Dynamic LLC Buffer Allocation for Networking with Dual Receive Ring

Filipe Pirola, Marcos A. M. Vieira, *Member, IEEE,* Luiz F. M. Vieira, *Member, IEEE.*

***Abstract*—Placas de rede (NICs) de até 800 Gbps [1] modernas escrevem pacotes recebidos diretamente na Last Level Cache (LLC) do processador por meio de DMA assistido por cache (Intel DDIO e equivalentes). Esse mecanismo cria um dilema na configuração do anel de recepção, conhecido como Leaky DMA: anéis grandes absorvem rajadas de tráfego, mas poluem a LLC com pacotes ainda não processados; anéis pequenos preservam a cache, mas descartam pacotes sob picos. Este trabalho propõe e avalia o DualRing, uma arquitetura de dois anéis de recepção assimétricos, um pool pequeno residente em cache para o caminho rápido e um pool grande em DRAM para absorver rajadas, implementada inteiramente sobre as APIs padrão do DPDK, sem qualquer modificação em driver, kernel ou firmware. A avaliação experimental, conduzida em hardware AMD EPYC 7452 com NICs ConnectX-5 de 100 GbE e gerador de tráfego T-Rex, mostra que o DualRing reduz a latência de cauda (p99) em 17% sob tráfego em rajada e praticamente elimina os outliers de latência de ordem de milissegundos observados no baseline (l2fwd) sob tráfego contínuo, sem degradar a vazão sustentada. Os resultados sustentam a viabilidade da abordagem e a hipótese de que o isolamento de um working set pequeno e residente em cache reduz a pressão sobre a LLC mesmo em processadores com cache de última geração de grande capacidade.**

Palavras-chave: Redes de Alto Desempenho, DPDK, Gerenciamento de Cache, DDIO, Anéis de Recepção.

I. Introdução

O cenário atual das redes de computadores tem sido marcado por um avanço acelerado na capacidade de transmissão de dados. Interfaces de rede modernas já alcançam velocidades de até 800 Gbps [1], impulsionando a demanda por sistemas capazes de processar volumes massivos de informação em tempo real. No entanto, essa evolução da infraestrutura de rede expõe uma disparidade crítica em relação ao desempenho dos subsistemas de memória. Enquanto a capacidade de rede e o número de núcleos de processamento cresceram exponencialmente a performance da memória não acompanhou o mesmo ritmo, Figura 1.

Esse descompasso cria desafios significativos na interação entre a Placa de Rede (NIC, do inglês Network Interface Card) e a Memória Cache de Último Nível (LLC, do inglês Last-Level Cache). A LLC, por possuir um espaço de armazenamento limitado, torna-se um recurso escasso diante do fluxo intenso de dados proveniente de redes de alta velocidade. A incapacidade da cache de acomodar eficientemente esse fluxo resulta em gargalos de desempenho, que se manifesta através do aumento da latência e do descarte de pacotes (*packet loss*) por contenção [2], [3], como representado na Figura 2.

Filipe Pirola, Marcos A. M. Vieira, and Luiz F. M. Vieira são do Departamento de Ciência da Computação, Universidade Federal de Minas Gerais (UFMG), Belo Horizonte, Brasil. (e-mail: {filipepirola, mmvieira, lfvieira}@dcc.ufmg.br)

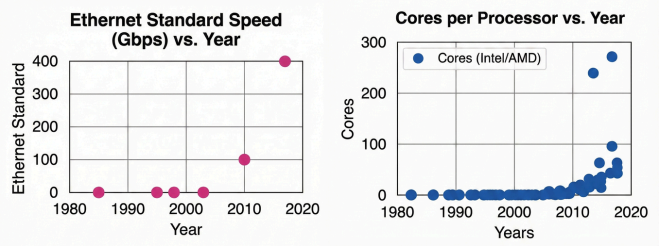

Fig. 1. Disparidade de Evolução: Enquanto a capacidade de rede e o número de núcleos crescem exponencialmente, a performance da memória estagnou. **Fonte:** Data from Karl Rupp / Creative Commons Attribution 4.0 International Public License

Dessa forma, o processamento de pacotes em software em alta velocidade depende de frameworks de kernel bypass como o DPDK (Data Plane Development Kit), que entregam os pacotes diretamente ao espaço de usuário por polling, eliminando interrupções e cópias do caminho tradicional do sistema operacional. Para reduzir ainda mais a latência, NICs modernas escrevem os pacotes recebidos diretamente na Last Level Cache (LLC) do processador, em vez da memória principal, usando tecnologias de DMA assistido por cache, Intel DDIO (Data Direct I/O) e mecanismos equivalentes em outras arquiteturas. Quando o pacote já está na cache no momento em que o núcleo de processamento o acessa, evita-se um acesso à DRAM, da ordem de uma centena de nanossegundos.

Esse ganho, contudo, traz um efeito colateral conhecido na literatura como Leaky DMA. A NIC aloca os buffers de recepção (mbufs) a partir de um pool de memória, e o tamanho do anel de recepção determina o volume de dados que a placa pode depositar na LLC antes que o software os consuma. Surge então um compromisso fundamental:

1) **Anel grande:** comporta rajadas de tráfego sem descarte, pois há buffers disponíveis para acomodar picos; porém, o conjunto de pacotes pendentes pode exceder a fração da LLC reservada ao DMA, expulsando dados úteis e forçando acessos à DRAM no processamento subsequente.
2) **Anel pequeno:** mantém o footprint de DMA contido na cache, preservando o desempenho de acesso; porém, esgota seus buffers rapidamente sob rajadas, causando descarte de pacotes na própria NIC.

A hipótese central deste trabalho é que esse trade-off pode ser quebrado, e não apenas equilibrado, por meio de dois anéis de recepção assimétricos operando em conjunto: um anel pequeno e residente em cache para o caminho comum,

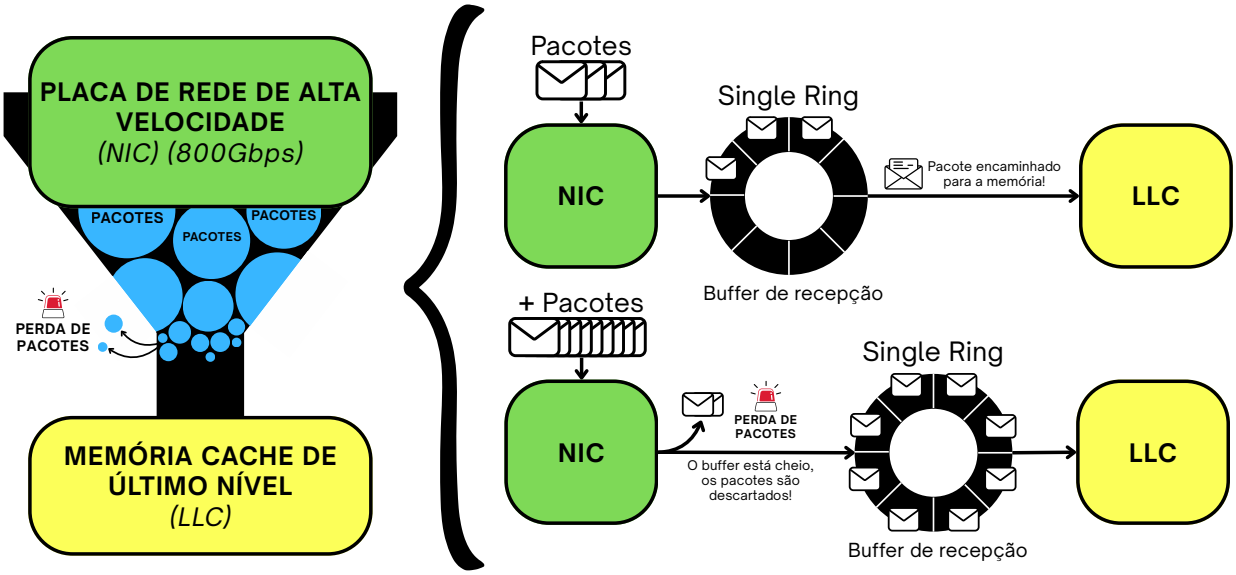

Fig. 2. Em cenários de alta transmissão de pacotes, a incapacidade da cache (LLC) de acomodar o fluxo da NIC resulta em contenção e perda de pacotes. O dilema do Leaky DMA no anel único.

e um anel grande em DRAM acionado sob demanda apenas durante rajadas. O objetivo é obter, simultaneamente, o baixo footprint de cache do anel pequeno e a capacidade de absorção de rajadas do anel grande, usando exclusivamente as APIs padrão do DPDK, sem modificar o driver da NIC, o kernel ou o firmware, o que distingue a proposta de soluções anteriores e a torna portável a qualquer PMD (Poll Mode Driver) compatível.

O restante deste artigo está organizado da seguinte forma: a Seção II aborda a fundamentação teórica necessária e a Seção III discute os trabalhos relacionados. A Seção VI discute a metodologia e os resultados experimentais são analisados na Seção VII. Por fim, a Seção VIII apresenta as conclusões e perspectivas para trabalhos futuros.

II. Fundamentação Teórica

Esta seção apresenta os conceitos fundamentais necessários para a compreensão do trabalho, abordando o processamento de pacotes de alto desempenho e o gerenciamento de recursos de hardware.

A. Data Plane Development Kit (DPDK)

O *Data Plane Development Kit* (DPDK) [4] consiste em um conjunto de bibliotecas e drivers executados em espaço de usuário (*user-space*), projetados para acelerar o processamento de pacotes em diversas arquiteturas de CPU. Originalmente criado pela Intel e atualmente um projeto da *Linux Foundation*, o DPDK tornou-se essencial para viabilizar o uso de processadores de propósito geral em ambientes de rede de alto desempenho.

Diferente da pilha de rede padrão do sistema operacional, que depende de interrupções e trocas de contexto custosas, o DPDK utiliza técnicas de *Kernel Bypass* e *Polling Mode Drivers* (PMD). Isso permite que as aplicações acessem diretamente as placas de rede e filas de hardware, eliminando o overhead do kernel do Linux no caminho de dados.

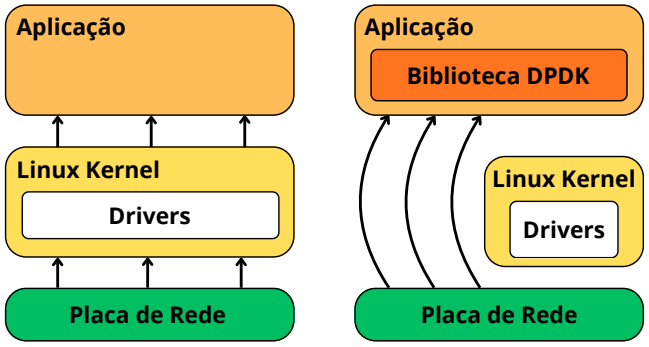

Fig. 3. Comparação entre a pilha de rede tradicional do Linux e a arquitetura DPDK.

B. Alocação de Memória e Interação NIC-CPU Clássica

A interação eficiente entre o *software* e as placas de interface de rede (NICs) modernas baseia-se no modelo produtor-consumidor implementado através de filas circulares de memória, denominadas anéis ou *rings*. Em arquiteturas de alto desempenho, como as utilizadas em NFV (*Network Function Virtualization*), cada núcleo de processamento (*core*) é tipicamente associado a um anel de recepção (Rx) e um de transmissão (Tx) exclusivos, modelo conhecido como *privRing* [2].

No caminho de recepção (Rx), o funcionamento clássico exige que o *software* pré-aloque *buffers* de memória na RAM e preencha os descritores do anel com ponteiros para esses *buffers* vazios. O tamanho de cada *buffer* deve ser suficiente para armazenar um pacote da Unidade Máxima de Transmissão (MTU), tipicamente 1500 bytes para Ethernet padrão. O ciclo de processamento ocorre da seguinte forma:

1) **Produção pela NIC:** Quando um pacote chega, a NIC consome um descritor apontando para um *buffer* vazio (indicado pelo índice *head* do anel), escreve o pacote

via DMA (*Direct Memory Access*) e atualiza o estado do anel.

2) **Consumo pela CPU:** O *software*, operando em *polling*, detecta a chegada do pacote, processa os dados e, crucialmente, deve repor imediatamente o descritor usado com um novo *buffer* vazio para manter o anel operacional.

Este modelo cria um entrelaçamento funcional onde o anel de recepção atua simultaneamente como uma fila de alocação de memória (onde a CPU produz *buffers* vazios) e uma fila de entrega de pacotes (onde a NIC produz *buffers* cheios) [3].

Para suportar as taxas de transferência atuais, tecnologias como o Intel DDIO (*Data Direct I/O*) permitem que a NIC realize operações de DMA diretamente na memória *cache* de último nível (LLC – L3) do processador, evitando a alta latência da memória principal (DRAM). O DDIO aloca linhas de *cache* para os pacotes recebidos, permitindo que a CPU acesse os dados muito mais rapidamente.

No entanto, a eficácia do DDIO é limitada pelo tamanho do "Conjunto de Trabalho de E/S" (*I/O Working Set*). Este conjunto é definido como a área total de memória referenciada pelos anéis de recepção ativos. Como os descritores são acessados ciclicamente, todos os *buffers* no anel devem ser preenchidos ou lidos antes de serem reutilizados. O tamanho desse conjunto (W_{set}) pode ser estimado pela Equação 1:

$$W_{set} = N_{\text{anéis}} \times T_{\text{anel}} \times \text{MTU} \quad (1)$$

Onde $N_{\text{anéis}}$ é o número de filas ativas e T_{anel} é o tamanho da fila. Com o aumento da velocidade das redes, o tamanho padrão dos anéis cresceu para 1024 entradas ou mais para absorver rajadas de tráfego (*micro-bursts*) sem descarte de pacotes [2]. Em um sistema *multicore*, a soma desses *buffers* frequentemente excede a capacidade da LLC. Quando W_{set} ultrapassa o tamanho da LLC (ou a fatia alocada para o DDIO), ocorre o fenômeno de *"Leaky DMA"*: novos pacotes recebidos pela NIC expulsam da *cache* dados úteis ou pacotes ainda não processados, forçando o sistema a recorrer à memória RAM, o que degrada severamente a vazão e aumenta a latência.

III. Trabalhos Relacionados

O gerenciamento eficiente da memória cache em redes de alta velocidade tem sido foco de pesquisas recentes, visto que o aumento da largura de banda das interfaces de rede (NICs) ultrapassou a evolução da capacidade das memórias cache de último nível (LLC). A seguir, discutimos os trabalhos mais relevantes que abordam o problema do *I/O working set* e como eles se contrastam com a nossa proposta.

A. ShRing

O *ShRing* [2] é um sistema projetado para mitigar o problema do tamanho excessivo do *working set* de I/O em ambientes multi-core. Em arquiteturas tradicionais, cada núcleo possui seu próprio anel de recepção (Rx Ring) de tamanho padrão (e.g., 1024 entradas) para absorver rajadas de pacotes. Contudo, a soma de todos os buffers apontados por esses anéis frequentemente excede a capacidade da LLC, forçando o tráfego de rede a ser servido pela memória principal, o que degrada a vazão e a latência devido à ineficiência do DDIO (*Data Direct I/O*).

O *ShRing* aborda esse problema compartilhando um único anel de recepção entre múltiplos núcleos (por exemplo, 8 núcleos). Isso reduz drasticamente a pegada de memória, permitindo que o *working set* caiba na LLC. Para viabilizar isso, o sistema utiliza anéis de completude (*Completion Rings*) por núcleo para notificar a chegada de pacotes sem a necessidade de sincronização no caminho de leitura.

Embora o *ShRing* reduza o consumo de cache, ele introduz um custo de sincronização de software (locks ou operações atômicas) quando os núcleos precisam devolver buffers livres ao anel compartilhado. Além disso, o *ShRing* sofre com cenários de desbalanceamento de carga: se um núcleo estiver sobrecarregado, ele pode monopolizar as entradas do anel compartilhado, causando descarte de pacotes para núcleos ociosos. Nossa abordagem se diferencia ao utilizar uma estratégia de *Dual Receive Ring* com alocação dinâmica, visando otimizar o uso da LLC sem sofrer com o bloqueio imposto pelo compartilhamento estático de um único anel e focando na gestão inteligente dos buffers via software (DPDK).

B. RxBisect

Em um trabalho subsequente, o *RxBisect* [3], que identifica a causa raiz das limitações do *ShRing* no "entrelaçamento" de duas funções distintas nos anéis de recepção tradicionais: a alocação de memória (produção de buffers vazios pelo núcleo) e a entrega de pacotes (consumo de buffers cheios pelo núcleo).

O *RxBisect* propõe uma nova interface entre CPU e NIC que desacopla essas funções, dividindo o anel tradicional em dois: (1) Anéis de Alocação (*Ax rings*), que fornecem buffers vazios; e (2) Anéis de Recepção Bisseccionados (*Bx rings*), que recebem os pacotes. Essa separação permite que a NIC consuma buffers de qualquer anel de alocação (*Ax*) para preencher qualquer anel de recepção (*Bx*), criando efetivamente um pool de buffers compartilhado gerenciado pelo hardware. Isso resolve o problema de desbalanceamento do *ShRing*, pois um núcleo sobrecarregado não impede que outros núcleos recebam pacotes, desde que haja buffers disponíveis globalmente.

O *RxBisect* propõe uma alteração arquitetural na interface da NIC ou depende de emulação via software que pode introduzir latência. Nossa proposta de alocação dinâmica de LLC atua inteiramente em nível de software sobre hardware *off-the-shelf* (usando DPDK), sem exigir modificações no hardware da NIC. Enquanto o *RxBisect* foca no desacoplamento estático das filas para resolver o *head-of-line blocking*, nosso trabalho explora a dinamicidade da alocação de recursos da cache para adaptar o sistema às variações de carga em tempo real, utilizando a estrutura de anéis duplos para gerenciar a pressão sobre a memória.

IV. DualRing: A Arquitetura

O DualRing organiza a recepção em torno de dois pools de memória DPDK (rte_mempool) com propósitos distintos,

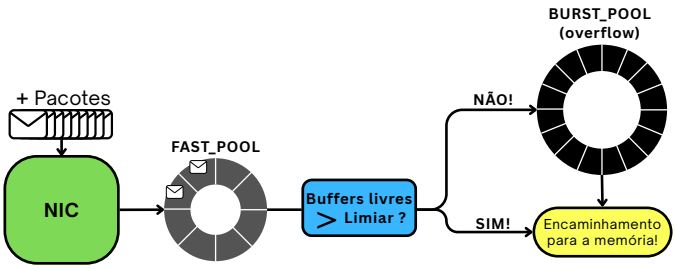

Fig. 4. Arquitetura DualRing. Em operação normal o pacote segue pelo fast_pool (caminho rápido); quando os buffers livres caem abaixo do limiar, o pacote transborda para o burst_pool em DRAM, liberando o buffer rápido imediatamente.

e um anel de transbordo (rte_ring) que conecta o caminho de exceção ao processamento diferido. A inspiração estrutural vem do modelo de quatro anéis do AF_XDP [5] (fill, completion, RX, TX), mas a realização é feita inteiramente com primitivas padrão do DPDK.

A. Componentes

1) **fast_pool:** Mempool pequeno (2.048 mbufs, aproximadamente 4,4 MiB) que alimenta a fila de recepção da NIC. É dimensionado para residir na fração de LLC associada ao caminho de DMA, de modo que os pacotes recém-recebidos e seus buffers permaneçam quentes na cache. A reciclagem rápida desses buffers mantém o working set da NIC pequeno.
2) **burst_pool:** Mempool grande (65.536 mbufs) alocado em DRAM, usado apenas como destino de transbordo durante rajadas.
3) **spill_watermark:** Limiar de disponibilidade do fast_pool que dispara o caminho de transbordo.

B. Polítca de operação (caminho de recepção)

A cada lote de pacotes recebido por rte_eth_rx_burst, o sistema verifica quantos buffers ainda estão disponíveis no fast_pool e decide entre dois caminhos, como visto na Figura 4:

Algorithm 1 Mecanismo de Encaminhamento com Caminho de Transbordo

se *fast_pool.disponível* $\geq$ *spill_watermark* **então**
    `// CAMINHO RÁPIDO`
1   encaminha o lote diretamente (swap de MAC + TX) drena pendências do burst_ring, se houver
fim
senão
    `// CAMINHO DE TRANSBORDO // rajada detectada`
2   (a) copia o conteúdo do mbuf rápido para um mbuf do burst_pool (DRAM)
3   (b) libera imediatamente o buffer rápido $\rightarrow$ mantém o pool quente e enxuto
4   (c) enfileira o mbuf de transbordo no burst_ring para TX diferido
fim

O ponto crucial é a etapa (b): ao liberar o buffer rápido assim que o pacote é copiado para a DRAM, o working set que a NIC mantém na cache permanece pequeno mesmo durante uma rajada, é isso que preserva o benefício de cache do caminho comum enquanto a rajada é absorvida pela DRAM. O custo dessa proteção é uma cópia de memória por pacote transbordado, paga apenas pelos pacotes que efetivamente caem no caminho de exceção.

V. Implementação

A implementação parte do exemplo l2fwd do DPDK 21.11 LTS, preservando integralmente sua estrutura de polling, inicialização de portas e laço de encaminhamento. Essa decisão é metodológica: derivar de um ponto de partida comprovadamente funcional garante que qualquer diferença de desempenho observada seja atribuível ao mecanismo DualRing, e não a divergências de versão, de configuração de PMD ou de inicialização. As adições ao l2fwd são a criação dos dois pools, do anel de transbordo, e a inserção da política da Seção IV no laço de recepção.

A implementação é parametrizável por linha de comando, o que permite instanciar diferentes configurações a partir do mesmo binário. Quatro sistemas foram avaliados:

TABLE I
Configurações dos Sistemas Avaliados

Sistema	Configuração	Papel na avaliação
l2fwd	privRing padrão do DPDK (anel de 1.024, pool de 8.192)	Baseline
DR small	anel pequeno (128), fast_pool 2.048, sem transbordo (`--disable-burst`)	Isola o efeito do pool pequeno
DR dual (sem work)	anel pequeno + burst_pool + transbordo ativo	DualRing completo, sem carga de trabalho sintética
DR dual	idem anterior + WorkPackage (4 acessos/pacote em 80 MiB)	DualRing com workload que força pressão de cache

O WorkPackage da última variante replica a carga sintética usada na avaliação do ShRing: a cada pacote, são feitos quatro acessos aleatórios a um buffer de 80 MiB. Como esse buffer excede a capacidade tipicamente reservada ao DDIO, ele força misses de LLC de forma controlada, simulando uma função de rede (NF) realista que toca dados além do cabeçalho do pacote. Sua finalidade é permitir comparação metodológica direta de ciclos por pacote com o estado da arte. O parâmetro spill_watermark foi calibrado em 1.400 mbufs: em regime estável há cerca de 1.024 buffers disponíveis, de modo que o caminho de transbordo não é acionado em tráfego contínuo e só dispara sob carga de rajada, exatamente o comportamento desejado para a fase de particionamento estático.

O código com a implementação completa pode ser encontrado no *Github*.

VI. METODOLOGIA

Esta seção detalha a infraestrutura e o procedimento experimental utilizados para avaliar a arquitetura proposta e os *baselines* de comparação.

A. Configuração Experimental

Os experimentos foram conduzidos na infraestrutura de pesquisa CloudLab [6]. O ambiente foi instanciado a partir de um perfil composto por dois nós *bare metal* do tipo **d6515**, conectados diretamente (*back-to-back*) por dois enlaces dedicados de 100 Gbps.

O nó designado como Dispositivo Sob Teste (DUT) é equipado com um processador **AMD EPYC 7452** (32 núcleos físicos, microarquitetura Zen 2), com 128 MiB de cache de último nível (LLC) particionada entre os *Core Complexes* (CCX), e 125 GiB de memória RAM. Para a comunicação de rede, cada nó dispõe de duas interfaces **NVIDIA/Mellanox ConnectX-5 Ex de 100 GbE**. As especificações detalhadas do hardware podem ser consultadas na documentação oficial da plataforma [7]. O segundo nó, de hardware idêntico, atuou exclusivamente como gerador de tráfego.

No que tange ao software, ambas as máquinas foram configuradas com o sistema operacional Ubuntu 20.04 LTS[1] e com a versão **21.11 LTS** do DPDK, compilada a partir do código-fonte para habilitar o *Poll Mode Driver* (PMD) `mlx5` das interfaces ConnectX-5 e o acesso direto às filas de hardware.

B. Implementações Avaliadas

A avaliação compara o *baseline* clássico contra as variantes da arquitetura proposta, todas descritas na Seção V e resumidas na Tabela I. Por serem instanciadas a partir do mesmo binário, derivado do exemplo `l2fwd` do DPDK, eventuais diferenças de desempenho são atribuíveis ao mecanismo DualRing, e não a divergências de versão, de configuração de PMD ou de inicialização. O DUT executou a aplicação de encaminhamento sobre 2 núcleos (`-l 0-1`) e 2 portas (`-p 0x3`); o nó gerador operou a ferramenta **T-Rex** em modo *stateless*.

C. Perfis de Tráfego

Foram injetados pacotes de **64 bytes**, o pior caso para o subsistema de memória e de processamento, por maximizar a taxa de pacotes por segundo, em dois perfis que representam extremos de carga operacional:

- **steady_64b:** fluxo contínuo a uma taxa moderada (sem saturação da capacidade de processamento), destinado a medir a latência em regime estável e a evidenciar o efeito da poluição de cache sobre a cauda de latência.
- **bursty_64b:** tráfego em rajadas periódicas atingindo 100 % da taxa de linha durante os picos, destinado a avaliar o comportamento do sistema e o descarte de pacotes sob saturação extrema.

Cada combinação de sistema e perfil foi executada por 60 segundos, repetida **5 vezes**, com intervalo de 10 segundos entre repetições, para caracterizar a variabilidade entre execuções.

[1] Verificar a versão exata utilizada nos nós provisionados.

D. Métricas de Desempenho

Para validar a hipótese de redução da pressão sobre a LLC, foram coletadas as seguintes métricas:

- **Latência fim-a-fim:** percentis p50 e p99 do tempo de ida e volta (RTT), calculados a partir dos histogramas agregados das 5 repetições, reportados pelo fluxo de latência dedicado do T-Rex.
- **Cauda de latência:** contagem de pacotes *outliers* com latência ≥ 1 ms, métrica direta da ocorrência de *stalls* de acesso à DRAM associados ao *Leaky DMA*.
- **Vazão e taxa de perda:** pacotes transmitidos, recebidos e descartados, com a taxa de perda resultante sob cada perfil.

Como instrumentação de apoio à análise de pressão de cache, a taxa de *LLC misses* foi monitorada no DUT via `perf_event_open` (utilizando o PMU `amd_l3`, específico da arquitetura Zen 2), em paralelo à execução dos experimentos.

VII. RESULTADOS

Para aferir a eficácia da abordagem DualRing em relação ao baseline de alocação estática (l2fwd), os sistemas foram submetidos aos perfis de tráfego delimitados, resultando em melhorias significativas no tratamento do fenômeno Leaky DMA.

A. Tráfego Contínuo e Eliminação de Outliers

Sob o tráfego contínuo (steady_64b), onde não há saturação da capacidade de processamento, a latência mediana (p50) revelou-se idêntica para todos os sistemas, cravando $10\mu s$. Isso comprova que a arquitetura DualRing não adiciona penalidades ou custos indiretos no caso comum de operação. A principal divergência manifesta-se na latência de cauda. No ambiente baseline, cujo tamanho do anel supera a parcela de cache reservada ao DDIO, parte dos buffers é fatalmente enviada à DRAM. Quando o núcleo acessa esses dados, a penalidade do acesso reflete-se em outliers na ordem de milissegundos: em cerca de 304 mil pacotes amostrados, o l2fwd registrou 4.049 ocorrências com latência ≥ 1 ms (chegando a picos de 10 ms). Em contrapartida, as variantes do DualRing reduziram essa contagem para apenas 1 ou 2 pacotes. Tais achados demonstram de forma prática que restringir o working set de recepção à LLC através do fast_pool elimina os stalls de memória.

B. Tráfego em Rajada e Redução da Latência de Cauda

Durante os testes de rajada (bursty_64b), a injeção excessiva de pacotes superou massivamente a capacidade computacional da CPU, levando todos os sistemas avaliados a uma taxa de perda saturada ao redor de 76%. Como o gargalo recai sobre o tempo de processamento per-packet, o impacto da cópia extra do caminho de transbordo não degradou substancialmente a vazão sustentada, que manteve diferenças estatísticas ínfimas ($\sim 1\%$) atribuíveis à variabilidade entre execuções.

Apesar da alta taxa de descarte inevitável neste cenário estressor, o DualRing destacou-se na preservação da latência.

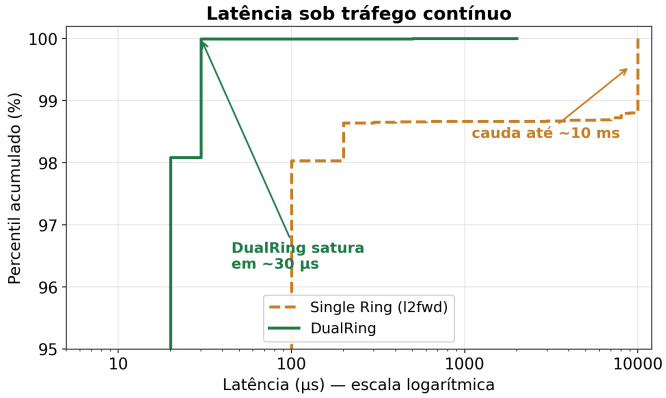

Fig. 5. Distribuição acumulada (CDF) de latência sob tráfego contínuo, com eixo vertical ampliado na faixa de 95% a 100% para evidenciar a cauda. As curvas DualRing atingem 100% por volta de 30 µs, enquanto o baseline l2fwd arrasta uma cauda longa até a ordem de 10 ms.

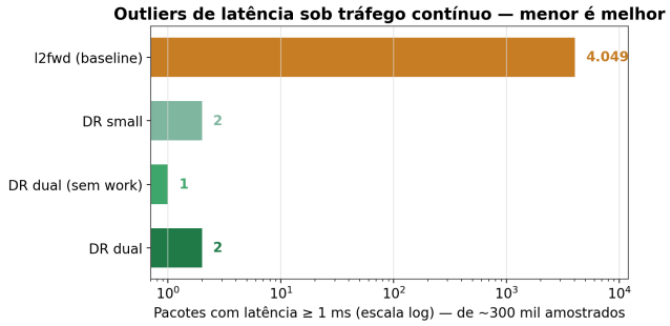

Fig. 6. Número de pacotes com latência igual ou superior a 1 ms (escala logarítmica). A diferença de três ordens de grandeza entre o baseline e as variantes DualRing é a evidência mais direta do efeito Leaky DMA: o pool rápido residente em cache elimina os stalls de acesso à DRAM responsáveis pelos outliers.

O baseline l2fwd apresentou uma latência p99 (percentil 99) de $600\mu s$, enquanto todas as implementações com DualRing consolidaram essa mesma métrica em $500\mu s$. Essa redução consistente de 17% corrobora a hipótese de que aliviar a pressão de E/S na LLC garante uma previsibilidade muito maior no tempo de processamento dos pacotes que conseguem ser absorvidos.

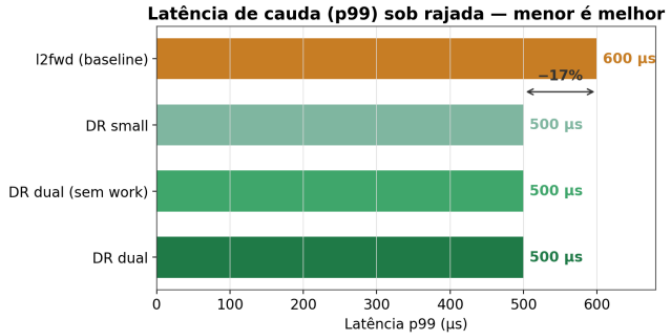

Fig. 7. Latência de cauda (p99) sob tráfego em rajada. Todas as variantes DualRing reduzem o p99 de 600 µs para 500 µs em relação ao baseline.

VIII. Conclusão

O avanço da velocidade das interfaces de rede expôs a hierarquia de memória, especificamente a capacidade da cache LLC, como um gargalo primário no processamento de pacotes. Este trabalho investigou a ineficiência inerente ao modelo clássico de anéis de recepção estáticos, onde o superdimensionamento necessário para absorver rajadas de tráfego resulta em um "Conjunto de Trabalho de E/S" que excede a capacidade do DDIO, degradando a performance do sistema devido ao fenômeno de *Leaky DMA*.

Para solucionar esse impasse, este artigo apresentou e validou experimentalmente o DualRing, uma arquitetura de anéis duplos de recepção assimétricos desenvolvida inteiramente em software sobre o DPDK, sem exigir modificações nos drivers da interface de rede, no kernel do sistema operacional ou no hardware subjacente.

Os testes conduzidos em ambiente CloudLab sob arquitetura AMD EPYC indicaram que o isolamento de um pool pequeno e contido na cache LLC consegue praticamente zerar a ocorrência de outliers de latência (na casa dos milissegundos) sob condições de tráfego contínuo, além de mitigar a latência de cauda (p99) em 17% frente aos modelos tradicionais durante cenários de rajadas severas. Tais resultados provam que a eficiência de cache, independentemente do volume total de LLC disponível no processador, depende diretamente de um working set de recepção enxuto e estritamente controlado.

Os resultados sugerem que nossa abordagem oferece um compromisso favorável em relação ao estado da arte. Diferente do *ShRing*, que impõe custos de sincronização de software [2], e do *RxBisect*, que exige alterações de hardware ou emulação [3], nossa proposta alcançou benefícios similares de isolamento de recursos utilizando *hardware commodity* e modificações puramente em *software*.

Como diretrizes para trabalhos futuros, destaca-se a implementação do controle adaptativo do limiar de transbordo (spill_watermark), onde as decisões de alocação passariam a ser regidas pela taxa de misses da LLC medida em tempo real através dos contadores de desempenho de hardware (perf events). Adicionalmente, investigar cargas de trabalho intermediárias e expandir a arquitetura assimétrica para a rota de transmissão (TX) representam passos valiosos para o refinamento holístico do tráfego de rede na camada de aplicação. Por fim, comparar experimentalmente a nossa implementação com o estado da arte.

References

[1] H. Yu, D. Patel, W. Liu, Y. Malinge, P. Doussiere, W. Lin, S. Gupta, K. Narayanan, I. Hoshino, M. Bresnehan, S. Sunkoju, D. Mantegazza, R. Herrick, R. Venables, H. Mahalingam, P. Seddighian, A. Fuerst, J. Davis, D. Gold, X. Pan, K. Al-hemyari, A. Agrawal, Y. Li, X. Zheng, M. Geethachar, M. Favaro, D. Zhu, A. Liu, and Y. Akulova, "800 gbps fully integrated silicon photonics transmitter for data center applications," in *2022 Optical Fiber Communications Conference and Exhibition (OFC)*, 2022, pp. 1–3.

[2] B. Pismenny, A. Morrison, and D. Tsafrir, "{ShRing}: Networking with shared receive rings," in *17th USENIX Symposium on Operating Systems Design and Implementation (OSDI 23)*, 2023, pp. 949–968.

[3] ——, "Disentangling the dual role of {NIC} receive rings," in *19th USENIX Symposium on Operating Systems Design and Implementation (OSDI 25)*, 2025, pp. 651–669.

[4] DPDK Project, “Data plane development kit (DPDK),” 2024, acessado: 02-12-2025. [Online]. Available: https://www.dpdk.org/
[5] eBPF Community. (2026) Af_xdp — ebpf docs. [Online]. Available: https://docs.ebpf.io/linux/concepts/af_xdp/
[6] D. Duplyakin, R. Ricci, A. Maricq, G. Wong, J. Duerig, E. Eide, L. Stoller, M. Hibler, D. Johnson, K. Webb, A. Akella, K. Wang, G. Ricart, L. Landweber, C. Elliott, M. Zink, and E. Cecchet, “The design and operation of CloudLab,” in *Proceedings of the 2019 USENIX Annual Technical Conference (USENIX ATC ’19)*. Renton, WA: USENIX Association, 2019, pp. 1–14. [Online]. Available: https://www.usenix.org/conference/atc19/presentation/duplyakin
[7] CloudLab Team, “CloudLab Hardware Documentation,” 2025, acessado: 02-12-2025. [Online]. Available: https://docs.cloudlab.us/hardware.html

Filipe Pirola é estudante de graduação em Ciência da Computação na Universidade Federal de Minas Gerais (UFMG), atualmente cursando seus últimos anos.

Marcos A. M. Vieira (Membro, IEEE) é Professor de Ciência da Computação na Universidade Federal de Minas Gerais (UFMG) desde 2011. Ele é bolsista de produtividade em pesquisa nível 1 do CNPq. Obteve seu Bacharelado pela UFMG em 2002, seu Mestrado pela UFMG em 2004 e seu Ph.D. em Ciência da Computação pela University of Southern California (USC) em 2010. Marcos foi Pesquisador Visitante na University of Southern California de 2018 a 2019. Seus interesses de pesquisa são em Redes de Computadores e Redes Sem Fio. Ele também é membro das comunidades de pesquisa IEEE e ACM.

Luiz F. M. Vieira (Membro, IEEE) é Professor de Ciência da Computação na Universidade Federal de Minas Gerais (UFMG). Ele obteve sua graduação e mestrado na Universidade Federal de Minas Gerais em Belo Horizonte, e o grau de Ph.D. em Ciência da Computação pela University of California Los Angeles (UCLA). Seu interesse de pesquisa é em Redes de Computadores.