem uma chance de 99 em 100 de entrar na fila neste ponto. Com cada pacote sucessivo

que não é descartado, `P` aumenta lentamente e, quando 50 pacotes chegam sem uma queda, `P` teria dobrado para 0,02. No improvável evento de que 99 pacotes cheguem sem perda, `P` chega a 1, garantindo que o próximo pacote seja descartado. O importante sobre esta parte do algoritmo é que ele garante uma distribuição aproximadamente uniforme de quedas ao longo do tempo.

A intenção é que, se o RED descartar uma pequena porcentagem de pacotes quando `AvgLen` exceder `MinThreshold`, isso fará com que algumas conexões TCP reduzam seus tamanhos de janela, o que, por sua vez, reduzirá a taxa de chegada de pacotes ao roteador. Se tudo correr bem, a taxa de chegada de pacotes `AvgLen` diminuirá e o congestionamento será evitado. O comprimento da fila pode ser mantido curto, enquanto a taxa de transferência permanece alta, já que poucos pacotes são descartados.

Observe que, como o RED opera com um comprimento de fila médio ao longo do tempo, é possível que o comprimento instantâneo da fila seja muito maior que `AvgLen`. Nesse caso, se um pacote chegar e não houver onde colocá-lo, ele terá que ser descartado. Quando isso acontece, o RED opera no modo de descarte de cauda. Um dos objetivos do RED é evitar o comportamento de descarte de cauda, se possível.

A natureza aleatória do RED confere uma propriedade interessante ao algoritmo. Como o RED descarta pacotes aleatoriamente, a probabilidade de o RED decidir descartar o(s) pacote(s) de um fluxo específico é aproximadamente proporcional à parcela da largura de banda que esse fluxo está recebendo naquele roteador. Isso ocorre porque um fluxo que envia um número relativamente grande de pacotes fornece mais candidatos para descarte aleatório. Portanto, há algum senso de alocação justa de recursos embutido no RED, embora não seja de forma alguma preciso. Embora indiscutivelmente justo, como o RED pune fluxos de alta largura de banda mais do que fluxos de baixa largura de banda, ele aumenta a probabilidade de uma reinicialização do TCP, o que é duplamente prejudicial para esses fluxos de alta largura de banda.

Conclusão principal

Observe que uma quantidade considerável de análises foi dedicada à definição dos vários parâmetros RED — por exemplo, `MaxThreshold`, `MinThreshold`e `MaxP`— `Weight`tudo em nome da otimização da função de potência (razão throughput-to-delay). O desempenho desses parâmetros também foi confirmado por simulação, e o algoritmo demonstrou não ser excessivamente sensível a eles. É importante ter em mente, no entanto, que toda essa análise e simulação dependem de uma caracterização específica da carga de trabalho da rede. A contribuição real do RED é um mecanismo pelo qual o roteador pode gerenciar com mais precisão o comprimento da sua fila. Definir precisamente o que constitui um comprimento de fila ideal depende da combinação de tráfego e ainda é objeto de pesquisa, com informações reais sendo coletadas a partir da implantação operacional do RED na Internet. [Próximo]

Considere a configuração dos dois limites, `MinThreshold`e `MaxThreshold`. Se o tráfego for bastante intermitente, então `MinThreshold` deve ser suficientemente grande para permitir que a utilização do link seja mantida em um nível aceitavelmente alto. Além disso, a diferença entre os dois limites deve ser maior do que o aumento típico no comprimento médio da fila calculado em um RTT. Definir `MaxThreshold`como duas vezes `MinThreshold`parece ser uma regra prática razoável, dada a combinação de tráfego na Internet atual. Além disso, como esperamos que o comprimento médio da fila fique entre os dois limites durante períodos de alta carga, deve haver espaço livre suficiente no buffer *acima* `MaxThreshold` para absorver os picos naturais que ocorrem no tráfego da Internet sem forçar o roteador a entrar no modo de queda de cauda.

Observamos acima que `Weight`determina a constante de tempo para o filtro passa-baixa de média móvel, e isso nos dá uma pista de como podemos escolher um valor adequado para ele. Lembre-se de que o RED está tentando enviar sinais para fluxos TCP descartando pacotes durante períodos de congestionamento. Suponha que um roteador descarta um pacote de alguma conexão TCP e, em seguida, encaminha imediatamente mais alguns pacotes da mesma conexão. Quando esses pacotes

chegam ao receptor, ele começa a enviar ACKs duplicados para o remetente. Quando o remetente vê ACKs duplicados suficientes, ele reduz o tamanho da janela. Portanto, do momento em que o roteador descarta um pacote até o momento em que o mesmo roteador começa a ver algum alívio da conexão afetada em termos de um tamanho de janela reduzido, pelo menos um tempo de ida e volta deve decorrer para essa conexão. Provavelmente não faz muito sentido fazer o roteador responder ao congestionamento em escalas de tempo muito menores do que o tempo de ida e volta das conexões que passam por ele. Como observado anteriormente, 100 ms não é uma estimativa ruim dos tempos médios de ida e volta na Internet. Portanto, `Weight`deve ser escolhido de forma que as alterações no comprimento da fila em escalas de tempo muito menores que 100 ms sejam filtradas.

Como o RED funciona enviando sinais aos fluxos TCP para que diminuam a velocidade, você pode se perguntar o que aconteceria se esses sinais fossem ignorados. Isso costuma ser chamado de problema *do fluxo não responsivo* . Fluxos não responsivos usam mais do que sua cota justa de recursos da rede e poderiam causar colapso congestivo se houvesse um número suficiente deles, assim como nos dias anteriores ao controle de congestionamento do TCP. Algumas das técnicas descritas na próxima seção podem ajudar a resolver esse problema, isolando certas classes de tráfego de outras. Há também a possibilidade de que uma variante do RED possa sofrer uma queda mais acentuada em fluxos que não respondem às dicas iniciais que ele envia.

Notificação explícita de congestionamento

O RED é o mecanismo de AQM mais estudado, mas não foi amplamente implementado, em parte porque não resulta em comportamento ideal em todas as circunstâncias. No entanto, é a referência para a compreensão do comportamento do AQM. Outro ponto positivo do RED é o reconhecimento de que o TCP poderia ter um desempenho melhor se os roteadores enviassem um sinal de congestionamento mais explícito.

Ou seja, em vez de *descartar* um pacote e presumir que o TCP eventualmente o notará (por exemplo, devido à chegada de um ACK duplicado), o RED (ou qualquer algoritmo AQM) pode fazer um trabalho melhor se, em vez disso, *marcar* o pacote e continuar a enviá-lo ao destino. Essa ideia foi codificada em alterações nos cabeçalhos IP e TCP conhecidas como *Notificação Explícita de Congestionamento* (ECN).

Especificamente, esse feedback é implementado tratando dois bits no `TOS`campo IP como bits ECN. Um bit é definido pela fonte para indicar que ela é compatível com ECN, ou seja, capaz de reagir a uma notificação de congestionamento. Isso é chamado de `ECT`bit (Transporte com Capacidade ECN). O outro bit é definido pelos roteadores ao longo do caminho de ponta a ponta quando ocorre congestionamento, conforme calculado pelo algoritmo AQM que ele está executando. Isso é chamado de `CE`bit (Congestionamento Encontrado).

Além desses dois bits no cabeçalho IP (que são independentes de transporte), o ECN também inclui a adição de dois sinalizadores opcionais ao cabeçalho TCP. O primeiro, `ECE`(ECN-Echo), comunica do receptor ao remetente que recebeu um pacote com o `CE`bit definido. O segundo, `CWR`(Congestion Window Reduced), comunica do remetente ao receptor que reduziu a janela de congestionamento.

Embora ECN seja atualmente a interpretação padrão de dois dos oito bits no `TOS`campo do cabeçalho IP e o suporte a ECN seja altamente recomendado, ele não é obrigatório. Além disso, não existe um único algoritmo AQM recomendado, mas sim uma lista de requisitos que um bom algoritmo AQM deve atender. Assim como os algoritmos de controle de congestionamento TCP, cada algoritmo AQM tem suas vantagens e desvantagens e, portanto, precisamos de muitos deles. Há um cenário específico, no entanto, em que o algoritmo de controle de congestionamento TCP e o algoritmo AQM são projetados para funcionar em conjunto: o data center. Retornaremos a este caso de uso no final desta seção.

6.4.2 Abordagens baseadas na fonte (Vegas, BBR, DCTCP)

Diferentemente dos esquemas anteriores de prevenção de congestionamento, que dependiam da cooperação de roteadores, agora descrevemos uma estratégia para detectar os estágios iniciais de congestionamento — antes que ocorram perdas — a partir dos hosts finais. Primeiro, apresentamos uma breve visão geral de um conjunto de mecanismos relacionados que utilizam diferentes informações para detectar os estágios iniciais de congestionamento e, em seguida, descrevemos dois mecanismos específicos com mais detalhes.

A ideia geral dessas técnicas é observar um sinal da rede de que a fila de algum roteador está se acumulando e que o congestionamento ocorrerá em breve se nada for feito a respeito. Por exemplo, a fonte pode notar que, à medida que as filas de pacotes se acumulam nos roteadores da rede, há um aumento mensurável no RTT para cada pacote sucessivo enviado. Um algoritmo em particular explora essa observação da seguinte maneira: a janela de congestionamento normalmente aumenta como no TCP, mas a cada dois atrasos de ida e volta, o algoritmo verifica se o RTT atual é maior que a média dos RTTs mínimo e máximo observados até o momento. Se for, o algoritmo diminui a janela de congestionamento em um oitavo.

Um segundo algoritmo faz algo semelhante. A decisão de alterar ou não o tamanho da janela atual é baseada nas alterações no RTT e no tamanho da janela. A janela é ajustada a cada dois atrasos de ida e volta com base no produto.

```
(CurrentWindow - OldWindow) x (CurrentRTT - OldRTT)
```

Se o resultado for positivo, a fonte diminui o tamanho da janela em um oitavo; se o resultado for negativo ou 0, a fonte aumenta a janela em um tamanho máximo de

pacote. Observe que a janela muda a cada ajuste; ou seja, ela oscila em torno de seu ponto ótimo.

Outra mudança observada à medida que a rede se aproxima do congestionamento é o achatamento da taxa de envio. Um terceiro esquema aproveita esse fato. A cada RTT, o algoritmo aumenta o tamanho da janela em um pacote e compara a taxa de transferência alcançada com a taxa de transferência quando a janela era um pacote menor. Se a diferença for menor que a metade da taxa de transferência alcançada quando apenas um pacote estava em trânsito — como era o caso no início da conexão — o algoritmo diminui a janela em um pacote. Esse esquema calcula a taxa de transferência dividindo o número de bytes restantes na rede pelo RTT.

TCP Vegas

O mecanismo que descreveremos em mais detalhes é semelhante ao algoritmo anterior, pois analisa mudanças na taxa de transferência ou, mais especificamente, mudanças na taxa de envio. No entanto, ele difere do algoritmo anterior na forma como calcula a taxa de transferência e, em vez de procurar uma mudança na inclinação da taxa de transferência, compara a taxa de transferência medida com uma taxa de transferência esperada. O algoritmo, TCP Vegas, não é amplamente utilizado na internet atualmente, mas a estratégia que ele utiliza foi adotada por outras implementações que estão sendo implementadas atualmente.

A intuição por trás do algoritmo de Vegas pode ser vista no rastreamento do TCP padrão dado na Figura 170. O gráfico superior mostrado na Figura 170 rastreia a janela de congestionamento da conexão; ele mostra as mesmas informações que os rastreamentos fornecidos anteriormente nesta seção. Os gráficos do meio e inferior mostram novas informações: o gráfico do meio mostra a taxa média de envio medida na origem, e o gráfico inferior mostra o comprimento médio da fila medido no roteador gargalo. Todos os três gráficos são sincronizados no tempo. No período entre 4,5 e 6,0 segundos (região sombreada), a janela de congestionamento aumenta (gráfico superior). Esperamos que a taxa de transferência observada também aumente, mas

em vez disso ela permanece estável (gráfico do meio). Isso ocorre porque a taxa de transferência não pode aumentar além da largura de banda disponível. Além desse ponto, qualquer aumento no tamanho da janela resulta apenas em pacotes ocupando espaço de buffer no roteador gargalo (gráfico inferior).

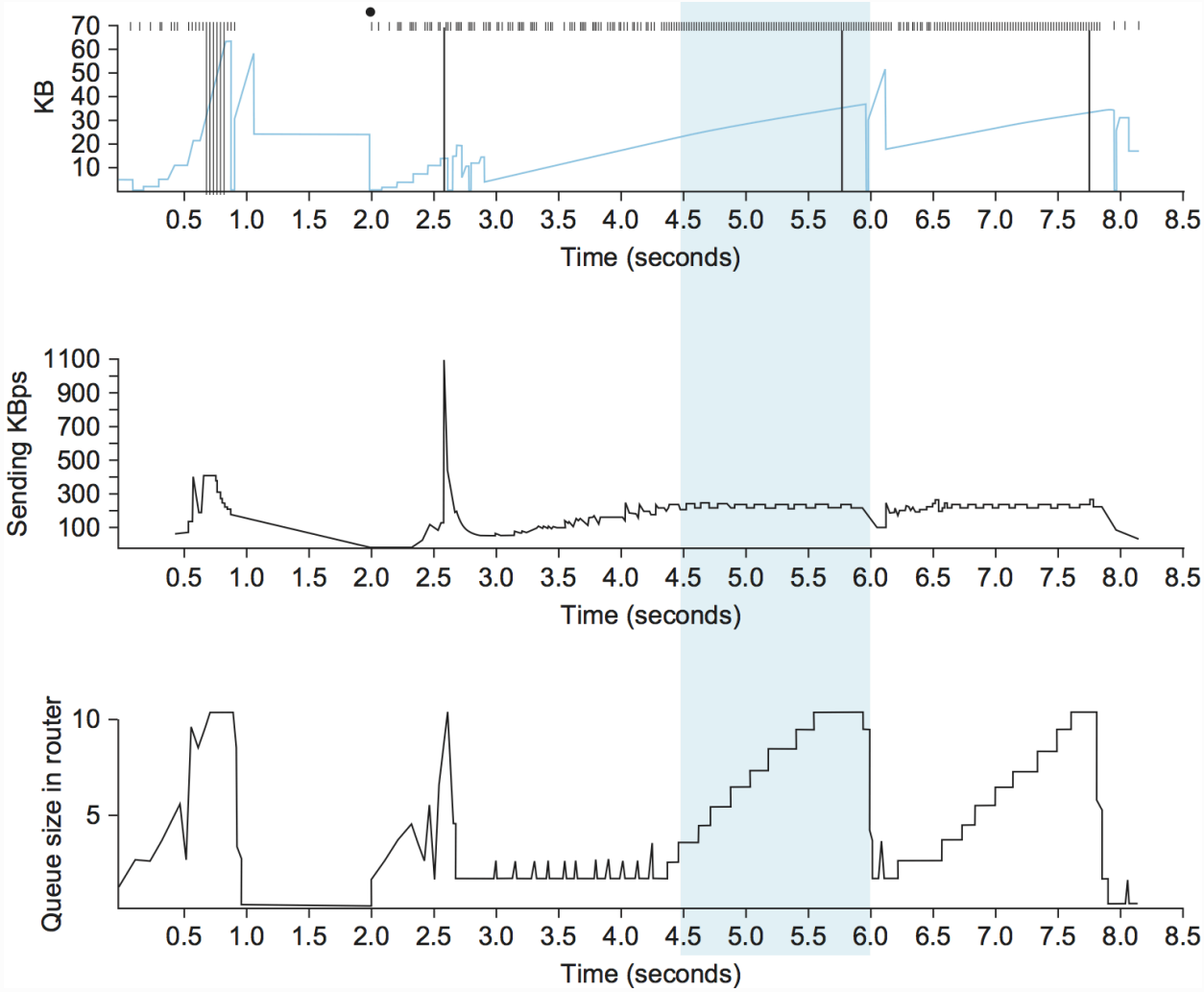

Figura 170. *Janela de congestionamento versus taxa de transferência observada (os três gráficos estão sincronizados). Acima, janela de congestionamento; no meio, taxa de transferência observada; abaixo, espaço de buffer ocupado no roteador. Linha colorida = CongestionWindow ; marcador sólido = tempo limite; marcas de hash = horário em que cada pacote é transmitido; barras verticais = horário em que um pacote que foi eventualmente retransmitido foi transmitido pela primeira vez.*

Uma metáfora útil que descreve o fenômeno ilustrado na Figura 170 é dirigir no gelo. O velocímetro (janela de congestionamento) pode indicar que você está a 48 km/h, mas ao olhar pela janela do carro e ver as pessoas passando por você a pé (taxa de envio medida), você sabe que não está a mais de 8 km/h. A energia extra está sendo absorvida pelos pneus do carro (tampões do roteador).

O TCP Vegas usa essa ideia para medir e controlar a quantidade de dados extras que essa conexão tem em trânsito, onde por "dados extras" entendemos dados que a fonte não teria transmitido se estivesse tentando corresponder exatamente à largura de banda disponível da rede. O objetivo do TCP Vegas é manter a quantidade "correta" de dados extras na rede. Obviamente, se uma fonte estiver enviando muitos dados extras, isso causará longos atrasos e possivelmente levará a congestionamento. Menos obviamente, se uma conexão estiver enviando poucos dados extras, ela não poderá responder com rapidez suficiente a aumentos transitórios na largura de banda da rede disponível. As ações de prevenção de congestionamento do TCP Vegas são baseadas em mudanças na quantidade estimada de dados extras na rede, não apenas em pacotes descartados. Agora descrevemos o algoritmo em detalhes.

Primeiro, defina um dado fluxo `BaseRTT`como o RTT de um pacote quando o fluxo não está congestionado. Na prática, o TCP Vegas define `BaseRTT`o menor tempo de ida e volta medido; geralmente, é o RTT do primeiro pacote enviado pela conexão, antes que as filas do roteador aumentem devido ao tráfego gerado por esse fluxo. Se assumirmos que não estamos sobrecarregando a conexão, a vazão esperada é dada por

```
ExpectedRate = CongestionWindow / BaseRTT
```

onde `CongestionWindow`está a janela de congestionamento TCP, que assumimos (para o propósito desta discussão) ser igual ao número de bytes em trânsito.

Em segundo lugar, o TCP Vegas calcula a taxa de envio atual, `ActualRate`. Isso é feito registrando o tempo de envio de um pacote distinto, registrando quantos bytes são transmitidos entre o momento em que o pacote é enviado e o momento em que sua confirmação é recebida, calculando o RTT da amostra para o pacote distinto quando sua confirmação chega e dividindo o número de bytes transmitidos pelo RTT da amostra. Esse cálculo é feito uma vez por tempo de ida e volta.

Terceiro, o TCP Vegas compara `ActualRate`e `ExpectedRate`ajusta a janela adequadamente. Deixamos . Observe que é positivo ou 0 por definição, pois implica que precisamos mudar para o RTT amostrado mais recente. Também definimos dois limites, $\alpha < \beta$, correspondendo aproximadamente a ter poucos e muitos dados extras na rede, respectivamente. Quando $< \alpha$, o TCP Vegas aumenta a janela de congestionamento linearmente durante o próximo RTT, e quando $> \beta$, o TCP Vegas diminui a janela de congestionamento linearmente durante o próximo RTT. O TCP Vegas deixa a janela de congestionamento inalterada quando $\alpha < < \beta$.`Diff = ExpectedRate - ActualRateDiffActualRate >ExpectedRateBaseRTTDiffDiffDiff`

Intuitivamente, podemos observar que quanto mais distante a taxa de transferência real da taxa de transferência esperada, maior o congestionamento na rede, o que implica que a taxa de envio deve ser reduzida. O limite β desencadeia essa redução. Por outro lado, quando a taxa de transferência real se aproxima muito da taxa de transferência esperada, a conexão corre o risco de não utilizar a largura de banda disponível. O limite α desencadeia esse aumento. O objetivo geral é manter entre α e β bytes extras na rede.

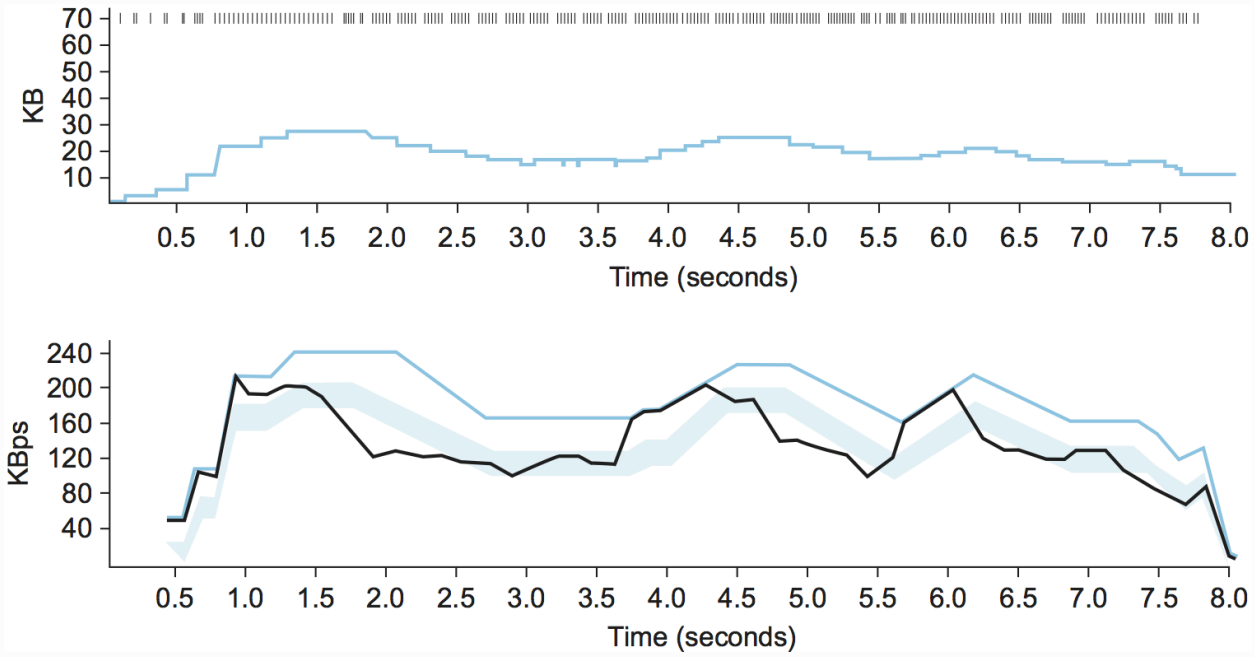

Figura 171. *Rastreamento do mecanismo de prevenção de congestionamento do TCP Vegas. Acima, janela de congestionamento; abaixo, vazão esperada (linha colorida) e real (linha preta). A área sombreada representa a região entre os limiares α e β .*

A Figura 171 traça o algoritmo de prevenção de congestionamento do TCP Vegas. O gráfico superior traça a janela de congestionamento, mostrando as mesmas informações dos outros traçados fornecidos ao longo deste capítulo. O gráfico inferior traça as taxas de transferência esperadas e reais que regem como a janela de congestionamento é definida. É este gráfico inferior que melhor ilustra como o algoritmo funciona. A linha colorida rastreia o`ExpectedRate`, enquanto a linha preta rastreia o`ActualRate`. A faixa sombreada larga fornece a região entre os *limites α* e *β* ; a parte superior da faixa sombreada está *a α* KBps de distância de`ExpectedRate`, e a parte inferior da faixa sombreada está a *β* KBps de distância de`ExpectedRate`. O objetivo é manter o `ActualRate`entre esses dois limites, dentro da região sombreada. Sempre que`ActualRate`cai abaixo da região sombreada (ou seja, fica muito longe de`ExpectedRate`), o TCP Vegas diminui a janela de congestionamento porque teme que muitos pacotes estejam sendo armazenados em buffer na rede. Da mesma forma, sempre que`ActualRate`fica acima da região sombreada (ou seja, fica muito perto

de`ExpectedRate`), o TCP Vegas aumenta a janela de congestionamento porque teme estar subutilizando a rede.

Como o algoritmo, como apresentado, compara a diferença entre as taxas de transferência reais e esperadas com os limiares *α* e *β , esses dois limiares são definidos em termos de KBps. No entanto, talvez seja mais preciso pensar em termos de quantos buffers* extras a conexão está ocupando na rede. Por exemplo, em uma conexão com uma taxa de `BaseRTT`transferência de 100 ms e um tamanho de pacote de 1 KB, se *α* = 30 KBps e *β* = 60 KBps, podemos pensar em *α* como especificando que a conexão precisa ocupar pelo menos 3 buffers extras na rede e *β* como especificando que a conexão não deve ocupar mais do que 6 buffers extras na rede. Na prática, uma configuração de *α* para 1 buffer e *β* para 3 buffers funciona bem.

Por fim, você notará que o TCP Vegas diminui a janela de congestionamento linearmente, aparentemente em conflito com a regra de que a redução multiplicativa é necessária para garantir a estabilidade. A explicação é que o TCP Vegas usa a redução multiplicativa quando ocorre um timeout; a redução linear que acabamos de descrever é uma redução *antecipada* na janela de congestionamento que deve ocorrer antes que o congestionamento ocorra e os pacotes comecem a ser descartados.

TCP BBR

BBR (Bottleneck Bandwidth and RTT) é um novo algoritmo de controle de congestionamento TCP desenvolvido por pesquisadores do Google. Assim como o Vegas, o BBR é baseado em atraso, o que significa que tenta detectar o crescimento do buffer para evitar congestionamento e perda de pacotes. Tanto o BBR quanto o Vegas utilizam o RTT mínimo e o RTT máximo, calculados ao longo de um intervalo de tempo, como seus principais sinais de controle.

O BBR também introduz novos mecanismos para melhorar o desempenho, incluindo ritmo de pacotes, sondagem de largura de banda e sondagem de RTT. O ritmo de pacotes espaça os pacotes com base na estimativa da largura de banda disponível.

Isso elimina rajadas e enfileiramento desnecessário, o que resulta em um sinal de feedback melhor. O BBR também aumenta periodicamente sua taxa, sondando assim a largura de banda disponível. Da mesma forma, o BBR diminui periodicamente sua taxa, sondando assim um novo RTT mínimo. O mecanismo de sondagem de RTT tenta ser autossincronizado, ou seja, quando há vários fluxos BBR, suas respectivas sondagens de RTT acontecem ao mesmo tempo. Isso fornece uma visão mais precisa do RTT do caminho descongestionado real, o que resolve um dos principais problemas com mecanismos de controle de congestionamento baseados em atraso: ter conhecimento preciso do RTT do caminho descongestionado.

O BBR está sendo trabalhado ativamente e evoluindo rapidamente. Um dos principais focos é a imparcialidade. Por exemplo, alguns experimentos mostram que fluxos CUBIC obtêm 100 vezes menos largura de banda ao competir com fluxos BBR, e outros mostram que a imparcialidade entre fluxos BBR é até possível. Outro foco importante é evitar altas taxas de retransmissão, nas quais, em alguns casos, até 10% dos pacotes são retransmitidos.

DCTCP

Concluímos com um exemplo de uma situação em que uma variante do algoritmo de controle de congestionamento TCP foi projetada para funcionar em conjunto com o ECN: em data centers em nuvem. A combinação é chamada de DCTCP, sigla para *Data Center TCP* . A situação é única, pois um data center é autocontido e, portanto, é possível implantar uma versão personalizada do TCP que não precisa se preocupar em tratar outros fluxos TCP de forma justa. Os data centers também são únicos por serem construídos usando switches comuns e, como não há necessidade de se preocupar com longos e grossos canais que abrangem um continente, os switches são normalmente provisionados sem excesso de buffers.

A ideia é simples. O DCTCP adapta o ECN estimando a fração de bytes que encontra congestionamento, em vez de simplesmente detectar que algum congestionamento está prestes a ocorrer. Nos hosts finais, o DCTCP dimensiona a janela de

congestionamento com base nessa estimativa. O algoritmo TCP padrão ainda entra em ação caso um pacote seja realmente perdido. A abordagem foi projetada para atingir alta tolerância a picos, baixa latência e alta taxa de transferência com switches com buffer raso.

O principal desafio que o DCTCP enfrenta é estimar a fração de bytes que encontram congestionamento. Cada switch é simples. Se um pacote chega e o switch detecta que o comprimento da fila (K) está acima de um determinado limite; por exemplo,

$$K > (RTT \times C)/7$$

onde C é a taxa de enlace em pacotes por segundo, então o switch define o bit CE no cabeçalho IP. A complexidade do RED não é necessária.

O receptor então mantém uma variável booleana para cada fluxo, que denotaremos `SeenCE`, e implementa a seguinte máquina de estados em resposta a cada pacote recebido:

- Se o bit CE estiver definido e `SeenCE=False`, defina `SeenCE`como Verdadeiro e envie um ACK imediato.
- Se o bit CE não estiver definido e `SeenCE=True`, defina `SeenCE`como Falso e envie um ACK imediato.
- Caso contrário, ignore o bit CE.

A consequência não óbvia do caso "caso contrário" é que o receptor continua a enviar ACKs atrasados a cada *n* pacotes, independentemente de o bit CE estar definido ou não. Isso se mostrou importante para manter o alto desempenho.

Por fim, o remetente calcula a fração de bytes que encontraram congestionamento durante a janela de observação anterior (geralmente escolhida como aproximadamente o RTT), como a razão entre o total de bytes transmitidos e os bytes confirmados com o sinalizador ECE definido. O DCTCP aumenta a janela de congestionamento

exatamente da mesma forma que o algoritmo padrão, mas reduz a janela proporcionalmente ao número de bytes que encontraram congestionamento durante a última janela de observação.

6.5 Qualidade do Serviço

A promessa das redes comutadas por pacotes de uso geral é que elas suportam todos os tipos de aplicações e dados, incluindo aplicações multimídia que transmitem fluxos de áudio e vídeo digitalizados. No início, um obstáculo para o cumprimento dessa promessa era a necessidade de links com maior largura de banda. Isso não é mais um problema, mas transmitir áudio e vídeo em uma rede envolve mais do que apenas fornecer largura de banda suficiente.

Participantes de uma conversa telefônica, por exemplo, esperam poder conversar de forma que uma pessoa possa responder a algo dito pela outra e ser ouvida quase imediatamente. Assim, a pontualidade da entrega pode ser muito importante. Nos referimos a aplicativos que são sensíveis à pontualidade dos dados como *aplicativos em tempo real* . Aplicativos de voz e vídeo tendem a ser os exemplos canônicos, mas existem outros, como o controle industrial — você gostaria que um comando enviado a um braço robótico o alcançasse antes que o braço colidisse com algo. Até mesmo aplicativos de transferência de arquivos podem ter restrições de pontualidade, como a exigência de que uma atualização de banco de dados seja concluída durante a noite antes que o negócio que precisa dos dados seja retomado no dia seguinte.

A característica distintiva das aplicações em tempo real é que elas precisam de algum tipo de garantia *da rede* de que os dados provavelmente chegarão no prazo (para alguma definição de "no prazo"). Enquanto uma aplicação que não seja em tempo real pode usar uma estratégia de retransmissão ponta a ponta para garantir que os dados cheguem *corretamente* , tal estratégia não pode fornecer pontualidade: a retransmissão

apenas adiciona à latência total se os dados chegarem atrasados. A chegada oportuna deve ser fornecida pela própria rede (os roteadores), não apenas nas bordas da rede (os hosts). Portanto, concluímos que o modelo de melhor esforço, no qual a rede tenta entregar seus dados, mas não faz promessas e deixa a operação de limpeza para as bordas, não é suficiente para aplicações em tempo real. O que precisamos é de um novo modelo de serviço, no qual as aplicações que precisam de garantias mais altas possam solicitá-las à rede. A rede pode então responder fornecendo uma garantia de que fará melhor ou talvez dizendo que não pode prometer nada melhor no momento. Observe que esse modelo de serviço é um superconjunto do modelo original: aplicações que se adaptam ao serviço de melhor esforço devem ser capazes de usar o novo modelo de serviço; seus requisitos são apenas menos rigorosos. Isso implica que a rede tratará alguns pacotes de forma diferente de outros — algo que não ocorre no modelo de melhor esforço. Uma rede que consegue fornecer esses diferentes níveis de serviço costuma ser considerada compatível com a qualidade de serviço (QoS).

6.5.1 Requisitos de aplicação

Antes de analisar os vários protocolos e mecanismos que podem ser usados para fornecer qualidade de serviço aos aplicativos, devemos tentar entender quais são as necessidades desses aplicativos. Para começar, podemos dividir os aplicativos em dois tipos: em tempo real e não em tempo real. Estes últimos são às vezes chamados de aplicativos *de dados tradicionais* , uma vez que tradicionalmente são os principais aplicativos encontrados em redes de dados. Eles incluem os aplicativos mais populares, como SSH, transferência de arquivos, e-mail, navegação na web e assim por diante. Todos esses aplicativos podem funcionar sem garantias de entrega oportuna de dados. Outro termo para essa classe de aplicativos não em tempo real é *elástico* , uma vez que eles são capazes de se esticar graciosamente diante do aumento do atraso. Observe que esses aplicativos podem se beneficiar de atrasos mais curtos, mas eles não se tornam inutilizáveis à medida que os atrasos aumentam. Observe também que seus requisitos de atraso variam de aplicativos interativos, como

SSH, a aplicativos mais assíncronos, como e-mail, com transferências interativas em massa, como transferência de arquivos, no meio.

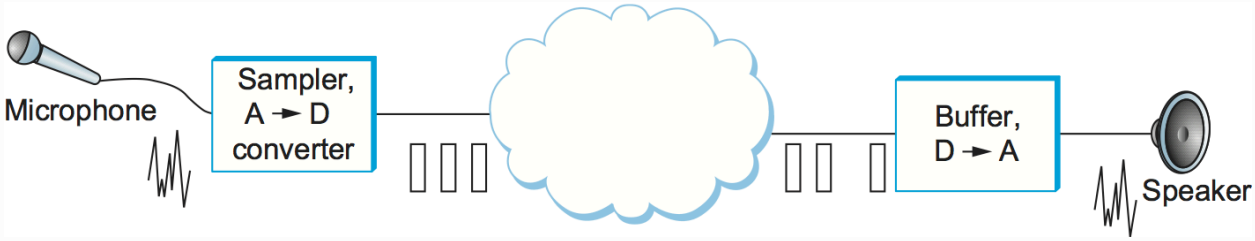

Figura 172. *Um aplicativo de áudio.*

Exemplo de áudio em tempo real

Como exemplo concreto de uma aplicação em tempo real, considere uma aplicação de áudio semelhante à ilustrada na Figura 172. Os dados são gerados pela coleta de amostras de um microfone e sua digitalização usando um conversor analógico-digital (A-D). As amostras digitais são colocadas em pacotes, que são transmitidos pela rede e recebidos na outra extremidade. No host receptor, os dados devem ser *reproduzidos* a uma taxa apropriada. Por exemplo, se as amostras de voz foram coletadas a uma taxa de uma a cada 125 μs, elas devem ser reproduzidas na mesma taxa. Assim, podemos pensar em cada amostra como tendo um *tempo de reprodução* específico : o ponto no tempo em que ela é necessária no host receptor. No exemplo da voz, cada amostra tem um tempo de reprodução 125 μs posterior à amostra anterior. Se os dados chegarem após seu tempo de reprodução apropriado, seja porque foram atrasados na rede ou porque foram descartados e posteriormente retransmitidos, eles são essencialmente inúteis. É a completa inutilidade dos dados atrasados que caracteriza as aplicações em tempo real. Em aplicações elásticas, pode ser bom se os dados aparecerem na hora certa, mas ainda podemos usá-los quando isso não acontece.

Uma maneira de fazer nossa aplicação de voz funcionar seria garantir que todas as amostras levem exatamente o mesmo tempo para percorrer a rede. Assim, como as amostras são injetadas a uma taxa de uma a cada 125 μs, elas aparecerão no receptor na mesma taxa, prontas para serem reproduzidas. No entanto, geralmente é difícil

garantir que todos os dados que trafegam por uma rede comutada por pacotes experimentem exatamente o mesmo atraso. Os pacotes encontram filas em switches ou roteadores, e os comprimentos dessas filas variam com o tempo, o que significa que os atrasos tendem a variar com o tempo e, como consequência, são potencialmente diferentes para cada pacote no fluxo de áudio. A maneira de lidar com isso no receptor é armazenar em buffer uma certa quantidade de dados em reserva, sempre fornecendo assim um estoque de pacotes aguardando para serem reproduzidos no momento certo. Se um pacote sofrer um atraso curto, ele fica no buffer até que chegue o momento de reprodução. Se sofrer um atraso longo, não precisará ser armazenado por muito tempo no buffer do receptor antes de ser reproduzido. Assim, adicionamos efetivamente um deslocamento constante ao tempo de reprodução de todos os pacotes como uma forma de segurança. Chamamos esse deslocamento de *ponto de reprodução.* O único problema que enfrentamos é se os pacotes ficarem atrasados na rede por um tempo tão longo que cheguem após o tempo de reprodução, causando o esgotamento do buffer de reprodução.

A operação de um buffer de reprodução é ilustrada na Figura 173. A linha diagonal à esquerda mostra os pacotes sendo gerados a uma taxa constante. A linha ondulada mostra quando os pacotes chegam, um tempo variável após serem enviados, dependendo do que encontraram na rede. A linha diagonal à direita mostra os pacotes sendo reproduzidos a uma taxa constante, após permanecerem no buffer de reprodução por algum período de tempo. Contanto que a linha de reprodução esteja suficientemente à direita no tempo, a variação no atraso da rede nunca é percebida pelo aplicativo. No entanto, se movermos a linha de reprodução um pouco para a esquerda, alguns pacotes começarão a chegar tarde demais para serem úteis.

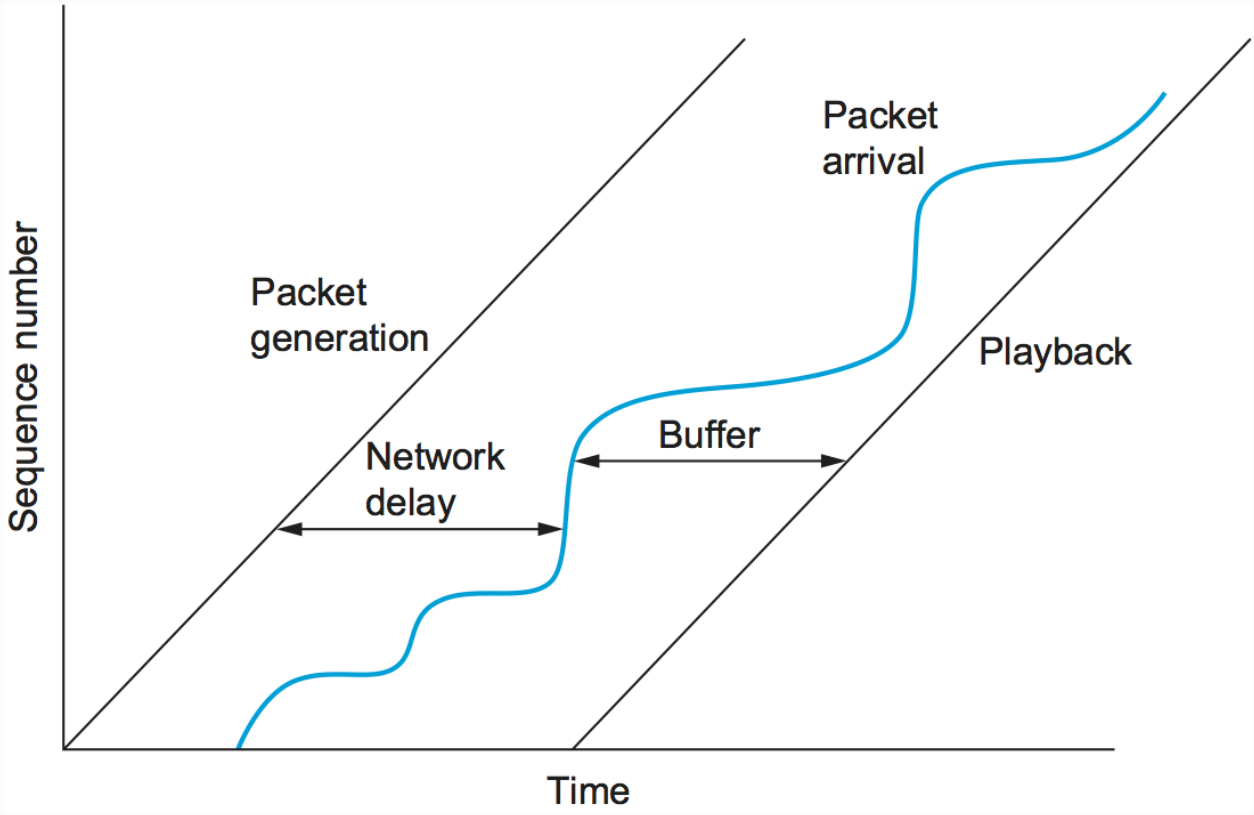

Figura 173. *Um buffer de reprodução.*

Para nossa aplicação de áudio, há limites para o quanto podemos atrasar a reprodução de dados. É difícil manter uma conversa se o tempo entre o momento em que você fala e o momento em que seu ouvinte ouve você for superior a 300 ms. Portanto, o que queremos da rede neste caso é uma garantia de que todos os nossos dados chegarão dentro de 300 ms. Se os dados chegarem antes, nós os armazenamos em buffer até o momento correto de reprodução. Se chegarem depois, não teremos utilidade para eles e devemos descartá-los.

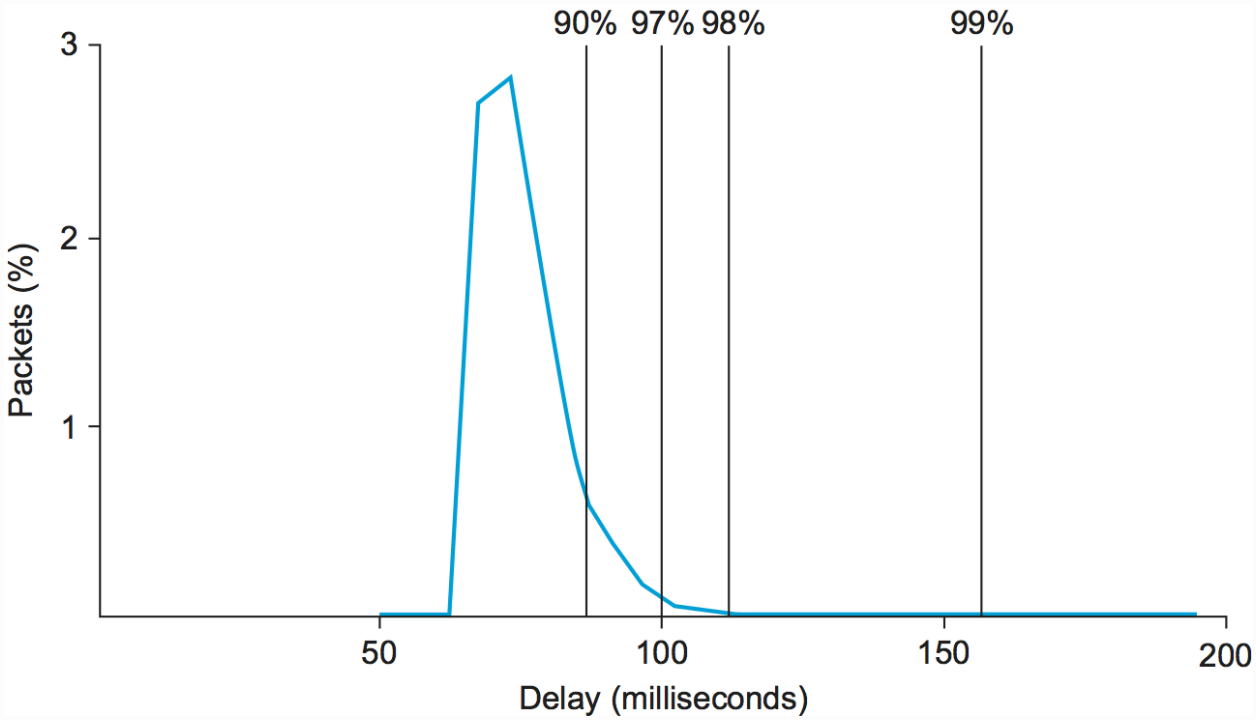

Figura 174. *Exemplo de distribuição de atrasos para uma conexão de Internet.*

Para melhor compreender a variação do atraso da rede, a Figura 174 mostra o atraso unidirecional medido em um determinado caminho pela internet ao longo de um dia específico. Embora os números exatos variem dependendo do caminho e da data, o fator-chave aqui é a *variabilidade* do atraso, que é consistentemente encontrada em quase todos os caminhos, a qualquer momento. Conforme indicado pelas porcentagens cumulativas apresentadas na parte superior do gráfico, 97% dos pacotes neste caso tiveram uma latência de 100 ms ou menos. Isso significa que, se nosso aplicativo de áudio de exemplo definisse o ponto de reprodução em 100 ms, em média, 3 em cada 100 pacotes chegariam tarde demais para serem úteis. Um ponto importante a ser observado neste gráfico é que a cauda da curva — o quanto ela se estende para a direita — é muito longa. Teríamos que definir o ponto de reprodução em mais de 200 ms para garantir que todos os pacotes chegassem a tempo.

Taxonomia de Aplicações em Tempo Real

Agora que temos uma ideia concreta de como funcionam as aplicações em tempo real, podemos analisar algumas classes diferentes de aplicações que servem para motivar nosso modelo de serviço. A taxonomia a seguir deve muito ao trabalho de Clark, Braden, Shenker e Zhang, cujos artigos sobre o assunto podem ser encontrados na seção Leituras Complementares deste capítulo. A taxonomia das aplicações está resumida na Figura 175 .

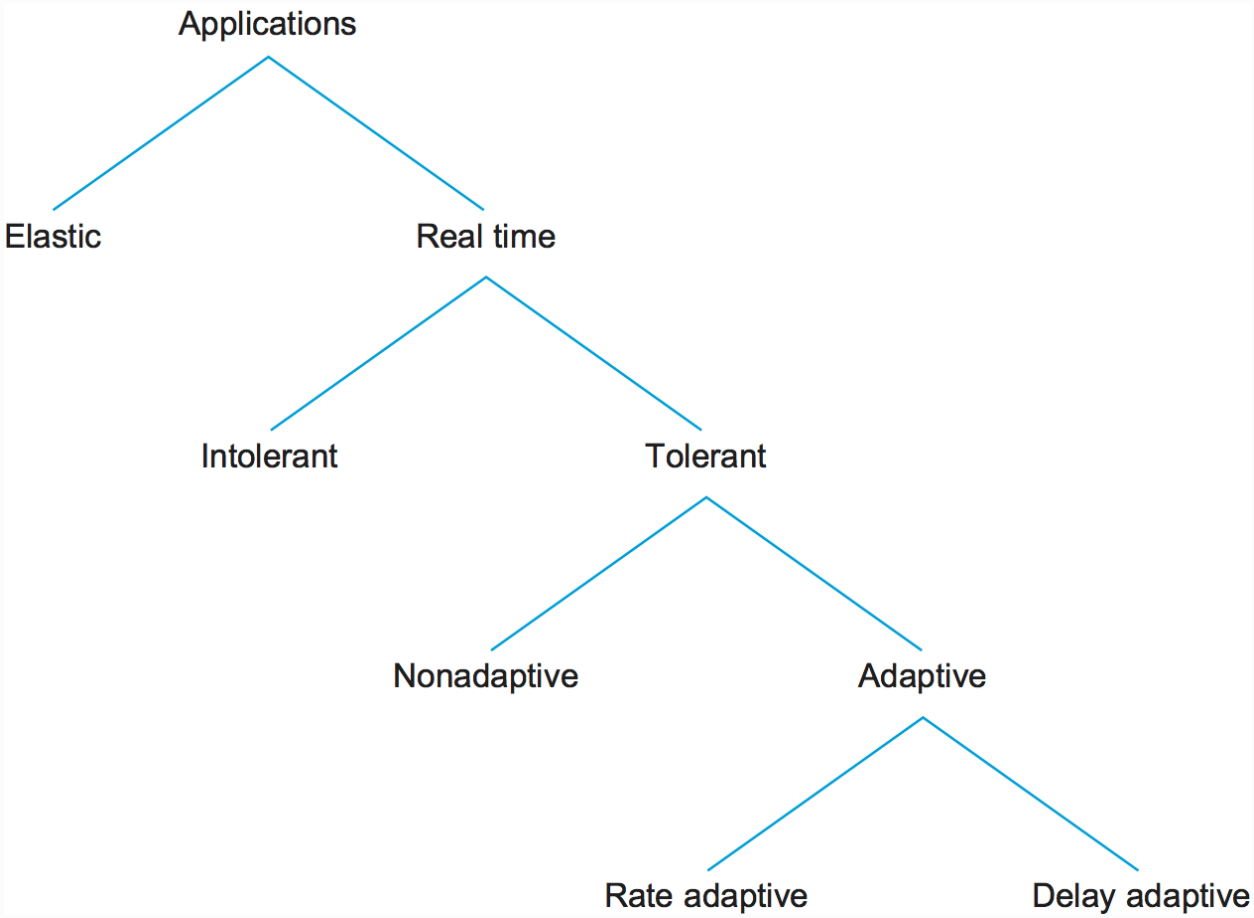

Figura 175. *Taxonomia de aplicações.*

A primeira característica pela qual podemos categorizar as aplicações é sua tolerância à perda de dados, onde a "perda" pode ocorrer porque um pacote chegou tarde demais para ser reproduzido, bem como decorrente das causas usuais na rede. Por um lado, uma amostra de áudio perdida pode ser interpolada a partir das amostras circundantes com relativamente pouco efeito na qualidade de áudio percebida. É somente à medida

que mais e mais amostras são perdidas que a qualidade decai a ponto de a fala se tornar incompreensível. Por outro lado, um programa de controle de robô provavelmente é um exemplo de aplicação em tempo real que não tolera perdas — perder o pacote que contém o comando que instrui o braço do robô a parar é inaceitável. Assim, podemos categorizar as aplicações em tempo real como *tolerantes* ou *intolerantes,* dependendo de sua tolerância a perdas ocasionais. (A propósito, observe que muitas aplicações em tempo real são mais tolerantes a perdas ocasionais do que aplicações que não são em tempo real; por exemplo, compare nossa aplicação de áudio à transferência de arquivos, onde a perda não corrigida de um bit pode tornar um arquivo completamente inútil.)

Uma segunda maneira de caracterizar aplicações em tempo real é por sua adaptabilidade. Por exemplo, uma aplicação de áudio pode ser capaz de se adaptar à quantidade de atraso que os pacotes sofrem ao atravessar a rede. Se notarmos que os pacotes quase sempre chegam em até 300 ms após o envio, podemos definir nosso ponto de reprodução de acordo, armazenando em buffer quaisquer pacotes que cheguem em menos de 300 ms. Suponha que, posteriormente, observemos que todos os pacotes chegam em até 100 ms após o envio. Se aumentássemos nosso ponto de reprodução para 100 ms, os usuários da aplicação provavelmente perceberiam uma melhoria. O processo de mudança do ponto de reprodução exigiria, na verdade, que reproduzíssemos amostras a uma taxa maior por algum período de tempo. Com uma aplicação de voz, isso pode ser feito de uma forma quase imperceptível, simplesmente encurtando os silêncios entre as palavras. Portanto, o ajuste do ponto de reprodução é bastante fácil neste caso e foi implementado com eficácia para diversas aplicações de voz, como o programa de teleconferência de áudio conhecido como `vat`. Observe que o ajuste do ponto de reprodução pode ocorrer em qualquer direção, mas isso, na verdade, envolve distorcer o sinal reproduzido durante o período de ajuste, e que os efeitos dessa distorção dependerão muito de como o usuário final usa os dados.

Observe que, se definirmos nosso ponto de reprodução partindo do pressuposto de que todos os pacotes chegarão em 100 ms e, em seguida, descobrirmos que alguns

pacotes estão chegando ligeiramente atrasados, teremos que descartá-los, ao passo que não teríamos que descartá-los se tivéssemos deixado o ponto de reprodução em 300 ms. Portanto, devemos avançar o ponto de reprodução somente quando ele fornecer uma vantagem perceptível e somente quando tivermos alguma evidência de que o número de pacotes atrasados será aceitavelmente pequeno. Podemos fazer isso devido ao histórico recente observado ou a alguma garantia da rede.

Chamamos de aplicativos que podem ajustar o *atraso do ponto de reprodução de aplicativos adaptativos ao atraso* . Outra classe de aplicativos adaptativos é *a adaptativa à taxa* . Por exemplo, muitos algoritmos de codificação de vídeo podem equilibrar taxa de bits com qualidade. Assim, se descobrirmos que a rede suporta uma determinada largura de banda, podemos definir nossos parâmetros de codificação de acordo. Se mais largura de banda ficar disponível posteriormente, podemos alterar os parâmetros para aumentar a qualidade.

Abordagens para suporte de QoS

Considerando esse amplo escopo de requisitos de aplicação, o que precisamos é de um modelo de serviço mais completo que atenda às necessidades de qualquer aplicação. Isso nos leva a um modelo de serviço não com apenas uma classe (melhor esforço), mas com várias classes, cada uma disponível para atender às necessidades de um conjunto específico de aplicações. Para isso, estamos prontos para analisar algumas das abordagens que foram desenvolvidas para fornecer uma gama de qualidades de serviço. Estas podem ser divididas em duas grandes categorias:

- *Abordagens granulares* , que fornecem QoS para aplicações ou fluxos individuais
- *Abordagens de granulação grossa* , que fornecem QoS para grandes classes de dados ou tráfego agregado

Na primeira categoria, encontramos *os Serviços Integrados* , uma arquitetura de QoS desenvolvida no IETF e frequentemente associada ao Protocolo de Reserva de

Recursos (RSVP). Na segunda categoria, encontram-se *os Serviços Diferenciados* , que provavelmente são o mecanismo de QoS mais amplamente implantado atualmente. Discutiremos esses mecanismos nas próximas duas subseções.

Por fim, como sugerimos no início desta seção, adicionar suporte a QoS à rede não é necessariamente a única solução para o suporte a aplicações em tempo real. Concluímos nossa discussão revisitando o que o host final pode fazer para oferecer melhor suporte a fluxos em tempo real, independentemente da ampla implantação de mecanismos de QoS, como Serviços Integrados ou Diferenciados.

6.5.2 Serviços Integrados (RSVP)

O termo *Serviços Integrados* (frequentemente chamado de IntServ) refere-se a um conjunto de trabalhos produzidos pelo IETF entre 1995 e 1997. O grupo de trabalho do IntServ desenvolveu especificações de diversas *classes de serviço* projetadas para atender às necessidades de alguns dos tipos de aplicação descritos acima. Também definiu como o RSVP poderia ser usado para fazer reservas usando essas classes de serviço. Os parágrafos a seguir fornecem uma visão geral dessas especificações e dos mecanismos usados para implementá-las.

Classes de serviço

Uma das classes de serviço é projetada para aplicações intolerantes. Essas aplicações exigem que um pacote nunca chegue atrasado. A rede deve garantir que o atraso máximo que qualquer pacote experimentará tenha um valor especificado; a aplicação pode então definir seu ponto de reprodução para que nenhum pacote chegue após seu tempo de reprodução. Assumimos que a chegada antecipada de pacotes sempre pode ser tratada por buffering. Esse serviço é chamado de serviço *garantido* .

Além do serviço garantido, a IETF considerou vários outros serviços, mas acabou optando por um para atender às necessidades de aplicações tolerantes e adaptativas. O serviço é conhecido como *carga controlada* e foi motivado pela observação de que

aplicações existentes desse tipo funcionam muito bem em redes que não são muito carregadas. Aplicações de áudio, por exemplo, ajustam seu ponto de reprodução conforme o atraso da rede varia e produzem qualidade de áudio razoável desde que as taxas de perda permaneçam na ordem de 10% ou menos.

O objetivo do serviço de carga controlada é emular uma rede com carga leve para as aplicações que solicitam o serviço, mesmo que a rede como um todo possa estar, de fato, muito carregada. O segredo para isso é usar um mecanismo de enfileiramento, como o WFQ, para isolar o tráfego de carga controlada do restante do tráfego, e alguma forma de controle de admissão para limitar a quantidade total de tráfego de carga controlada em um link, de forma que a carga seja mantida razoavelmente baixa. Discutiremos o controle de admissão com mais detalhes a seguir.

Claramente, essas duas classes de serviço são um subconjunto de todas as classes que podem ser fornecidas. De fato, outros serviços foram especificados, mas nunca padronizados como parte do trabalho do IETF. Até o momento, os dois serviços descritos acima (juntamente com o melhor esforço tradicional) provaram ser flexíveis o suficiente para atender às necessidades de uma ampla gama de aplicações.

Visão geral dos mecanismos

Agora que ampliamos nosso modelo de serviço de melhor esforço com algumas novas classes de serviço, a próxima questão é como implementar uma rede que forneça esses serviços às aplicações. Esta seção descreve os principais mecanismos. Ao ler esta seção, tenha em mente que os mecanismos descritos ainda estão sendo aprimorados pela comunidade de design da Internet. O principal ponto a ser retirado da discussão é uma compreensão geral das partes envolvidas no suporte ao modelo de serviço descrito acima.

Primeiro, enquanto com um serviço de melhor esforço podemos simplesmente dizer à rede para onde queremos que nossos pacotes vão e deixá-la assim, um serviço em tempo real envolve dizer à rede algo mais sobre o tipo de serviço que precisamos.

Podemos dar a ela informações qualitativas, como "usar um serviço de carga controlada", ou informações quantitativas, como "preciso de um atraso máximo de 100 ms". Além de descrever o que queremos, precisamos dizer à rede algo sobre o que vamos injetar nela, já que uma aplicação de baixa largura de banda exigirá menos recursos de rede do que uma aplicação de alta largura de banda. O conjunto de informações que fornecemos à rede é chamado de *flowspec* . Esse nome vem da ideia de que um conjunto de pacotes associados a uma única aplicação e que compartilham requisitos comuns é chamado de *fluxo* , consistente com nosso uso do termo na seção anterior, descrevendo as questões relevantes.

Em segundo lugar, quando solicitamos à rede que nos forneça um serviço específico, a rede precisa decidir se pode, de fato, fornecê-lo. Por exemplo, se 10 usuários solicitarem um serviço em que cada um usará consistentemente 2 Mbps de capacidade de link, e todos compartilharem um link com capacidade de 10 Mbps, a rede terá que recusar alguns deles. O processo de decidir quando recusar é chamado de *controle de admissão* .

Terceiro, precisamos de um mecanismo pelo qual os usuários da rede e os componentes da própria rede troquem informações, como solicitações de serviço, especificações de fluxo e decisões de controle de admissão. Isso às vezes é chamado de *sinalização* , mas como essa palavra tem vários significados, chamamos esse processo de *reserva de recursos* , e ele é realizado por meio de um protocolo de reserva de recursos.

Por fim, após a descrição dos fluxos e seus requisitos, e a tomada das decisões de controle de admissão, os switches e roteadores da rede precisam atender aos requisitos dos fluxos. Uma parte fundamental do atendimento a esses requisitos é gerenciar a maneira como os pacotes são enfileirados e agendados para transmissão nos switches e roteadores. Este último mecanismo é *o agendamento de pacotes* .

Especificações de fluxo

A especificação de fluxo possui duas partes separáveis: a parte que descreve as características de tráfego do fluxo (chamada *TSpec*) e a parte que descreve o serviço solicitado da rede (*RSpec*). A RSpec é muito específica para o serviço e relativamente fácil de descrever. Por exemplo, com um serviço de carga controlada, a RSpec é trivial: a aplicação apenas solicita o serviço de carga controlada sem parâmetros adicionais. Com um serviço garantido, você pode especificar uma meta ou limite de atraso. (Na especificação de serviço garantido da IETF, você especifica não um atraso, mas outra quantidade a partir da qual o atraso pode ser calculado.)

O TSpec é um pouco mais complicado. Como nosso exemplo acima mostrou, precisamos fornecer à rede informações suficientes sobre a largura de banda usada pelo fluxo para permitir a tomada de decisões inteligentes de controle de admissão. Para a maioria das aplicações, no entanto, a largura de banda não é um número único; é algo que varia constantemente. Uma aplicação de vídeo, por exemplo, geralmente gera mais bits por segundo quando a cena está mudando rapidamente do que quando está parada. Apenas conhecer a largura de banda média de longo prazo não é suficiente, como ilustra o exemplo a seguir. Suponha que temos 10 fluxos que chegam a um switch em portas de entrada separadas e que todos saem no mesmo link de 10 Mbps. Suponha que, em um intervalo adequadamente longo, cada fluxo possa enviar no máximo 1 Mbps. Você pode pensar que isso não representa um problema. No entanto, se essas forem aplicações com taxa de bits variável, como vídeo compactado, elas ocasionalmente enviarão mais do que suas taxas médias. Se fontes suficientes enviarem a taxas acima de suas taxas médias, a taxa total na qual os dados chegam ao switch será maior que 10 Mbps. Esses dados excedentes serão enfileirados antes de poderem ser enviados pelo link. Quanto mais tempo essa condição persistir, maior será a fila. Pacotes podem precisar ser descartados e, mesmo que isso não aconteça, os dados na fila estão sendo atrasados. Se os pacotes forem atrasados por tempo suficiente, o serviço solicitado não será fornecido.

Exatamente como gerenciamos nossas filas para controlar o atraso e evitar a perda de pacotes é algo que discutiremos a seguir. No entanto, observe aqui que precisamos

saber algo sobre como a largura de banda de nossas fontes varia com o tempo. Uma maneira de descrever as características de largura de banda das fontes é chamada de filtro *de balde de tokens* . Esse filtro é descrito por dois parâmetros: uma taxa de token *r* e uma profundidade de balde *B.* Ele funciona da seguinte maneira. Para poder enviar um byte, preciso de um token. Para enviar um pacote de comprimento *n* , preciso *de n* tokens. Começo sem tokens e os acumulo a uma taxa de *r* por segundo. Não posso acumular mais do que *B* tokens. Isso significa que posso enviar uma rajada de até *B* bytes para a rede na velocidade que quiser, mas em um intervalo suficientemente longo não posso enviar mais do que *r* bytes por segundo. Acontece que essa informação é muito útil para o algoritmo de controle de admissão quando ele tenta descobrir se pode acomodar uma nova solicitação de serviço.

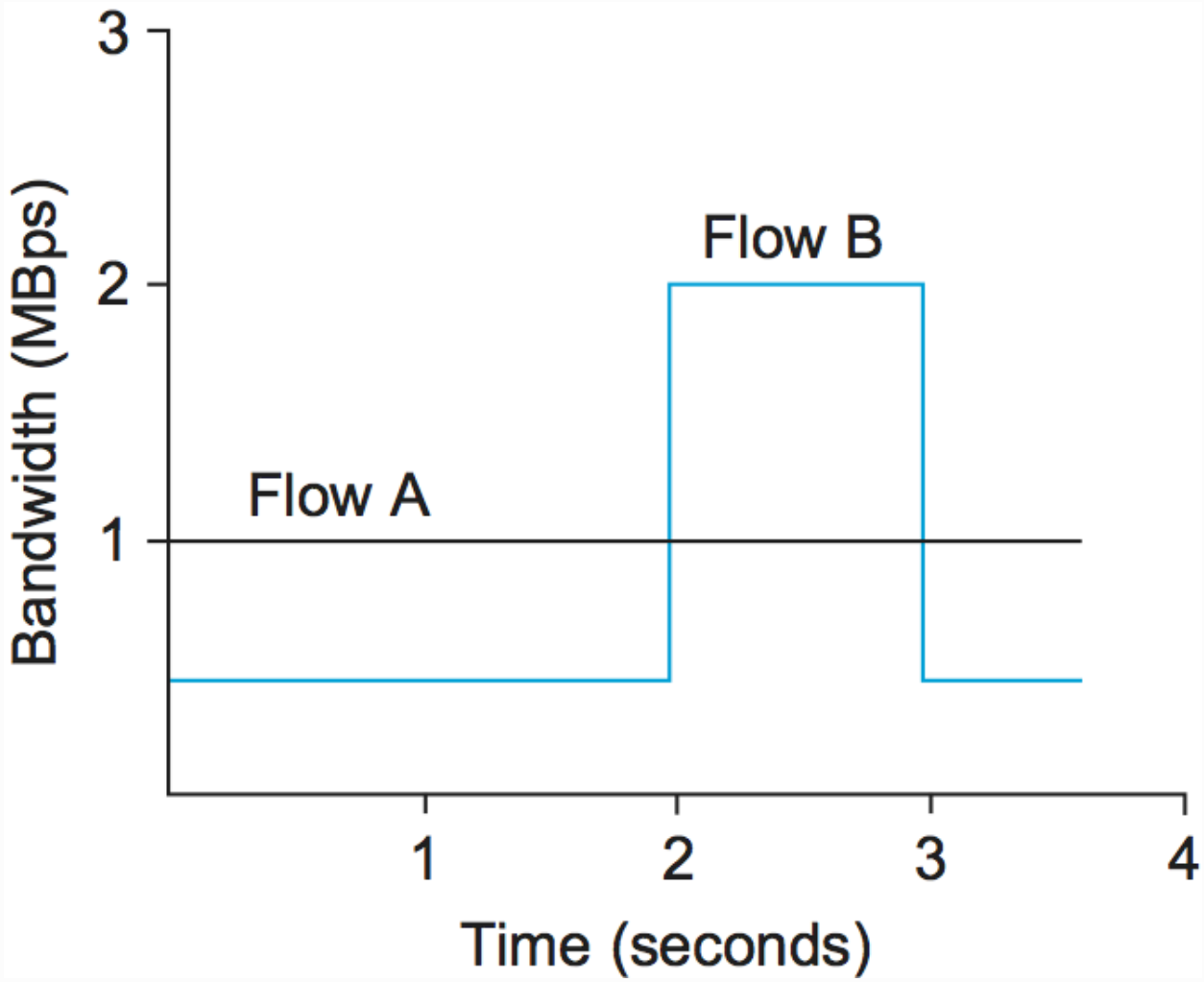

Figura 176. *Dois fluxos com taxas médias iguais, mas descrições diferentes de token bucket.*

A Figura 176 ilustra como um bucket de tokens pode ser usado para caracterizar os requisitos de largura de banda de um fluxo. Para simplificar, suponha que cada fluxo possa enviar dados como bytes individuais em vez de pacotes. O fluxo A gera dados a uma taxa constante de 1 MBps, portanto, pode ser descrito por um filtro de bucket de tokens com uma taxa $r = 1$ MBps e uma profundidade de bucket de 1 byte. Isso significa que ele recebe tokens a uma taxa de 1 MBps, mas não pode armazenar mais de 1 token — ele os gasta imediatamente. O fluxo B também envia a uma taxa que se aproxima de 1 MBps a longo prazo, mas o faz enviando a 0,5 MBps por 2 segundos e, em seguida, a 2 MBps por 1 segundo. Como a taxa r do bucket de tokens é, em certo sentido, uma taxa média de longo prazo, o fluxo B pode ser descrito por um bucket de tokens com uma taxa de 1 MBps. Ao contrário do fluxo A, no entanto, o fluxo B precisa de uma profundidade de bucket B de pelo menos 1 MB, para que possa armazenar tokens enquanto envia a menos de 1 MBps, para serem usados quando enviar a 2 MBps. Nos primeiros 2 segundos neste exemplo, ele recebe tokens a uma taxa de 1 MBps, mas os gasta a apenas 0,5 MBps, o que permite economizar $2 \times 0{,}5 = 1$ MB de tokens, que são gastos no terceiro segundo (junto com os novos tokens que continuam a ser acumulados naquele segundo) para enviar dados a 2 MBps. Ao final do terceiro segundo, tendo gasto os tokens excedentes, ele começa a salvá-los novamente, enviando novamente a 0,5 MBps.

É interessante notar que um único fluxo pode ser descrito por muitos buckets de tokens diferentes. Como exemplo trivial, o fluxo A poderia ser descrito pelo mesmo bucket de tokens que o fluxo B, com uma taxa de 1 MBps e uma profundidade de bucket de 1 MB. O fato de ele nunca precisar acumular tokens não torna essa descrição imprecisa, mas significa que falhamos em transmitir alguma informação útil à rede — o fato de o fluxo A ser, na verdade, muito consistente em suas necessidades de largura de banda. Em geral, é bom ser o mais explícito possível sobre as necessidades de largura de banda de uma aplicação para evitar a superalocação de recursos na rede.

Controle de admissão

A ideia por trás do controle de admissão é simples: quando um novo fluxo deseja receber um determinado nível de serviço, o controle de admissão analisa o TSpec e o RSpec do fluxo e tenta decidir se o serviço desejado pode ser fornecido para aquela quantidade de tráfego, considerando os recursos disponíveis no momento, sem fazer com que qualquer fluxo admitido anteriormente receba um serviço pior do que o solicitado. Se puder fornecer o serviço, o fluxo é admitido; caso contrário, é negado. A parte difícil é descobrir quando dizer sim e quando dizer não.

O controle de admissão depende muito do tipo de serviço solicitado e da disciplina de enfileiramento empregada nos roteadores; discutiremos este último tópico mais adiante nesta seção. Para um serviço garantido, você precisa de um bom algoritmo para tomar uma decisão definitiva de sim/não. A decisão é bastante direta se o enfileiramento justo ponderado for usado em cada roteador. Para um serviço de carga controlada, a decisão pode ser baseada em heurísticas, como "A última vez que permiti um fluxo com este TSpec nesta classe, os atrasos para a classe excederam o limite aceitável, então é melhor eu dizer não" ou "Meus atrasos atuais estão tão dentro dos limites que eu deveria ser capaz de admitir outro fluxo sem dificuldade".

O controle de admissão não deve ser confundido com *policiamento* . O primeiro é uma decisão por fluxo de admitir ou não um novo fluxo. O último é uma função aplicada por pacote para garantir que um fluxo esteja em conformidade com o TSpec que foi usado para fazer a reserva. Se um fluxo não estiver em conformidade com seu TSpec — por exemplo, porque está enviando o dobro de bytes por segundo do que disse que faria — é provável que interfira no serviço fornecido a outros fluxos, e alguma ação corretiva deve ser tomada. Existem várias opções, a mais óbvia sendo descartar pacotes ofensivos. No entanto, outra opção seria verificar se os pacotes realmente estão interferindo no serviço de outros fluxos. Se não estiverem interferindo, os pacotes podem ser enviados após serem marcados com uma tag que diz, na verdade, "Este é um pacote não conforme. Descarte-o primeiro se precisar descartar algum pacote".

O controle de admissão está intimamente relacionado à importante questão da *política* . Por exemplo, um administrador de rede pode desejar permitir que reservas feitas pelo CEO de sua empresa sejam admitidas, enquanto rejeita reservas feitas por funcionários de escalão inferior. É claro que a solicitação de reserva do CEO ainda pode falhar se os recursos solicitados não estiverem disponíveis, portanto, vemos que questões de política e disponibilidade de recursos podem ser abordadas quando as decisões de controle de admissão são tomadas. A aplicação de políticas a redes é uma área que recebe muita atenção no momento da redação deste texto.

Protocolo de Reserva

Embora redes orientadas à conexão sempre tenham precisado de algum tipo de protocolo de configuração para estabelecer o estado do circuito virtual necessário nos switches, redes sem conexão, como a Internet, não possuíam tais protocolos. Como esta seção indicou, no entanto, precisamos fornecer muito mais informações à nossa rede quando queremos um serviço em tempo real dela. Embora tenham sido propostos vários protocolos de configuração para a Internet, aquele em que a atenção atual se concentra mais é o RSVP. Ele é particularmente interessante porque difere substancialmente dos protocolos de sinalização convencionais para redes orientadas à conexão.

Uma das principais premissas subjacentes ao RSVP é que ele não deve prejudicar a robustez encontrada nas redes sem conexão atuais. Como as redes sem conexão dependem de pouco ou nenhum estado armazenado na própria rede, é possível que roteadores travem e reiniciem, e que links fiquem ativos e inativos enquanto a conectividade ponta a ponta ainda é mantida. O RSVP tenta manter essa robustez usando a ideia de *estado suave* nos roteadores. O estado suave — em contraste com o estado rígido encontrado em redes orientadas à conexão — não precisa ser explicitamente excluído quando não é mais necessário. Em vez disso, ele expira após um período relativamente curto (digamos, um minuto) se não for atualizado periodicamente. Veremos mais adiante como isso contribui para a robustez.

Outra característica importante do RSVP é que ele visa suportar fluxos multicast com a mesma eficácia que os fluxos unicast. Isso não é surpreendente, visto que muitas das primeiras aplicações que puderam se beneficiar da melhoria da qualidade de serviço também eram aplicações multicast — ferramentas de videoconferência, por exemplo. Uma das percepções dos projetistas do RSVP é que a maioria das aplicações multicast tem muito mais receptores do que emissores, como exemplificado pela grande plateia e um único palestrante por palestra. Além disso, os receptores podem ter requisitos diferentes. Por exemplo, um receptor pode querer receber dados de apenas um emissor, enquanto outros podem desejar receber dados de todos os emissores. Em vez de os emissores monitorarem um número potencialmente grande de receptores, faz mais sentido deixar que os receptores monitorem suas próprias necessidades. Isso sugere a abordagem *orientada ao receptor* adotada pelo RSVP. Em contraste, as redes orientadas à conexão geralmente deixam a reserva de recursos para o emissor, assim como normalmente é o originador de uma chamada telefônica que faz com que os recursos sejam alocados na rede telefônica.

O estado suave e a natureza orientada ao receptor do RSVP conferem-lhe uma série de boas propriedades. Uma delas é a facilidade de aumentar ou diminuir o nível de alocação de recursos fornecido a um receptor. Como cada receptor envia periodicamente mensagens de atualização para manter o estado suave, é fácil enviar uma nova reserva solicitando um novo nível de recursos. Além disso, o estado suave lida com elegância com a possibilidade de falhas na rede ou nos nós. Em caso de falha de um host, os recursos alocados por esse host a um fluxo expirarão naturalmente e serão liberados. Para ver o que acontece em caso de falha de um roteador ou link, precisamos analisar mais detalhadamente a mecânica de fazer uma reserva.

Inicialmente, considere o caso de um remetente e um destinatário tentando obter uma reserva para o tráfego que flui entre eles. Há duas coisas que precisam acontecer antes que um destinatário possa fazer a reserva. Primeiro, o destinatário precisa saber qual tráfego o remetente provavelmente enviará para que possa fazer uma reserva apropriada. Ou seja, ele precisa saber o TSpec do remetente. Segundo, ele precisa

saber qual caminho os pacotes seguirão do remetente ao destinatário, para que possa estabelecer uma reserva de recursos em cada roteador no caminho. Ambos os requisitos podem ser atendidos enviando uma mensagem do remetente para o destinatário contendo o TSpec. Obviamente, isso leva o TSpec ao destinatário. A outra coisa que acontece é que cada roteador analisa essa mensagem (chamada de mensagem PATH) à medida que ela passa e descobre o *caminho inverso* que será usado para enviar reservas do destinatário de volta ao remetente, em um esforço para levar a reserva a cada roteador no caminho. A construção da árvore multicast, em primeiro lugar, é feita por mecanismos como os descritos em outro capítulo.

Após receber uma mensagem PATH, o receptor envia uma reserva de volta à árvore multicast em uma mensagem RESV. Essa mensagem contém o TSpec do remetente e um RSpec descrevendo os requisitos do receptor. Cada roteador no caminho analisa a solicitação de reserva e tenta alocar os recursos necessários para atendê-la. Se a reserva puder ser feita, a solicitação RESV é repassada ao próximo roteador. Caso contrário, uma mensagem de erro é retornada ao receptor que fez a solicitação. Se tudo correr bem, a reserva correta é instalada em todos os roteadores entre o remetente e o receptor. Enquanto o receptor desejar manter a reserva, ele envia a mesma mensagem RESV aproximadamente uma vez a cada 30 segundos.

Agora podemos ver o que acontece quando um roteador ou link falha. Os protocolos de roteamento se adaptam à falha e criam um novo caminho do remetente ao destinatário. As mensagens PATH são enviadas a cada 30 segundos e podem ser enviadas antes se um roteador detectar uma alteração em sua tabela de encaminhamento. Assim, a primeira mensagem após a estabilização da nova rota chegará ao destinatário pelo novo caminho. A próxima mensagem RESV do destinatário seguirá o novo caminho e, se tudo correr bem, estabelecerá uma nova reserva no novo caminho. Enquanto isso, os roteadores que não estiverem mais no caminho deixarão de receber mensagens RESV, e essas reservas expirarão e serão liberadas. Portanto, o RSVP lida muito bem com mudanças na topologia, desde que as alterações de roteamento não sejam excessivamente frequentes.

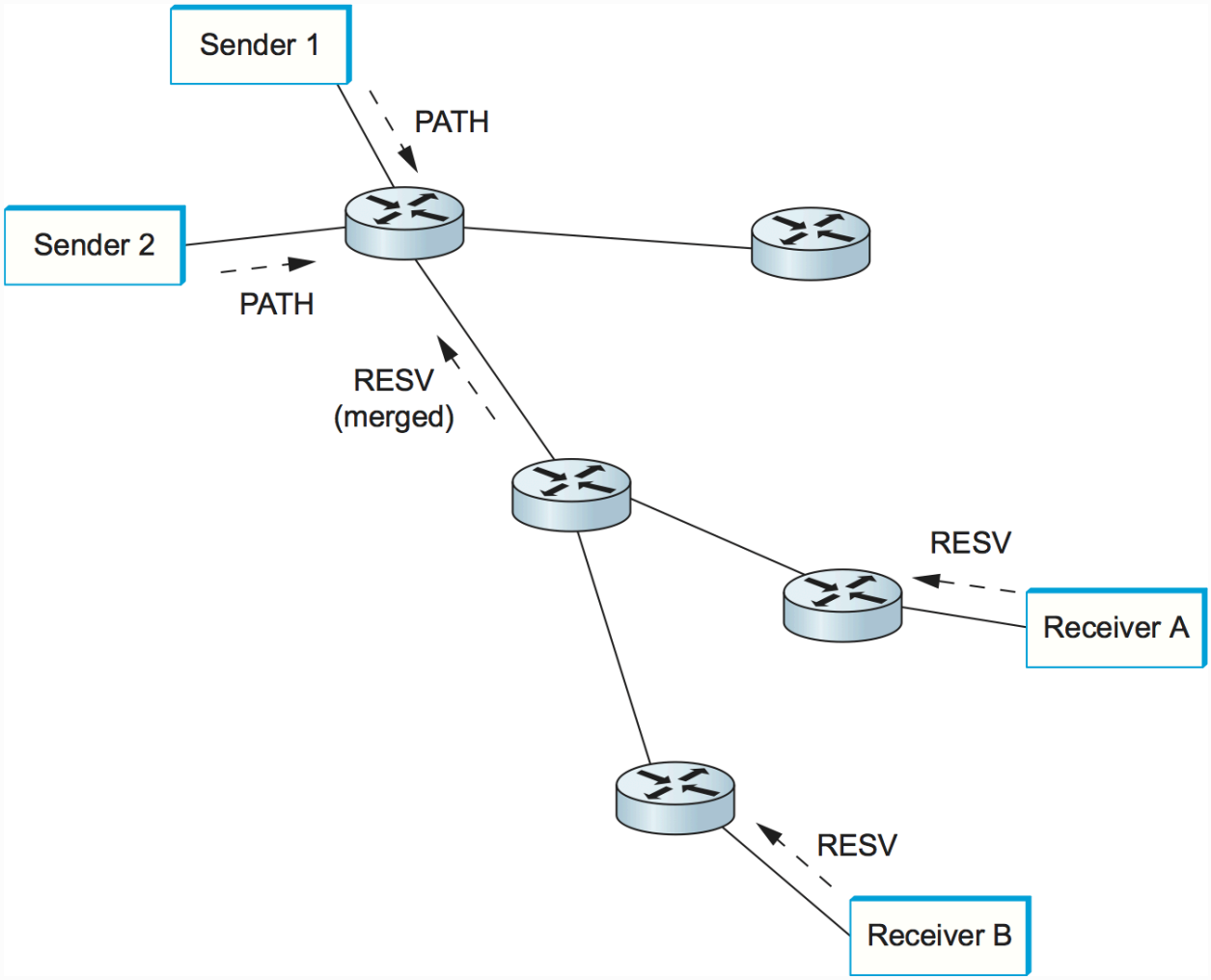

Figura 177. *Fazendo reservas em uma árvore multicast.*

A próxima coisa que precisamos considerar é como lidar com multicast, onde pode haver vários remetentes para um grupo e vários receptores. Essa situação é ilustrada na Figura 177. Primeiro, vamos lidar com vários receptores para um único remetente. À medida que uma mensagem RESV viaja pela árvore de multicast, é provável que ela atinja um pedaço da árvore onde a reserva de algum outro receptor já foi estabelecida. Pode ser o caso de que os recursos reservados a montante deste ponto sejam adequados para atender a ambos os receptores. Por exemplo, se o receptor A já fez uma reserva que prevê um atraso garantido de menos de 100 ms, e a nova solicitação do receptor B é para um atraso de menos de 200 ms, então nenhuma nova reserva é necessária. Por outro lado, se a nova solicitação fosse para um atraso de menos de 50

ms, então o roteador precisaria primeiro ver se poderia aceitar a solicitação; em caso afirmativo, ele enviaria a solicitação a montante. Na próxima vez que o receptor A solicitar um atraso mínimo de 100 ms, o roteador não precisará repassar essa solicitação. Em geral, as reservas podem ser mescladas dessa forma para atender às necessidades de todos os receptores a jusante do ponto de mesclagem.

Se também houver vários remetentes na árvore, os receptores precisam coletar os TSpecs de todos os remetentes e fazer uma reserva grande o suficiente para acomodar o tráfego de todos os remetentes. No entanto, isso pode não significar que os TSpecs precisem ser somados. Por exemplo, em uma audioconferência com 10 palestrantes, não faz muito sentido alocar recursos suficientes para transportar 10 fluxos de áudio, já que o resultado de 10 pessoas falando ao mesmo tempo seria incompreensível. Assim, poderíamos imaginar uma reserva grande o suficiente para acomodar dois palestrantes e não mais. Calcular o TSpec geral correto a partir de todos os TSpecs do remetente é claramente específico da aplicação. Além disso, podemos estar interessados apenas em ouvir um subconjunto de todos os palestrantes possíveis; o RSVP tem diferentes estilos de reserva para lidar com opções como "Reservar recursos para todos os palestrantes", "Reservar recursos para qualquer

n

falantes" e "Reserve recursos apenas para os falantes A e B".

Classificação e escalonamento de pacotes

Depois de descrevermos nosso tráfego e o serviço de rede desejado e instalarmos uma reserva adequada em todos os roteadores do caminho, resta apenas que os roteadores entreguem o serviço solicitado aos pacotes de dados. Há duas coisas que precisam ser feitas:

- Associe cada pacote à reserva apropriada para que ele possa ser manipulado corretamente, um processo conhecido como *classificação de* pacotes.
- Gerencie os pacotes nas filas para que eles recebam o serviço que foi solicitado, um processo conhecido como *agendamento* de pacotes .

A primeira parte é feita examinando até cinco campos no pacote: o endereço de origem, o endereço de destino, o número do protocolo, a porta de origem e a porta de destino. (No IPv6, é possível que o `FlowLabel`campo no cabeçalho possa ser usado para permitir que a pesquisa seja feita com base em uma única chave mais curta.) Com base nessas informações, o pacote pode ser colocado na classe apropriada. Por exemplo, ele pode ser classificado nas classes de carga controladas ou pode fazer parte de um fluxo garantido que precisa ser tratado separadamente de todos os outros fluxos garantidos. Em resumo, há um mapeamento das informações específicas do fluxo no cabeçalho do pacote para um único identificador de classe que determina como o pacote é tratado na fila. Para fluxos garantidos, esse pode ser um mapeamento um-para-um, enquanto para outros serviços pode ser muitos-para-um. Os detalhes da classificação estão intimamente relacionados aos detalhes do gerenciamento de filas.

Deve ficar claro que algo tão simples como uma fila FIFO em um roteador será inadequado para fornecer muitos serviços diferentes e diferentes níveis de atraso em cada serviço. Diversas disciplinas mais sofisticadas de gerenciamento de filas foram discutidas em uma seção anterior, e alguma combinação delas provavelmente será usada em um roteador.

Os detalhes do agendamento de pacotes idealmente não devem ser especificados no modelo de serviço. Em vez disso, esta é uma área onde os implementadores podem tentar fazer coisas criativas para implementar o modelo de serviço de forma eficiente. No caso de serviço garantido, foi estabelecido que uma disciplina de enfileiramento justo ponderado, na qual cada fluxo obtém sua própria fila individual com uma determinada parcela do enlace, fornecerá um limite de atraso ponta a ponta garantido que pode ser facilmente calculado. Para carga controlada, esquemas mais simples

podem ser usados. Uma possibilidade inclui tratar todo o tráfego de carga controlada como um único fluxo agregado (no que diz respeito ao mecanismo de agendamento), com o peso desse fluxo sendo definido com base na quantidade total de tráfego admitido na classe de carga controlada. O problema se torna mais complexo quando se considera que, em um único roteador, muitos serviços diferentes provavelmente serão fornecidos simultaneamente e que cada um desses serviços pode exigir um algoritmo de agendamento diferente. Portanto, algum algoritmo geral de gerenciamento de filas é necessário para gerenciar os recursos entre os diferentes serviços.

Problemas de escalabilidade

Embora a arquitetura de Serviços Integrados e o RSVP representassem um aprimoramento significativo do modelo de serviço de melhor esforço do IP, muitos provedores de serviços de Internet sentiram que não era o modelo certo para implantar. O motivo dessa reticência está relacionado a um dos objetivos fundamentais do projeto do IP: escalabilidade. No modelo de serviço de melhor esforço, os roteadores na Internet armazenam pouco ou nenhum estado sobre os fluxos individuais que passam por eles. Assim, à medida que a Internet cresce, a única coisa que os roteadores precisam fazer para acompanhar esse crescimento é mover mais bits por segundo e lidar com tabelas de roteamento maiores, mas o RSVP levanta a possibilidade de que cada fluxo que passa por um roteador possa ter uma reserva correspondente. Para entender a gravidade desse problema, suponha que cada fluxo em um link OC-48 (2,5 Gbps) represente um fluxo de áudio de 64 kbps. O número desses fluxos é

$$2{,}5 \times 10^{9/64} \times 10^{3} = 39.000$$

Cada uma dessas reservas requer uma certa quantidade de estado que precisa ser armazenada na memória e atualizada periodicamente. O roteador precisa classificar, policiar e enfileirar cada um desses fluxos. Decisões de controle de admissão precisam ser tomadas sempre que um fluxo como esse solicitar uma reserva, e algum mecanismo é necessário para "repelir" os usuários (por exemplo, cobrar seus cartões

de crédito) para que eles não façam reservas arbitrariamente grandes por longos períodos.

Essas preocupações com a escalabilidade limitaram a ampla implantação do IntServ. Devido a essas preocupações, outras abordagens que não exigem tanto estado "por fluxo" foram desenvolvidas. A próxima seção discute algumas dessas abordagens.

6.5.3 Serviços Diferenciados (EF, AF)

Enquanto a arquitetura de Serviços Integrados aloca recursos para fluxos individuais, o modelo de Serviços Diferenciados (frequentemente chamado de DiffServ) aloca recursos para um pequeno número de classes de tráfego. De fato, algumas abordagens propostas para o DiffServ simplesmente dividem o tráfego em duas classes. Esta é uma abordagem eminentemente sensata: se considerarmos a dificuldade que as operadoras de rede enfrentam apenas para manter uma internet de melhor esforço funcionando sem problemas, faz sentido adicionar recursos ao modelo de serviço em pequenos incrementos.

Suponha que decidimos aprimorar o modelo de serviço de melhor esforço adicionando apenas uma nova classe, que chamaremos de "premium". Claramente, precisaremos de alguma forma para descobrir quais pacotes são premium e quais são os tradicionais de melhor esforço. Em vez de usar um protocolo como o RSVP para informar a todos os roteadores que algum fluxo está enviando pacotes premium, seria muito mais fácil se os pacotes pudessem simplesmente se identificar para o roteador ao chegarem. Isso poderia ser feito, obviamente, usando um bit no cabeçalho do pacote — se esse bit for 1, o pacote é premium; se for 0, o pacote é de melhor esforço. Com isso em mente, há duas questões que precisamos abordar:

- Quem define o prêmio e em que circunstâncias?
- O que um roteador faz de diferente quando vê um pacote com o bit definido?

Há muitas respostas possíveis para a primeira pergunta, mas uma abordagem comum é definir o bit em um limite administrativo. Por exemplo, o roteador na borda da rede de um provedor de serviços de Internet pode definir o bit para pacotes que chegam em uma interface que se conecta à rede de uma determinada empresa. O provedor de serviços de Internet pode fazer isso porque essa empresa pagou por um nível de serviço superior ao de melhor esforço. Também é possível que nem todos os pacotes sejam marcados como premium; por exemplo, o roteador pode ser configurado para marcar pacotes como premium até uma determinada taxa máxima e deixar todos os pacotes excedentes como de melhor esforço.

Supondo que os pacotes tenham sido marcados de alguma forma, o que os roteadores que os encontram fazem com eles? Novamente, há muitas respostas. De fato, a IETF padronizou um conjunto de comportamentos de roteador a serem aplicados a pacotes marcados. Esses comportamentos são chamados de *comportamentos por salto* (PHBs), um termo que indica que eles definem o comportamento de roteadores individuais, em vez de serviços ponta a ponta. Como há mais de um novo comportamento, também é necessário mais de um bit no cabeçalho do pacote para informar aos roteadores qual comportamento aplicar. A IETF decidiu pegar o `TOS`byte antigo do cabeçalho IP, que não era amplamente utilizado, e redefini-lo. Seis bits desse byte foram alocados para pontos de código DiffServ (DSCPs), onde cada DSCP é um valor de 6 bits que identifica um PHB específico a ser aplicado a um pacote. (Os dois bits restantes são usados pelo ECN.)

Encaminhamento Acelerado (EF) PHB

Um dos PHBs mais simples de explicar é conhecido como *encaminhamento acelerado* (EF). Pacotes marcados para tratamento EF devem ser encaminhados pelo roteador com o mínimo de atraso e perda. A única maneira de um roteador garantir isso a todos os pacotes EF é se a taxa de chegada de pacotes EF no roteador for estritamente limitada a ser menor do que a taxa na qual o roteador pode encaminhar pacotes EF. Por exemplo, um roteador com uma interface de 100 Mbps precisa ter certeza de que a taxa de chegada de pacotes EF destinados a essa interface nunca exceda 100 Mbps.

Ele também pode querer ter certeza de que a taxa será um pouco abaixo de 100 Mbps, para que ocasionalmente tenha tempo de enviar outros pacotes, como atualizações de roteamento.

A limitação de taxa de pacotes EF é alcançada configurando os roteadores na borda de um domínio administrativo para permitir uma determinada taxa máxima de chegada de pacotes EF ao domínio. Uma abordagem simples, embora conservadora, seria garantir que a soma das taxas de todos os pacotes EF que entram no domínio seja menor que a largura de banda do link mais lento do domínio. Isso garantiria que, mesmo no pior caso, em que todos os pacotes EF convergem para o link mais lento, ele não fique sobrecarregado e possa apresentar o comportamento correto.

Existem várias estratégias de implementação possíveis para o comportamento EF. Uma delas é dar aos pacotes EF prioridade estrita sobre todos os outros pacotes. Outra é realizar um enfileiramento justo ponderado entre os pacotes EF e outros pacotes, com o peso do EF definido como suficientemente alto para que todos os pacotes EF possam ser entregues rapidamente. Isso tem uma vantagem sobre a prioridade estrita: os pacotes não EF podem ter certeza de obter algum acesso ao link, mesmo que a quantidade de tráfego EF seja excessiva. Isso pode significar que os pacotes EF não apresentem exatamente o comportamento especificado, mas também pode impedir que tráfego de roteamento essencial seja bloqueado da rede em caso de carga excessiva de tráfego EF.

Encaminhamento Garantido (AF) PHB

O PHB *de encaminhamento garantido* (AF) tem suas raízes em uma abordagem conhecida como *RED* com Entrada e Saída (RIO) ou RED Ponderado, ambos aprimoramentos do algoritmo RED básico descrito em uma seção anterior. A Figura 178 mostra como o RIO funciona; assim como no RED, vemos a probabilidade de queda no

-eixo aumentando à medida que o comprimento médio da fila aumenta ao longo do

x

-eixo. Mas agora, para nossas duas classes de tráfego, temos duas curvas de probabilidade de queda separadas. O RIO chama as duas classes de "entrada" e "saída" por razões que ficarão claras em breve. Como a curva "saída" tem uma curva menor `MinThreshold`que a curva "entrada", fica claro que, em baixos níveis de congestionamento, apenas pacotes marcados como "saída" serão descartados pelo algoritmo RED. Se o congestionamento se agravar, uma porcentagem maior de pacotes "saída" será descartada e, se o comprimento médio da fila exceder Min in , o RED também começará a descartar pacotes "entrada".

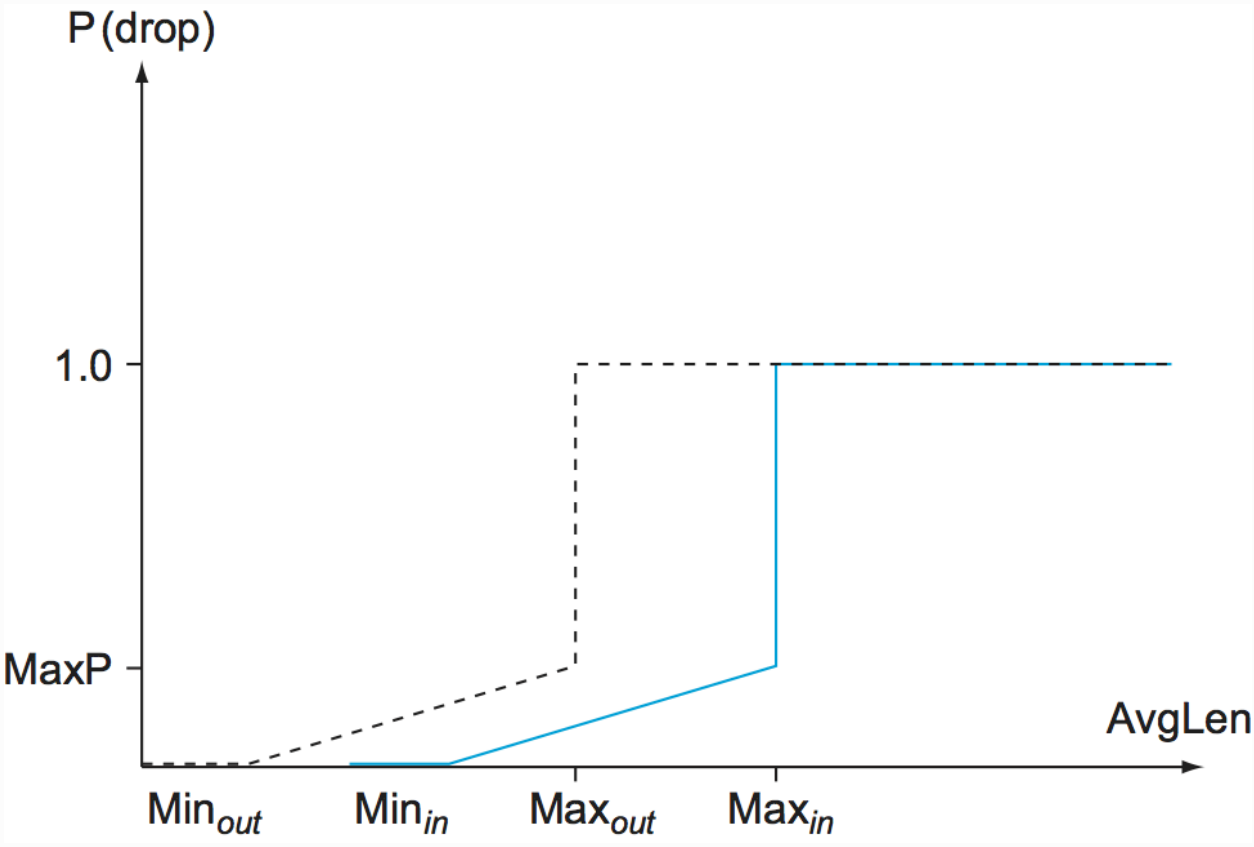

Figura 178. *VERMELHO com probabilidades de queda de entrada e saída.*

A razão para chamar as duas classes de pacotes de "entrada" e "saída" decorre da forma como os pacotes são marcados. Já observamos que a marcação de pacotes pode ser realizada por um roteador na borda de um domínio administrativo. Podemos pensar nesse roteador como estando na fronteira entre um provedor de serviços de rede e algum cliente dessa rede. O cliente pode ser qualquer outra rede — por exemplo, a rede de uma empresa ou de outro provedor de serviços de rede. O cliente e o provedor de serviços de rede concordam com algum tipo de perfil para o serviço garantido (e talvez o cliente pague ao provedor de serviços de rede por esse perfil). O perfil pode ser algo como "O cliente X tem permissão para enviar até

y

Mbps de tráfego garantido", ou pode ser significativamente mais complexo. Seja qual for o perfil, o roteador de borda pode marcar claramente os pacotes que chegam deste cliente como estando dentro ou fora do perfil. No exemplo mencionado, desde que o cliente envie menos de

y

Mbps, todos os seus pacotes serão marcados como "entrada", mas quando ele exceder essa taxa, os pacotes excedentes serão marcados como "saída".

A combinação de um medidor de perfil na borda e RIO em todos os roteadores da rede do provedor de serviços deve fornecer ao cliente uma alta segurança (mas não uma garantia) de que os pacotes dentro de seu perfil possam ser entregues. Em particular, se a maioria dos pacotes, incluindo aqueles enviados por clientes que não pagaram a mais para estabelecer um perfil, forem pacotes de "saída", então, normalmente, o mecanismo RIO atuará para manter o congestionamento baixo o suficiente para que os pacotes de "entrada" raramente sejam descartados. Claramente, deve haver largura de banda suficiente na rede para que os pacotes de "entrada" sozinhos raramente consigam congestionar um link a ponto de o RIO começar a descartar pacotes de "entrada".

Assim como o RED, a eficácia de um mecanismo como o RIO depende, em certa medida, da escolha correta dos parâmetros, e há consideravelmente mais parâmetros a serem definidos para o RIO. No momento da redação deste texto, ainda não se sabe exatamente o quão bem o esquema funcionará em redes de produção.

Uma propriedade interessante do RIO é que ele não altera a ordem dos pacotes de "entrada" e "saída". Por exemplo, se uma conexão TCP estiver enviando pacotes por meio de um medidor de perfil, e alguns pacotes estiverem sendo marcados como "entrada" enquanto outros como "saída", esses pacotes receberão diferentes probabilidades de descarte nas filas do roteador, mas serão entregues ao receptor na mesma ordem em que foram enviados. Isso é importante para a maioria das implementações TCP, que apresentam desempenho muito melhor quando os pacotes chegam em ordem, mesmo que sejam projetadas para lidar com a desordem. Observe também que mecanismos como a retransmissão rápida podem ser acionados erroneamente quando ocorre a desordem.

A ideia do RIO pode ser generalizada para fornecer mais de duas curvas de probabilidade de queda, e essa é a ideia por trás da abordagem conhecida como *RED ponderado* (WRED). Nesse caso, o valor do campo DSCP é usado para escolher uma entre várias curvas de probabilidade de queda, de modo que diversas classes de serviço diferentes possam ser fornecidas.

Uma terceira maneira de fornecer Serviços Diferenciados é usar o valor DSCP para determinar em qual fila colocar um pacote em um escalonador de enfileiramento justo ponderado. Em um caso muito simples, podemos usar um ponto de código para indicar a fila *de melhor esforço* e um segundo ponto de código para selecionar a fila *premium* . Precisamos então escolher um peso para a fila premium que faça com que os pacotes premium obtenham um serviço melhor do que os pacotes de melhor esforço. Isso depende da carga oferecida de pacotes premium. Por exemplo, se atribuirmos à fila premium um peso de 1 e à fila de melhor esforço um peso de 4, isso garante que a largura de banda disponível para os pacotes premium seja

$$\text{Prêmio } B = \text{Prêmio } W / (\text{Prêmio } W + \text{Melhor esforço } W) = 1/(1 + 4) = 0,2$$

Ou seja, reservamos efetivamente 20% do link para pacotes premium; portanto, se a carga oferecida de tráfego premium for de apenas 10% do link, em média, o tráfego premium se comportará como se estivesse em uma rede com carga muito baixa e o serviço será muito bom. Em particular, o atraso experimentado pela classe premium pode ser mantido baixo, já que o WFQ tentará transmitir os pacotes premium assim que eles chegarem nesse cenário. Por outro lado, se a carga de tráfego premium fosse de 30%, ela se comportaria como uma rede altamente carregada, e o atraso poderia ser muito alto para os pacotes premium — até pior do que para os chamados pacotes de melhor esforço. Portanto, o conhecimento da carga oferecida e a definição cuidadosa dos pesos são importantes para esse tipo de serviço. No entanto, observe que a abordagem segura é ser muito conservador ao definir o peso para a fila premium. Se esse peso for muito alto em relação à carga esperada, ele fornecerá uma margem de erro, mas não impedirá que o tráfego de melhor esforço use qualquer largura de banda que tenha sido reservada para pacotes premium, mas não seja usada por pacotes premium.

Assim como no WRED, podemos generalizar essa abordagem baseada em WFQ para permitir mais de duas classes representadas por diferentes pontos de código. Além disso, podemos combinar a ideia de um seletor de fila com uma preferência de descarte. Por exemplo, com 12 pontos de código, podemos ter quatro filas com pesos diferentes, cada uma com três preferências de descarte. É exatamente isso que a IETF fez na definição de "serviço garantido".

6.5.4 Controle de congestionamento baseado em equações

Concluímos nossa discussão sobre QoS retornando ao controle de congestionamento do TCP, mas desta vez no contexto de aplicações em tempo real. Lembre-se de que o TCP ajusta a janela de congestionamento do remetente (e, portanto, a taxa na qual ele pode transmitir) em resposta a eventos de ACK e timeout. Um dos pontos fortes dessa

abordagem é que ela não requer cooperação dos roteadores da rede; é uma estratégia puramente baseada em host. Tal estratégia complementa os mecanismos de QoS que temos considerado, porque (1) as aplicações podem usar soluções baseadas em host sem depender do suporte do roteador e (2) mesmo com o DiffServ totalmente implantado, ainda é possível que uma fila de roteadores seja sobrecarregada, e gostaríamos que as aplicações em tempo real reagissem de forma razoável caso isso acontecesse.

Embora queiramos aproveitar o algoritmo de controle de congestionamento do TCP, o TCP em si não é apropriado para aplicações em tempo real. Um dos motivos é que o TCP é um protocolo confiável, e aplicações em tempo real muitas vezes não podem arcar com os atrasos introduzidos pela retransmissão. No entanto, e se desvinculássemos o TCP de seu mecanismo de controle de congestionamento para adicionar um controle de congestionamento semelhante ao do TCP a um protocolo não confiável como o UDP? Aplicações em tempo real poderiam utilizar tal protocolo?

Por um lado, essa é uma ideia atraente, pois faria com que os fluxos em tempo real competissem de forma justa com os fluxos TCP. A alternativa (que já existe hoje) é que os aplicativos de vídeo usem UDP sem qualquer forma de controle de congestionamento e, como consequência, roubem largura de banda dos fluxos TCP, que recuam na presença de congestionamento. Por outro lado, o comportamento dente de serra do algoritmo de controle de congestionamento do TCP não é apropriado para aplicativos em tempo real; isso significa que a taxa de transmissão do aplicativo está constantemente aumentando e diminuindo. Em contraste, os aplicativos em tempo real funcionam melhor quando conseguem sustentar uma taxa de transmissão estável por um período de tempo relativamente longo.

É possível alcançar o melhor dos dois mundos: compatibilidade com o controle de congestionamento TCP por uma questão de justiça, enquanto se mantém uma taxa de transmissão suave para o bem da aplicação? Trabalhos recentes sugerem que a resposta é sim. Especificamente, vários algoritmos de controle de congestionamento chamados amigáveis ao TCP foram propostos. Esses algoritmos têm dois objetivos

principais. Um é adaptar lentamente a janela de congestionamento. Isso é feito adaptando-se ao longo de períodos de tempo relativamente mais longos (por exemplo, um RTT) em vez de por pacote. Isso suaviza a taxa de transmissão. O segundo é ser amigável ao TCP no sentido de ser justo para fluxos TCP concorrentes. Essa propriedade é frequentemente reforçada garantindo que o comportamento do fluxo obedeça a uma equação que modela o comportamento do TCP. Portanto, essa abordagem às vezes é chamada de *controle de congestionamento baseado em equações* .

Vimos uma forma simplificada da equação da taxa TCP em uma seção anterior. Para nossos propósitos, basta observar que a equação assume esta forma geral:

$$\text{Rate} \propto (1\text{RTT} \times \rho)$$

que diz que para ser amigável ao TCP, a taxa de transmissão deve ser inversamente proporcional ao tempo de ida e volta (RTT) e à raiz quadrada da taxa de perdas (

$$\rho$$

). Em outras palavras, para construir um mecanismo de controle de congestionamento a partir dessa relação, o receptor deve reportar periodicamente ao remetente a taxa de perda que está enfrentando (por exemplo, pode reportar que não recebeu 10% dos últimos 100 pacotes), e o remetente então ajusta sua taxa de envio para cima ou para baixo, de forma que essa relação continue válida. É claro que ainda cabe à aplicação se adaptar a essas mudanças na taxa disponível, mas, como veremos no próximo capítulo, muitas aplicações em tempo real são bastante adaptáveis.

Perspectiva: Engenharia de Tráfego Definida por Software

O problema abrangente que este capítulo aborda é como alocar a largura de banda de rede disponível para um conjunto de fluxos ponta a ponta. Seja controle de congestionamento TCP, serviços integrados ou serviços diferenciados, pressupõe-se que a largura de banda de rede subjacente alocada seja fixa: um link de 1 Gbps entre os sites A e B é sempre um link de 1 Gbps, e os algoritmos se concentram em como melhor compartilhar esse 1 Gbps entre usuários concorrentes. Mas e se esse não for o caso? E se você pudesse adquirir capacidade adicional "instantaneamente", de modo que o link de 1 Gbps fosse atualizado para um link de 10 Gbps, ou talvez pudesse adicionar um novo link entre dois sites que não estavam conectados anteriormente?

Essa possibilidade é real e é um tópico geralmente chamado de *engenharia de tráfego* , um termo que remonta aos primórdios das redes, quando as operadoras analisavam as cargas de trabalho de tráfego em suas redes e as reprojetavam periodicamente para adicionar capacidade quando os links existentes ficavam cronicamente sobrecarregados. Naqueles primeiros dias, a decisão de adicionar capacidade não era tomada de ânimo leve; era preciso ter certeza de que a tendência de uso observada não era apenas um pequeno problema, já que mudar a rede levaria uma quantidade significativa de tempo e dinheiro. Na pior das hipóteses, isso poderia envolver a instalação de cabos através de um oceano ou o lançamento de um satélite no espaço.

Mas com o advento de tecnologias como DWDM (Seção 3.1) e MPLS (Seção 4.4), nem sempre precisamos instalar mais fibra, mas podemos, em vez disso, ativar comprimentos de onda adicionais ou estabelecer novos circuitos entre qualquer par de sites. (Esses sites não precisam ser conectados diretamente por fibra. Por exemplo, um comprimento de onda entre Boston e São Francisco pode passar por ROADMs em Chicago e Denver, mas da perspectiva da topologia de rede L2/L3, Boston e São Francisco são conectados por um link direto.) Isso reduz drasticamente o tempo de disponibilidade, mas a reconfiguração do hardware ainda requer intervenção manual e,

portanto, nossa definição de "instantaneamente" ainda é medida em dias, se não semanas. Afinal, há formulários de requisição para serem preenchidos, em triplicado!

Mas, como vimos repetidamente, uma vez fornecidas as interfaces programáticas corretas, o software pode ser acionado para resolver o problema, e "instantaneamente" pode, para todos os efeitos práticos, ser verdadeiramente instantâneo. É isso que os provedores de nuvem fazem com os backbones privados que constroem para interconectar seus data centers. Por exemplo, o Google descreveu publicamente sua WAN privada, chamada B4, que é construída inteiramente usando switches bare-metal e SDN. A B4 não adiciona/remove comprimentos de onda para ajustar a largura de banda entre nós — ela constrói túneis ponta a ponta dinamicamente usando uma técnica chamada *Equal-Cost Multipath* (ECMP), uma alternativa ao CSPF apresentada na Seção 4.4 — mas a flexibilidade que ela oferece é semelhante.

Um programa de controle de Engenharia de Tráfego (TE) provisiona a rede de acordo com as necessidades de várias classes de aplicações. O B4 identifica três dessas classes: (1) cópia de dados do usuário (por exemplo, e-mail, documentos, áudio/vídeo) para data centers remotos para fins de disponibilidade; (2) acesso ao armazenamento remoto por meio de cálculos executados em fontes de dados distribuídas; e (3) envio de dados em larga escala para sincronizar o estado entre vários data centers. Essas classes são ordenadas em volume crescente, sensibilidade à latência decrescente e prioridade geral decrescente. Por exemplo, dados do usuário representam o menor volume no B4, são os mais sensíveis à latência e têm a maior prioridade.

Ao centralizar o processo de tomada de decisão, uma das vantagens alegadas da SDN, o Google conseguiu elevar a utilização de seus links para perto de 100%. Isso é de duas a três vezes melhor do que a utilização média de 30 a 40% para a qual os links WAN são normalmente provisionados, o que é necessário para permitir que essas redes lidem tanto com picos de tráfego quanto com falhas de link/switch. Se você puder decidir centralmente como alocar recursos em toda a rede, é possível operar a rede muito mais próxima da utilização máxima. Lembre-se de que o provisionamento de links na rede é feito para classes de aplicativos de granularidade grossa. O controle de

congestionamento TCP ainda opera conexão por conexão, e as decisões de roteamento ainda são tomadas com base na topologia B4. (A propósito, vale a pena notar que, como a B4 é uma WAN privada, o Google tem liberdade para executar seu próprio algoritmo de controle de congestionamento, como o BBR, sem medo de que isso prejudique injustamente outros algoritmos.)

Uma lição a ser aprendida com sistemas como o B4 é que a linha entre engenharia de tráfego e controle de congestionamento (bem como entre engenharia de tráfego e roteamento) é tênue. Existem diferentes mecanismos trabalhando para resolver o mesmo problema geral e, portanto, não há uma linha fixa e rígida que indique onde um mecanismo termina e outro começa. Em suma, os limites das camadas tornam-se suaves (e fáceis de mover) quando as camadas são implementadas em software em vez de hardware. Isso está se tornando cada vez mais a norma.

Perspectiva mais ampla

Para continuar lendo sobre a cloudificação da Internet, consulte Perspectiva: Big Data e Análise .

Para saber mais sobre o B4, recomendamos: B4: Experiência com uma WAN definida por software implantada globalmente , agosto de 2013.

Para uma visão mais abrangente do controle de congestionamento, incluindo alguns dos desenvolvimentos mais recentes relacionados ao TCP, consulte nosso livro complementar: Controle de Congestionamento TCP: Uma Abordagem de Sistemas .

Anterior

Próximo