Roteadores: Arquitetura, Buffers, Políticas de Enfileiramento

Diego Passos

1 Roteadores: Arquitetura

Conforme discutido na aula anterior, há duas funções básicas em um roteador na Internet: o roteamento e o encaminhamento. O roteamento, que será estudado em detalhes nas últimas aulas deste capítulo, é implementado através da execução de protocolos e algoritmos de roteamento. Já o encaminhamento se preocupa com a movimentação de datagramas entre as portas de entrada e saída de um roteador, incluindo a consulta à tabela de roteamento.

De uma forma geral, a arquitetura de um roteador pode ser dividida em dois **planos**: o *plano de controle* e o *plano de dados* (este último também chamado de *plano de encaminhamento de dados*. Em termos de *hardware*, o plano de controle contém o processador do roteador, responsável pela execução de *softwares* de gerência, configuração e dos protocolos de roteamento. O plano de controle, portanto, é implementado, em grande parte, em *software*. Já o plano de dados é basicamente composto pelas portas de entrada e saída, e por uma **malha de comutação** — *i.e.*, uma espécia de rede de comunicação interna ao roteador que permite a transferência de datagramas das portas de entrada para as portas de saída.

Neste ponto, é importante deixar claro a distinção entre portas de entrada e de saída. Boa parte das tecnologias de camada de enlace são **duplex**, *i.e.*, permitem comunicação bidirecional. Por tanto, uma interface de rede geralmente é tanto uma porta de entrada, quanto uma porta de saída. Na aula de hoje, no entanto, trataremos estes dois elementos como sistemas separados.

1.1 Funções das Portas de Entrada

Portas — de entrada ou de saída — realizam a interface entre o roteador e os enlaces físicos. As portas, por tanto, implementam a camada física, "colocando" bits nos enlaces e interpretando os sinais recebidos pelo meio físico de volta à forma de bits. A camada de enlace também é comumente implementada nestes dispositivos de *hardware*.

Além destas funcionalidades de baixo nível, as portas de entrada ainda dão início ao processo de **comutação** — parte importante do encaminhamento de datagramas. Quando um datagrama é recebido por um porta de entrada, já no nível da camada de rede, esta porta realiza a busca na tabela de roteamento com o objetivo de identificar a porta de saída adequada. Uma vez encontrada a porta de saída correta, a porta de entrada delega a tarefa de comutação para a malha de comutação (discutida em mais detalhes a seguir).

Para roteadores de grande porte é importante que este processo de comutação ocorra na chamada **velocidade de linha**. Isto é, idealmente, gostaríamos que a comutação não fosse mais demorada que o próprio processo de recebimento do pacote pelo meio físico. Do contrário, corre-se o risco de que pacotes cheguem pelo enlace mais rapidamente que a porta é capaz de comutá-los para a porta de saída correta. Neste caso, **os pacotes que entram precisam ser armazenados em um *buffer***.

De fato, nem sempre a comutação dos pacotes será tão ou mais rápida que a velocidade de linha — em geral, por limitação da malha de comutação, como discutido adiante. Logo, o ocorrência de enfileiramento nas portas de entrada não é rara. Por outro lado, é importante esclarecer que **esta não é a fila à qual nos referimos várias vezes até aqui neste curso**: outra fila mais comum e relevante será a fila encontrada nas portas de saída, discutidas mais à frente nesta aula. **Não obstante, note que esta fila também introduz atrasos no encaminhamento dos pacotes e, embora raro, pode levar descarte de pacotes.**

1.2 A Malha de Comutação

O objetivo da malha de comutação é transportar datagramas das portas de entrada para as portas de saída do roteador. É importante repetir: a escolha da porta de saída não é realizada pela malha de comutação, esta apenas realiza o transporte para a porta de saída informada.

A malha de comunicação, portanto, pode ser compreendida como um canal de comunicação interno do roteador, permitindo a transferência dos pacotes entre portas. Como todo canal de comunicação,

a malha de comutação possui uma capacidade, mais comumente chamada de **taxa de comutação**. Em outras palavras, a taxa de comutação diz o quão rapidamente pacotes são comutados de portas de entrada para portas de saída através da malha. Esta taxa é comumente dada em pacotes por segundo, mas pode ser especificada também em bits por segundo ou bytes por segundo. É também comum que a taxa de comutação seja dada em um múltiplo da taxa de transmissão dos enlaces. Isso é interessante porque, idealmente, com n portas entrada de R b/s, gostaríamos de ter uma malha capaz de comutar $n \times R$ b/s, de forma a suportar o tráfego de pico do roteador.

Existem diversas formas de se implementar uma malha de comutação, algumas mais eficientes, outras nem tanto. Em roteadores encontrados no mercado, é possível encontrar todas estas alternativas implementadas na prática. A opção por um ou outro tipo depende, em grande parte, de um compromisso entre custo e desempenho esperado. Os três tipos básicos de malha de comutação são:

1. **Por memória**. Qualquer computador de propósito geral pode atuar como um roteador utilizando este tipo de malha de comutação. A "malha", neste caso, é a própria memória principal do computador. Quando um pacote chega a uma porta de entrada, ele é enviado através de barramentos do sistema para a memória principal. Uma vez armazenado em memória, o pacote é comutado para a porta de saída passando novamente pelos barramentos do sistema. Em suma, a comutação, aqui, é basicamente uma sequência de operações de entrada e saída tradicional em qualquer computador de propósito geral. A taxa de comutação, portanto, é limitada pela velocidade de acesso à memória e pelo uso dos barramentos, geralmente compartilhados com outros dispositivos do sistema. Além disso, deve-se notar que o pacote precisa ser transmitido pelo barramento duas vezes: da porta de entrada para a memória e da memória para a porta de entrada.

2. **Por barramento**. Neste caso, há um barramento dedicado a interligar as portas de entrada e saída do roteador. Quando um pacote chega a uma porta de entrada, ele enviado através do barramento para a porta de saída. Em relação à comutação por memória, esta solução evita o uso duplicado do barramento, além de não estar limitada pelo desempenho de leitura/escrita na memória principal. Entretanto, todos os datagramas e portas compartilham um mesmo barramento, o que pode limitar o desempenho e causar **contenção** — *i.e.*, a possibilidade de duas ou mais portas de entrada tentarem utilizar o barramento simultaneamente, fazendo com que alguma(s) precise(m) esperar sua vez.

3. **Por rede de interconexão**. Estas malhas de comutação apresentam topologias mais complexas, permitindo, em alguns casos, a comutação de múltiplos pacotes simultaneamente (entre pares de portas diferentes). Isso aumenta a taxa de comutação máxima e reduz a probabilidade de contenção. Por outro lado, a complexidade de implementação destas redes é maior, resultando em *hardware* mais caro.

As malhas de comutação por memória são usadas normalmente em roteadores antigos ou de uso doméstico (por conta de seu custo mais baixo). As malhas por barramentos ou por redes de interconexão, por sua vez, são usadas em modelos mais sofisticados, dos quais se espera desempenho mais alto.

1.3 Funções das Portas de Saída

As portas de saída realizam o processo inverso das portas de entrada. Elas, portanto, também implementam as camadas de enlace e física. Do ponto de vista da camada de rede, as portas de saída tem como principal funcionalidade **o enfileiramento de pacotes**.

Enfileiramento pode ocorrer nas portas de saída por conta de um descasamento da sua capacidade com a taxa de chegada de pacotes a ela. O caso mais comum ocorre quando, em um certo intervalo, múltiplas portas de entrada do roteador comutam vários pacotes para uma mesma porta de saída. Lembre-se que a taxa de comutação da malha de comutação é, geralmente, bem maior que a capacidade de cada uma das portas isoladamente. Isso significa que pacotes podem ser comutados para a porta de saída mais rapidamente que a sua capacidade de escoá-los pelo enlace físico.

Repare que este tipo de enfileiramento — na porta de saída — é bem mais comum que o enfileiramento na porta de entrada. Isso se deve, justamente, à capacidade da malha de comutação. Quando esta é grande em relação às capacidades dos enlaces — o que é a situação normal — a probabilidade de que pacotes fiquem enfileirados na porta de entrada é baixa. Por outro lado, neste caso, basta que múltiplas interfaces recebam uma alta carga de pacotes direcionados por uma única interface de saída para que o enfileiramento acabe ocorrendo na porta de saída.

A ocorrência de enfileiramento nas portas de saídas dos roteadores tem como consequências o aumento do atraso e a da possibilidade de descarte de pacotes. Como já discutido nesta disciplina,

o enfileiramento tem grande importância no desempenho da Internet — e em redes de comutação de pacotes em geral. Portanto, o gerenciamento desta fila é uma tarefa com consequências potencialmente relevantes para o funcionamento da rede como um todo. Por este motivo, mais à frente nesta aula discutiremos dois aspectos relevantes deste gerenciamento: as políticas de escalonamento e descarte de pacotes.

Uma pergunta importante no projeto e na configuração de um roteador é: **o quanto de memória deve-se dedicar ao *buffer* que armazenará a fila de pacotes?** A resposta para esta pergunta, infelizmente, não é trivial.

À medida que a tecnologia utiliza para a produção de memórias RAM teve seu custo reduzido, muitos fabricantes passaram a introduzir *buffers* cada vez maiores em seus roteadores. O raciocínio simplório era o de que se um *buffer* pequeno é bom, um *buffer* maior deve ser melhor. De fato, *buffers* muito pequenos podem prejudicar o funcionamento do TCP que normalmente gera tráfego em rajadas. Se o *buffer* é pequeno demais — em particular, se ele é menor que o tamanho da rajada do TCP — segmentos serão perdidos ainda que a rede não esteja propriamente congestionada.

No entanto, rapidamente notou-se que *buffers* excessivamente grandes também eram prejudiciais. Em particular, como o TCP tradicionalmente infere congestionamento pela perda de segmentos, quanto maior o *buffer*, mais tempo um transmissor demorará para perceber a ocorrência do congestionamento e, portanto, reduzir a sua taxa de transmissão. Isso faz com que o desempenho do TCP varie muito ao longo do tempo. Além disso, *buffers* maiores também acabam levando a atrasos maiores de enfileiramento. Esse atraso alto causado por *buffers* excessivamente longos recebe o nome de *bufferbloat*.

Existe, portanto, um tamanho ideal para o *buffer* de uma porta de saída de um roteador. Infelizmente, este valor ideal depende de uma série de fatores, incluindo o número de fluxos que passam pelo roteador, a capacidade do enlace de saída e o RTT envolvido nas conexões TCP que utilizam aquele *buffer*.

A RFC 3439, por exemplo, sugere que o *buffer* de uma interface de saída deve ser suficiente para armazenar "250 ms de dados". Queremos dizer com isso que, segundo esta recomendação, o *buffer* deve ter capacidade para armazenar o equivalente a 250 ms de dados transmitidos pela respectiva interface de saída. Por exemplo, se a interface de saída tem capacidade de 1 Gb/s, a recomendação sugere a utilização de um *buffer* de 250 Mb. Este valor de 250 ms foi escolhido por ser um "RTT típico da Internet". De forma mais geral, portanto, a recomendação é que o tamanho do *buffer* seja $C \times RTT$, onde C é a capacidade do enlace e RTT é o "RTT típico" dos fluxos que passam pelo enlace.

Pesquisas mais recentes, no entanto, mostram que este tamanho de *buffer* pode ser excessivo, resultando no tal *bufferbloat* citado anteriormente. Uma recomendação mais recente cita a seguinte equação para cálculo do tamanho do *buffer*:

$$S = \frac{C \times RTT}{\sqrt{N}}, \tag{1}$$

onde S é o tamanho do *buffer* e N é o número de fluxos que passam pela interface. Obviamente, na Internet, o número de fluxos que passam por uma dada interface varia no tempo, assim como variam os RTTs de cada conexão. A ideia aqui, portanto, é utilizar valores típicos.

Repare que nesta nova recomendação o tamanho do *buffer* **diminui com o aumento do número de fluxos**, o que é contra-intuitivo. A explicação para isso está na maneira pela qual o TCP age sobre sua janela de congestionamento. Como um transmissor TCP não tem ideia do número de outros transmissores ativos simultaneamente, sua manipulação da janela não é alterada com a mudança no número de fluxos. Mas do ponto de vista da rede, quanto maior o número de fluxos TCP simultâneos, mais abruptas são as mudanças na carga da rede. Logo, com mais fluxos TCP simultâneos queremos um *buffer* menor para alertar os transmissores mais rapidamente sobre condições de congestionamento.

1.4 Head-of-Line Blocking

Considere novamente o enfileiramento que pode ocorrer **nas portas de entrada**. Este enfileiramento normalmente ocorre porque a malha de comutação está ocupada naquele momento, fazendo com que o processo de comutação tenha que esperar em uma certa porta de entrada.

Suponha que um roteador utilize uma malha de comutação baseada em uma rede de interconexão que permita, em certos casos, a comutação de múltiplos pacotes simultaneamente. Assuma que, em certo momento, duas portas de entrada, `E1` e `E2`, querem comutar pacotes para uma mesma porta de saída — chamaremos esta porta de saída de porta `S1`. Por mais sofisticada que seja a malha de comutação, esta situação provavelmente gerará uma contenção: apenas uma das duas portas de

entrada poderá ser atendida de vez, já que ambas querem comutar seus pacotes para a mesma porta de saída.

Assuma que, de acordo com algum critério de resolução de contenção, a porta `E1` ganha o direito de cumutar seu pacote primeiro, deixando a porta `E2` em espera pela sua oportunidade. Isso gerará um certo nível de enfileiramento em `E2`. Em particular, é possível que um segundo pacote chegue a `E2` e precise, também, ser enfileirado.

Repare que este segundo pacote enfileirado em `E2` pode ser destinado a outra porta de saída — digamos, `S2` — para a qual seria possível realizar a comutação imediatamente. No entanto, como este pacote está atrás de um pacote atualmente impedido de ser comutado, este também ficará bloqueado na fila da porta `E2`.

Esta situação, de um pacote que, a princípio, poderia ser comutado ficar bloqueado por estar enfileirado atrás de um pacote que não pode ser comutado agora, é chamada de *Head-of-Line Blocking*.

2 Políticas de Enfileiramento

Discutiremos agora a gerência das filas dos roteadores (em particular, das filas das portas de saída, embora os mesmos algoritmos possam ser usados para a gerência das filas das portas de entrada). A esta gerência dá-se o nome de **política de enfileiramento**. Estas políticas são também chamadas de **disciplinas de enfileiramento**.

Há aspectos básicos que devem ser controlados no enfileiramento: a ordem em que os pacotes são transmitidos, e quando e quais pacotes devem ser descartados. Há, portanto, as políticas de escalonamento de pacotes (qual pacote transmitir a seguir) e as políticas de descarte.

2.1 Políticas de Escalonamento

Até aqui, nesta disciplina, sempre assumimos implicitamente uma política de descarte do tipo FIFO (*First-In First-Out*). Em outras palavras, assumimos que os pacotes entram no final da fila e sempre o primeiro pacote é escolhido para a próxima transmissão.

A política FIFO é, de fato, a mais comum e popular na Internet. Ela é simples e tende a dar oportunidades iguais a todos os pacotes, por respeitar a ordem na qual eles chegam ao roteador. Entretanto, em certas situações, o emprego de outras políticas de escalonamento pode trazer benefícios. Se temos, por exemplo, fluxos de melhor esforço (*e.g.*, fluxos TCP relativos a aplicações de transferência de arquivos) competindo com fluxos de tempo-real (*e.g.*, uma chamada VoIP), pode ser do interesse da rede que os pacotes da chamada VoIP "passem à frente" na fila.

Para implementar este tipo de funcionalidade, é preciso que o roteador seja capaz de **diferenciar** os pacotes de acordo com seus fluxos ou, ao menos, em classes de importância. Neste caso, antes de chegar à fila propriamente dita, o pacote passa por um elemento chamado de **classificador**. Como um classificador distingue a classe a qual pertence um pacote está além do escopo desta aula (na verdade, isso será visto apenas em Redes II). O fato é que uma vez classificados, políticas de escalonamento cientes de prioridades podem ser utilizadas.

O exemplo mais simples de uma política deste tipo é a *Priority Queueing*. Esta política aloca *buffers* separados para cada classe. Quando um pacote chega à fila, portanto, ele é colocado no final do *buffer* da sua classe específica. Cada classe possui uma prioridade e, quando o próximo pacote a ser transmitido deve ser escolhido, a *priority queueing* percorre os *buffers* da classe de maior prioridade para a de menor prioridade: se houver um pacote no *buffer* da classe de maior prioridade, este é escolhido para transmissão; caso contrário, consulta-se o *buffer* da próxima classe com a maior prioridade entre as restantes.

Nota-se, portanto, que, enquanto houver pacotes nos *buffers* das classes de maior prioridade, as classes de menor prioridade não serão servidas. Isso efetivamente atinge o efeito desejado de deixar certos pacotes "furarem fila". Por outro lado, pacotes de mais baixa prioridade podem ser preteridos indefinidamente, uma situação chamada de *starvation* (esfomeação ou inanição em português).

O *starvation* pode ser combatido com outra política de escalonamento: a *round-robin*. Assim como na política *priority queueing*, pacotes são divididos em classes e cada classe possui seu próprio *buffer*. Quando o próximo pacote a ser transmitido precisa ser escolhido, a política *Round-Robin* percorre as filas em ordem, **a partir da fila logo após aquela da qual o último pacote transmitido foi tirado**. Em outras palavras, a cada nova oportunidade de transmissão, a política *round-robin* dá chance para uma nova classe. Isso significa que todas as classes receberão serviço, independentemente do que ocorre nas demais.

Embora a política *round-robin* não gere *starvation*, ela também não fornece nenhum tipo de prioridade a uma classe sobre outra — ao contrário, apenas ela garante que os recursos dedicados a

uma classe não interferirão com os recursos das demais classes, seja espaço em *buffer*, seja tempo de transmissão. Uma política que consegue, simultaneamente, garantir priorização de classes e evitar *starvation* é a *Weighted-Fair Queueing* (ou WFQ). A WFQ atribui prioridades numéricas que são proporcionais à fração do tempo de utilização do enlace que será dedicada a cada classe. Assim, classes de maior prioridade terão acesso a uma fração maior do tempo de uso do enlace. Por outro lado, classes de prioridade mais baixa ainda receberão *alguma* fração não-nula de tempo para utilizar o enlace, independentemente da ocupação dos *buffers* das classes de mais alta prioridade. A política WFQ será estudada em mais detalhes em Redes II (no contexto das aplicações multimídia).

2.2 Políticas de Descarte de Pacotes

A política de descarte de pacotes mais simples e comum é chamada de *Drop-tail*. Quando o *buffer* está cheio e um novo pacote chega, a política *Drop-tail* simplesmente descarta o pacote recém-recebido.

Embora esta abordagem seja intuitiva, a *Drop-tail* pode resultar em alguns problemas. Um destes problemas é a possibilidade de **sincronização de fluxos**. Suponha que dois *hosts* conectados a um mesmo roteador originam um fluxo TCP cada um, compartilhando um enlace de saída — e, portanto, sua fila. Como já discutido anteriormente, o TCP tem por característica a geração de tráfego em rajada. Suponha que um dos dois *hosts* realiza a transmissão de uma rajada de pacotes do tamanho do *buffer* disponível na porta de saída do roteador. Em seguida, o outro *host* transmite a sua rajada de pacotes. Supondo que o enlace de saída do roteador seja o gargalo, quando os pacotes da rajada gerada pelo segundo *host* chegam ao *buffer*, este já está totalmente ocupado por pacotes da rajada do primeiro *host*. Logo, em uma política do tipo *Drop-tail* os pacotes da rajada do segundo *host* poderiam ser todos descartados em sequência. Em última análise, este comportamento causa um compartilhamento injusto dos recursos da rede.

Uma política de descarte alternativa, porém ainda bastante simples, é a *Drop-head*. Como o nome sugere, quando um novo pacote chega ao *buffer* e este se encontra cheio, descarta-se o primeiro pacote da fila — ou seja, o pacote que está a mais tempo enfileirado. Uma análise superficial da política *Drop-head* pode sugerir que ela é injusta. Afinal, estamos descartando um pacote que está no início da fila e, portanto, muito próximo a ser transmitido.

No entanto, a motivação para esta decisão se torna clara quando consideramos fluxos TCP. Como já estudado nesta disciplina, o TCP infere congestionamento através da detecção de segmentos perdidos. Esta detecção, por sua vez, acontece pelo recebimento de *acks* duplicados sucessivos ou pelo estouro de temporizador. Em ambos os casos, as detecções de perda do TCP sempre são realizadas sobre os segmentos mais antigos atualmente em trânsito. Voltando à política *Drop-head*, em uma situação de congestionamento, se temos a opção entre descartar um dentre potencialmente vários segmentos TCP, pode ser interessante descartar o segmento mais antigo, já que os eventos que indicam perda de pacote para o TCP tenderão a ocorrer em um futuro mais próximo para o segmento que foi transmitido há mais tempo que para um segmento mais recente. Colocando de outra forma, se não descartarmos o segmento que está no início da fila — depois de ter sofrido um alto atraso de enfileiramento no *buffer* do nosso roteador — corremos o "risco" que ele chegue ao seu destinatário final e que o *ack* correspondente volte ao transmissor antes de eventos como *acks* duplicados ou estouro de temporizador ocorram e indiquem o congestionamento ao transmissor TCP. Desta forma, descartar o segmento no início do *buffer* pode ter o efeito desejável de avisar de maneira antecipada ao TCP sobre o congestionamento.

Note, no entanto, que o problema de sincronização de fluxos que afeta a política *Drop-tail* também pode se manifestar na política *Drop-head*, embora de forma invertida. Se repetirmos a análise do cenário com os dois *hosts* compartilhando um enlace de saída de um roteador e gerando tráfego em rajadas, concluiremos, novamente, que um dos *hosts* será consistentemente prejudicando, tendo seus pacotes descartados em benefício dos pacotes do outro fluxo. Desta vez, no entanto, por descartamos os pacotes do início da fila, o fluxo prejudicado será o do primeiro *host* a transmitir: à medida que os pacotes da rajada do segundo *host* chegam ao roteador e encontram a fila cheia, os pacotes que já estavam na fila, *i.e.*, os do primeiro *host*, serão descartados um a um.

A terceira e última política de descarte que estudaremos nesta disciplina é chamada de RED (do inglês *Random Early Detection* ou *Random Early Drop*). Esta política se diferencia das duas outras políticas estudadas nesta aula em vários aspectos. Talvez o aspecto mais marcante da política RED é o fato de que ela pode determinar o descarte de pacotes **mesmo se a fila não se encontra totalmente cheia**.

A política RED utiliza uma série de parâmetros para balizar seu funcionamento. De forma simplificada, estes parâmetros incluem:

1. **Mínimo**. Determina um limite inferior de ocupação da fila a partir do qual pacotes *podem* ser descartados.

2. **Máximo**. Determina um limite superior de ocupação da fila até o qual pacotes *podem* **não** ser descartados.

Quando um novo pacote chega ao *buffer*, a política RED compara a ocupação atual da fila. Se a fila está atualmente menor que o valor do parâmetro `Mínimo`, o novo pacote é sempre aceito. Se a fila está atualmente maior ou igual ao valor do parâmetro `Máximo`, o pacote recém-recebido é **sempre** descartado. Por outro lado, se a ocupação atual da fila se encontra entre os valores dos parâmetros `Mínimo` e `Máximo`, o pacote recém-recebido é descartado com uma probabilidade p que é proporcional a esta ocupação (a probabilidade cresce linearmente entre 0 e 1 à medida que a ocupação varia entre `Mínimo` e `Máximo`).

A motivação usada pelo RED para descartar pacotes antes do *buffer* estar completamente cheio é similar a utilizada para explicar os potenciais benefícios da política *Drop-head*: o objetivo a alertar antecipadamente as fontes de tráfego sobre a situação de congestionamento. Lembre-se que os descartes de pacotes são uma consequência de uma situação **prolongada** — e extrema — de congestionamento. Quando a ocupação do *buffer* de uma porta de saída é *representativa*, isso já sinaliza que existe congestionamento. Esta ocupação representativa é determinada no RED através do parâmetro `Mínimo`. Logo, a partir deste nível de enfileiramento, o RED começa a descartar pacotes — ainda que probabilisticamente — com o intuito de alertar as fontes que estas devem reduzir suas taxas de transmissão.

De fato, quando configurado de forma correta, o RED permite que o TCP convirja para a vazão máxima do caminho, mantendo atrasos relativamente baixos — em comparação ao obtido por outras disciplinas de enfileiramento. Um problema do RED é que nem sempre é fácil encontrar valores adequados para seus parâmetros de acordo com o cenário em questão. Isso dificulta uma adoção mais ampla desta técnica.

Um efeito colateral interessante do RED é a solução do problema do sincronismo. Repare que, embora o RED sempre descarte o pacote recém-recebido, ele não o faz apenas quando o *buffer* está cheio. Logo, de certa maneira, o pacote escolhido para descarte no RED pode ser considerado aleatório, ao contrário do que ocorre nas políticas *Drop-tail* e *Drop-head*. Esta natureza aleatória do RED faz com que ele não prejudique consistentemente um mesmo fluxo, distribuindo seus descartes de maneira homogênea entre os vários fluxos que compartilham o enlace.