AtesN-DS: Acelerando o DNS com eBPF

João V. Soares S.[*], André G. Vieira[*], Luiz F. M. Vieira[*], Marcos A. M. Vieira[*]
[*]Departamento de Ciência da Computação
Universidade Federal de Minas Gerais
Belo Horizonte, Brasil
{joaosoares, andregarcia, lfvieira, mmvieira}@dcc.ufmg.br

***Resumo*—O Sistema de Nomes de Domínio (DNS) é fundamental para a Internet, traduzindo nomes de domínio em endereços IP. No entanto, a maioria dos resolvedores modernos opera em espaço de usuário, o que gera alta latência devido à travessia da pilha de rede e trocas de contexto com o kernel. Soluções que evitam essa travessia geralmente dependem de espera ocupada, desperdiçando recursos sob baixa carga.**

Este artigo apresenta o AtesN-DS, um resolvedor DNS recursivo híbrido baseado em eBPF, que opera no kernel antes da pilha de rede, no gancho XDP, reduzindo a latência com uso eficiente de CPU. O AtesN-DS resolve nomes e gerencia *cache* diretamente no kernel. Consultas pendentes são armazenadas em mapas eBPF, e, à medida que esses mapas se aproximam da capacidade máxima, novas consultas são delegadas ao resolvedor em espaço de usuário. Tarefas mais complexas, como renovação proativa de *cache* e prevenção de erros via consultas redundantes, são encaminhadas ao espaço de usuário via *ring buffers*. Experimentos mostram que o AtesN-DS reduz em até 51% a latência e aumenta a vazão em até 213%, comparado com soluções como o hyDNS, mantendo o uso de CPU abaixo de 2%.

***Index Terms*—Sistema de domínio de nomes (DNS), eBPF (extended Berkeley Packet Filter), XDP (eXpress Data Path), In-Kernel Cache**

I. Introdução

O Sistema de Nomes de Domínio (DNS) é um sistema distribuído e hierárquico [18] que traduz nomes de domínio facilmente entendíveis por humanos em endereços IP legíveis por máquinas. Atualmente o DNS é uma parte indispensável da Internet, que além ser o pivô da resolução de nomes, também possibilita que os usuários localizem os serviços disponibilizados por um determinado nome de domínio [10] e permite o balanceamento de carga [6]. Ademais, as Redes de Distribuição de Conteúdo (CDNs) empregam o DNS para inferir a localização dos clientes, direcionando-os aos servidores geograficamente mais próximos [15]. Dessa forma, torna-se evidente o impacto global do DNS, ressaltando a importância de estudos voltados à redução do tempo de resolução de nomes e à melhoria da eficiência desse processo.

O sistema DNS tem duas partes principais: o lado do cliente (resolvedores recursivos) e o lado dos servidores (servidores raiz, TLD e autoritativos). O resolvedor recursivo, caso o domínio requisitado não esteja presente no *cache* do sistema, é responsável por percorrer desde o servidor raiz ao autoritativo, para obter o endereço IP solicitado pelo cliente. Durante uma resolução, no pior caso, uma consulta pode demandar centenas de mensagens [2]. E dado que a maioria dos resolvedores é implementada no *espaço de usuário* [13], cada uma dessas operações está sujeita a custos elevados de transição entre o espaço de usuário e kernel, além da travessia completa da pilha de rede. Alguns estudos mostram que essa travessia consome cerca de metade dos ciclos de CPU [5], mesmo em casos de acerto no *cache* em consultas DNS, sendo um dos fatores que aumentam a latência em diversas aplicações [22].

Para evitar a cópia de pacotes entre o kernel e o espaço de usuário, o uso do DPDK tem sido amplamente explorado. O DPDK é um conjunto de bibliotecas e drivers cujo objetivo é evitar completamente a pilha de rede do kernel, permitindo que os pacotes sejam processados diretamente no espaço de usuário, alcançando assim baixa latência e alta vazão [16]. Entretanto, o uso do DPDK traz desafios, como a necessidade de uma pilha de rede totalmente funcional e compatível com diversos protocolos no espaço de usuário, além do desperdício de ciclos de CPU em cenários de baixo tráfego, devido ao uso da espera ocupada para alcançar eficiência.

Outra abordagem é integrar a aplicação, ou parte dela, ao kernel do sistema operacional. Um candidato a explorar tal abordagem é utilizar o eBPF (do inglês *extended Berkely Packet Filter)*. O eBPF é um processador virtual presente no kernel do Linux capaz de executar instruções de máquina em contexto de privilégio. Tais funcionalidades são usadas principalmente para aumentar as capacidades do kernel com eficiência e segurança, sem a necessidade do kernel ser alterado. Vieira et al. [24] destacam as capacidades do eBPF, que pode ser utilizado para filtrar e processar pacotes, e também programar determinados dispositivos de redes.

O hyDNS [5] explora essa abordagem ao utilizar eBPF anexado ao gancho XDP (do inglês *eXpress Data Path)* para implementar um resolvedor recursivo híbrido diretamente no kernel. Entretanto, essa implementação apresenta limitações devido a restrições do eBPF, como a impossibilidade de criar pacotes no XDP, o que impede a implementação de funcionalidades presentes em resolvedores recursivos de alto desempenho. Para superar esses problemas, este artigo apresenta o AtesN-DS, um resolvedor recursivo híbrido implementado em eBPF e anexado ao gancho XDP, que delega a criação de pacotes em demanda ao espaço de usuário usando *ring buffers*, dispondo de *cache* e resolução de domínios em kernel, política de renovação de *cache* passiva e proativa e uma heurística de prevenção de falhas. Experimentos mostram que o AtesN-DS reduz em até 51% a latência e aumenta a vazão em até 213% em comparação com soluções como o hyDNS, mantendo o uso de CPU abaixo de 2%.

O restante deste artigo está organizado da seguinte forma: a Seção II apresenta os trabalhos relacionados. A Seção III descreve o design do AtesN-DS. A Seção IV discute os resultados obtidos. Por fim, a Seção V conclui o artigo e aponta direções para trabalhos futuros.

II. Trabalhos relacionados

O hyDNS [5] é um resolvedor DNS recursivo híbrido implementado com eBPF. Ele intercepta consultas DNS no gancho XDP e realiza a resolução recursiva, armazenando as respostas em um mapa eBPF para acelerar futuras consultas. Por híbrido, entende-se que o hyDNS opera integralmente no kernel, tornando-se completamente invisível para o espaço de usuário. No entanto, quando o número de consultas sendo resolvidas simultaneamente ou o espaço disponível no mapa de consultas atinge um limite pré-definido, as consultas excedentes são encaminhadas para serem tratadas por um resolvedor em espaço de usuário, atravessando a pilha de rede do kernel. Da mesma forma, o AtesN-DS é um resolvedor recursivo híbrido, porém não opera totalmente no espaço de kernel, devido a funcionalidades adicionais que não são viáveis apenas com o uso do eBPF no gancho XDP. Na Tabela I é apresentada uma comparação de funcionalidade dos dois sistemas. Os números ao lado de cada funcionalidade funcionam como referência para indicar onde elas são implementadas na arquitetura do AtesN-DS mostrada na Figura 2.

Tabela I
Comparação entre o AtesN-DS e o hyDNS

Funcionalidades	**hyDNS**	**AtesN-DS**
Suporte eBPF/XDP	Sim	Sim
Recursivo e Híbrido	Sim	Sim
Piggybacking **(1)**	Não	Sim
Mapa de zonas **(2)**	Não	Sim
Renovação passiva de *cache* **(3)**	Sim	Sim
Renovação proativa de *cache* **(4)**	Não	Sim
Prevenção a falhas **(5)**	Não	Sim

A. DNS

Cohen et al. [12] demonstram que a renovação proativa do *cache* de domínios populares pode reduzir significativamente a taxa de erros de *cache*, o que, por sua vez, diminui o tempo de resolução de nomes DNS, agilizando o estabelecimento de conexões e a realização de trocas HTTP. Shang et al. [21] propuseram o *piggybacking* de mensagens DNS com informações adicional, como ponteiros para servidores de *backup* ou de nível superior, para reduzir a latência geral das consultas e o volume de mensagens.

B. Latência

Diversos estudos têm investigado o impacto da latência na experiência dos usuários [4], incluindo pesquisas sobre navegação na web [3] e aplicações em nuvem [11]. Alguns estudos identificam a pilha de rede do sistema operacional como um dos principais fatores responsáveis pelo aumento da latência [22]. Para superar essa limitação, foram desenvolvidas pilhas de rede mais eficientes [17], estratégias para contornar a pilha [9] que entre elas se destaca o uso do DPDK [16] e o uso de eBPF [8]. O AtesN-DS e o hyDNS interceptam mensagens DNS no gancho XDP via eBPF para evitar a travessia pela pilha de rede.

C. eBPF

BMC [14] utiliza um programa eBPF conectado ao gancho XDP para interceptar requisições `GET` do Memcached via protocolo UDP, servindo respostas em *cache* diretamente de um mapa eBPF. Abranches et al. [1] demonstram que o uso de eBPF com XDP para monitoramento de rede é mais eficiente computacionalmente do que alternativas como o DPDK. Sistemas como o hXDP [7] e o eBPFlow [20, 19] demonstram que transferir o processamento de pacotes para FPGAs programados com eBPF pode alcançar baixa latência e alta vazão usando apenas uma fração dos recursos consumidos por abordagens tradicionais.

III. Design

Esta seção descreve a implementação do AtesN-DS[1], um resolvedor DNS recursivo híbrido que utiliza eBPF anexado ao gancho XDP. Para processar pacotes, programas eBPF podem ser ligados a diferentes pontos da pilha de rede do kernel Linux, e o desempenho obtido depende diretamente da camada em que estão conectados. Essas camadas são ilustradas na Figura 1. O AtesN-DS foi projetado para operar no caminho de dados expresso (XDP), de forma que os pacotes sejam interceptados e processados no nível mais baixo possível da pilha de rede, maximizando o desempenho.

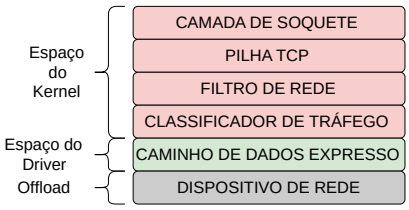

Figura 1. Camadas do fluxo de dados.

O AtesN-DS é dividido em sete programas para contornar as limitações do verificador eBPF, que atualmente consegue analisar apenas programas com até um milhão de instruções. A Figura 2 apresenta a arquitetura do resolvedor. Cada bloco pontilhado representa um dos sete programas, e os números na legenda, como **(1)**, correspondem às funcionalidades implementadas nele, conforme a Tabela I.

No programa *filtro DNS*, apenas os pacotes DNS são processados, os demais são encaminhados para a pilha de rede. Por simplicidade, o AtesN-DS oferece suporte exclusivamente a consultas do Tipo A e Classe IN transmitidas via protocolo UDP e com endereçamento Ipv4. As fora desse escopo são redirecionadas para um resolvedor em espaço de usuário. Durante a interceptação de uma consulta, caso encontrada

[1]https://github.com/ojoaosoares/AtesN-DS

no *cache* implementado por meio de um mapa do tipo *BPF_MAP_TYPE_LRU_HASH*, a resposta é imediatamente retornada para o cliente, e dependendo das condições a politica de renovação pode ser ativada, chamando o programa *renovar cache*. O mapa do tipo *BPF_MAP_TYPE_LRU_HASH* é uma tabela chave–valor que remove automaticamente as entradas menos utilizadas quando atinge sua capacidade. Neste artigo, ele será referido como *hash LRU*. Caso não haja resposta no *cache*, a consulta é registrada como pendente em um *hash LRU* e encaminhada a algum servidor DNS, possibilitando que a resposta seja enviada ao cliente assim que a resolução for concluída. O servidor escolhido para o envio da consulta é aquele cuja zona, armazenada no *cache* de zonas, implementado em um *hash LRU*, apresenta a correspondência mais específica com o nome de domínio consultado. Caso nenhuma zona correspondente seja encontrada, a consulta é encaminhada a um servidor raiz.

Quando uma resposta DNS é interceptada, ela é encaminhada ao programa *resposta DNS*, responsável por classificá-la em três categorias, cada uma determinando um comportamento específico do AtesN-DS: (1) *resposta autoritativa*, quando o servidor retorna diretamente o registro solicitado para o nome de domínio; (2) *resposta nameserver*, quando a resposta contém apenas o nome de um servidor DNS autoritativo, mas sem seu endereço IP; e (3) *resposta piggybacking*, quando o domínio do servidor autoritativo é acompanhado de seu endereço IP na seção adicional da resposta. Além dessa classificação, o programa atualiza o mapa de zonas, inserindo nele o endereço fonte da resposta.

No caso de uma *resposta nameserver*, o pacote é encaminhado ao programa *criar nova consulta DNS*, que irá resolver o nome de domínio do próximo servidor autoritativo. Para isso, é armazenada, em um *hash LRU*, uma associação entre o domínio atual e o domínio da próxima zona. Em seguida, o pacote é encaminhado para um servidor DNS com base nas informações do mapa de zonas.

Para uma *resposta piggybacking*, o pacote é direcionado ao programa *pular para próxima zona*, que extrai o IP da próxima zona na seção adicional e envia o pacote diretamente para ela. Além disso, a depender a quantidade de registros adicionais, a heurística de consultas redundantes para prevenir falhas é ativada através do programa *prevenção de falhas*.

Em caso de uma *resposta autoritativa*, duas situações são consideradas. Se for uma *nova resposta autoritativa*, gerada por uma consulta criada pelo programa *criar nova consulta DNS*, ela é encaminhada ao programa *voltar à consulta antiga*, que reconstrói a consulta anterior com base no mapa utilizado e a envia ao IP indicado. Dependendo da quantidade de respostas autoritativas, pode-se ativar a heurística de prevenção de falhas. Se for uma *resposta autoritativa original*, referente a uma consulta de cliente, ela é armazenada no *cache* e enviada ao solicitante com base no mapa de consultas pendentes.

Além desse processo básico de resolução, o AtesN-DS possui outras funcionalidades que não estão presentes no hyDNS como mostrado na tabela I.

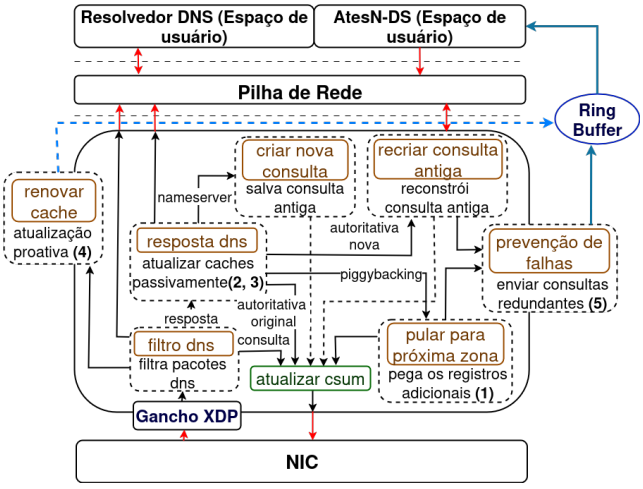

Figura 2. Arquitetura do AtesN-DS.

A. *Piggybacking*

Durante a troca de pacotes, caso um servidor não consiga resolver uma consulta, ele geralmente inclui em sua resposta referências a servidores autoritativos capazes de resolvê-la. Ao utilizar essas referências, como mostrado por Shang et al. [21], é possível reduzir significativamente a quantidade de mensagens trocadas para concluir a resolução, diminuindo assim a latência do processo. Dessa forma, ao entrar em contato com uma resposta possuindo esses registros adicionais, o AtesN-DS escolhe de forma gulosa o primeiro registro para ser o próximo servidor a ser consultado.

B. *Mapa de servidores*

Durante o processo de resolução, o resolvedor consulta diversos servidores DNS e mantém informações sobre quais zonas cada servidor é capaz de resolver. Na prática, armazenar essa associação entre zonas e servidores DNS em um *hash LRU*, apelidado de mapa de zonas, pode reduzir drasticamente a quantidade de mensagens necessárias para resolver um domínio, pois elimina a necessidade de consultas adicionais para localizar a zona responsável. Para garantir que o servidor esteja ativo, o AtesN-DS armazena no *cache* de zonas apenas as zonas obtidas durante o processamento de respostas: quando o servidor em questão retorna dados autoritativos ou referências para subzonas.

C. *Políticas de renovação de cache*

O tempo de permanência de um registro DNS em cache é determinado pelo seu TTL. Naturalmente, quando um cliente realiza uma consulta e o registro no *cache* já expirou, o resolvedor precisa efetuar uma nova consulta para obter o dado atualizado, resultando assim na atualização do *cache*. Esse mecanismo é conhecido como renovação passiva, dado que é apenas uma consequência da consulta do usuário. A principal desvantagem dessa abordagem é que as consultas após a expiração sofrerão uma falha de *cache*, aumentando a latência percebida pelos usuários. Para contornar essa limitação, a

renovação proativa [12] resolve antecipadamente domínios populares, prevenindo expirações no *cache*.

Embora pareça contraintuitivo, o AtesN-DS adota essa estratégia passivamente, fundamentando-se em princípios de localidade de referência. No momento que ocorre um acerto no *cache* para um registro válido, mas com tempo de expiração próximo de um limite configurado, é enviada uma mensagem para o espaço de usuário instruindo a resolução antecipada dessa consulta, visando renovar o *cache*. O envio do pacote ocorre a partir do espaço de usuário, uma vez que, atualmente, não é possível criar novos pacotes diretamente no gancho XDP. O AtesN-DS faz isso através do uso de BPF *ring buffers*. Eles permitem que, dentro no XDP, eventos sejam produzidos e enviados para o espaço de usuário, onde serão consumidos e as funcionalidades solicitadas poderão ser implementadas. Além disso, essa troca entre kernel e espaço de usuário ocorre de maneira eficiente sem cópias intermediarias. Após o envio do pacote pelo espaço de usuário, a resolução ocorre integralmente no kernel até a renovação do *cache*, evitando falhas de *cache* para o cliente.

D. Prevenção de falhas

O uso de BPF *ring buffers* é essencial para possibilitar a criação e o envio de pacotes no espaço de usuário, superando limitações do XDP. Essa necessidade se torna ainda mais crítica no contexto da prevenção de falhas, especialmente porque o DNS utiliza predominantemente o protocolo UDP, que não oferece garantias de entrega ou conexão. Assim, caso uma requisição seja enviada do XDP a um servidor indisponível, a resolução não poderá ser concluída no kernel, pois não haverá resposta a ser processada e o XDP não permite retransmissão ou reconstrução de pacotes. Nesses cenários, a única alternativa seria a retransmissão pelo próprio cliente.

Em geral, respostas DNS incluem múltiplos registros, tanto autoritativos quanto adicionais (*piggybacking*). Durante a resolução de domínios intermediários, o programa eBPF seleciona de forma gulosa o primeiro registro para continuar o processo iterativo no XDP. Para mitigar falhas, o AtesN-DS envia os registros restantes ao espaço de usuário, onde novas consultas são feitas para os outros endereços. Essa redundância ajuda a minimizar a chance da perda de pacotes. Além disso, ao registrar no mapa de zonas o servidor cuja resposta chegou primeiro, o sistema favorece o armazenamento em *cache* daqueles com menor latência observada.

IV. Resultados

Para avaliar o desempenho do AtesN-DS, realizamos uma comparação com nossa implementação do resolvedor HyDNS [5]. O ambiente de teste contou com dois computadores conectados ponto a ponto por fibra óptica. Essa topologia simplificada é suficiente, dado que o que está sendo avaliado é a capacidade do resolvedor, e não a rede. O servidor que executava os resolvedores possuía um processador Intel Core i7-8086K (6 núcleos a 4,00 GHz), 14 GiB de RAM, Ubuntu 22.04.5 LTS e uma placa de rede Netronome Agilio CX 10GbE. O cliente utilizava um Intel Core i7-4790 (4 núcleos a 3,6 GHz), 10 GiB de RAM, Ubuntu 24.04 LTS e placa Ethernet X710 10GbE.

Ambos os servidores realizavam consultas iterativas a servidores externos, garantindo um ambiente realista para medir os tempos de resposta. Já o AtesN-DS e o HyDNS foram executados diretamente no kernel, ambos anexados no modo driver, aproveitando a compatibilidade da placa de rede Netronome. Do lado do cliente, uma ferramente chamada dnspyre [23] foi usada junto com uma lista de domínios[2] para gerar as consultas e metrificar a latência e vazão, enquanto do lado do servidor foi usado um *script* para monitorar o uso de CPU (kernel e espaço de usuário). Cada teste foi precedido por um período de aquecimento de *cache* e teve duração de 60 segundos. Em cada comparação, os resolvedores foram avaliados sob diferentes níveis de carga de trabalho, variando o número de conexões simultâneas. Para assegurar significância estatística e possibilitar o cálculo dos desvios padrão, cada teste foi repetido 60 vezes.

A. Análise no lado do Cliente

A Figura 3 apresenta a comparação da latência média entre o AtesN-DS e o HyDNS sob diferentes números de conexões simultâneas. Observa-se que a latência diminui à medida que o número de conexões simultâneas aumenta, devido à utilização mais eficiente dos seis núcleos da máquina que executa os resolvedores e ao aquecimento do cache de instruções e de dados da CPU, bem como do cache de consultas DNS. Além disso, no cenário de 256 conexões simultâneas o AtesN-DS obteve uma latência média cerca de 51% inferior à do HyDNS, e de forma semelhante manteve uma vantagem consistente em todos os outros cenários. Vale destacar que, em todos os testes, os desvios padrão não se sobrepuseram, indicando significância estatística. Essa vantagem significativa é resultado da renovação proativa de *cache*, que eleva significativamente a taxa de acertos no *cache*, e o uso de um mapa de zonas, que reduz o número de trocas de mensagens necessárias para resolver consultas em caso de falhas no *cache*.

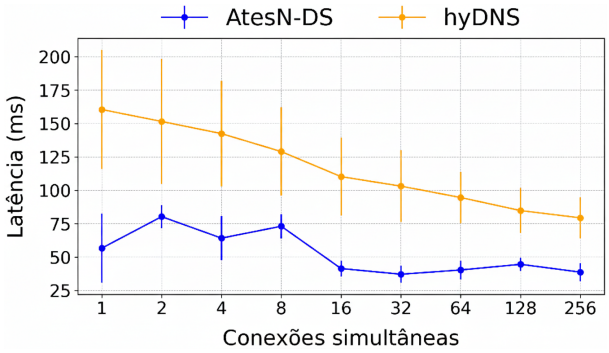

Figura 3. Latência média

A Figura 4 mostra a comparação da vazão em distintos níveis de conexões simultâneas. Nota-se que, a vazão de ambos

[2]https://raw.githubusercontent.com/zer0h/top-1000000-domains/refs/heads/master/top-10000-domains

os sistemas aumenta à medida que o número de conexões simultâneas cresce pelos mesmos motivos citados no teste de latência, e que no cenário de 256 conexões simultâneas, o AtesN-DS alcançou uma vazão de aproximadamente 2900 consultas por segundo, cerca de 213% superior ao HyDNS. Ressalta-se que, neste cenário, não houve sobreposição dos desvios padrões, o que confirma a significância estatística. Isso destaca a maior escalabilidade do AtesN-DS, que se deve principalmente ao mapa de zonas que é altamente reutilizável, já que uma zona pode resolver diversos domínios, reduzindo o número de mensagens trocadas e aumentando a capacidade de resolução do resolvedor. Além disso, a heurística de prevenção de falhas contribui para diminuir a probabilidade de perda de pacotes e falhas na resolução, que só seriam corrigidas por retransmissão do cliente.

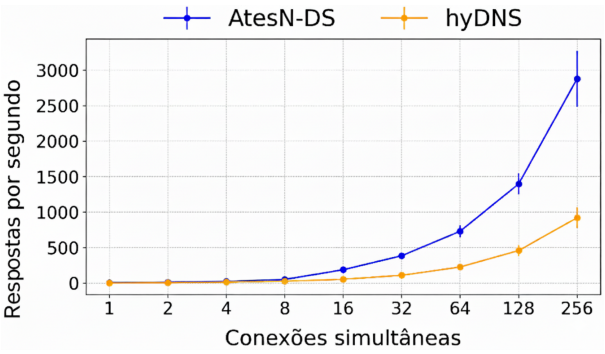

Figura 4. Vazão

B. Análise do lado do servidor

Na Figura 5 é possível ver a comparação do uso da CPU no kernel. Nela é visto que o AtesN-DS, possui um uso de CPU maior em comparação com o HyDNS. Os principais fatores que contribuem para esse aumento são a busca contínua no mapa de zonas, que, no pior caso, exige a verificação de cada subdomínio para selecionar a melhor zona, e a escrita no *ring buffer*. Embora exista esse aumento, o uso de CPU continua extremamente baixo, não ultrapassando 0.3%, o que demonstra a eficiência do AtesN-DS mesmo com as funcionalidades adicionais implementadas no kernel.

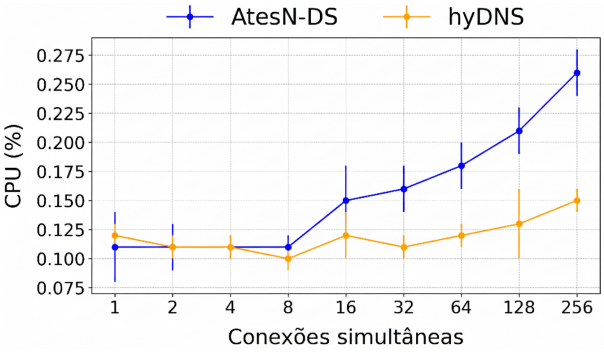

Figura 5. Uso de CPU no kernel

A Figura 6 apresenta a comparação do uso de CPU no espaço de usuário. Observa-se que o consumo do hyDNS permanece praticamente constante, o que se deve ao fato de sua execução ocorrer inteiramente no kernel. Em contraste, o AtesN-DS apresenta um uso de CPU proporcional à carga, resultado da geração de pacotes em espaço de usuário, seja para a renovação do *cache*, seja para o envio de consultas redundantes. Apesar desse crescimento proporcional, o uso absoluto de CPU permaneceu baixo, alcançando apenas 1,74% mesmo no cenário com 256 conexões simultâneas. Somando os consumos de CPU em espaço de kernel e de usuário, o uso total atingiu apenas 2% no cenário com 256 conexões simultâneas.

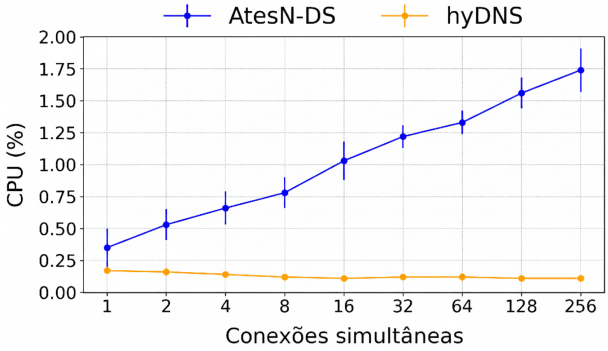

Figura 6. Uso de CPU no espaço de usuário

V. Conclusão

Neste trabalho, apresentou-se o AtesN-DS, um resolvedor recursivo híbrido baseado em eBPF, capaz de interceptar e processar pacotes diretamente no gancho XDP, evitando a pilha de rede e excessivas trocas de contexto. Além de realizar *cache* e resolução de domínios no espaço de kernel, o AtesN-DS oferece funcionalidades adicionais, como um *cache* de zonas, renovação proativa do *cache* e uma política de prevenção de falhas. Discute-se que certas funcionalidades não podem ser implementadas exclusivamente com eBPF no gancho XDP, uma vez que esse contexto ainda não permite a geração de novos pacotes. Para contornar essa limitação, o AtesN-DS utiliza *ring buffers* para requisitar a criação de novos pacotes no espaço de usuário. Os testes demonstram que, sob cargas elevadas, o AtesN-DS alcança latência até 51% menor e vazão até 213% maior quando comparado ao hyDNS, um resolvedor híbrido e recursivo que opera exclusivamente no kernel, consumindo, no máximo, 2% de CPU.

Como trabalhos futuros, pretende-se implementar o suporte a pacotes IPv6 no AtesN-DS, ampliando sua compatibilidade com a nova geração de protocolos da Internet. Com isso, também será possível resolver e armazenar registros do tipo AAAA, específicos para endereços IPv6. Além disso, visando aprimorar a escalabilidade do sistema, planeja-se a utilização de mapas eBPF por CPU, em conjunto com NICs programáveis, para manter um *cache* coeso e eficiente. Por fim, serão conduzidos testes adicionais para mensurar o impacto de cada nova funcionalidade no desempenho geral do sistema.

Referências

[1] Marcelo Abranches, Oliver Michel, Eric Keller, and Stefan Schmid. 2021. Efficient Network Monitoring Applications in the Kernel with eBPF and XDP. In *2021 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN)*. 28–34. doi:10.1109/NFV-SDN53031.2021.9665095

[2] Yehuda Afek, Anat Bremler-Barr, and Lior Shafir. 2020. NXNSAttack: Recursive DNS inefficiencies and vulnerabilities. In *29th USENIX Security Symposium (USENIX Security 20)*. 631–648.

[3] Ioannis Arapakis, Souneil Park, and Martin Pielot. 2021. Impact of response latency on user behaviour in mobile web search. In *Proceedings of the 2021 Conference on Human Information Interaction and Retrieval*. 279–283.

[4] Christiane Attig, Nadine Rauh, Thomas Franke, and Josef F Krems. 2017. System latency guidelines then and now–is zero latency really considered necessary?. In *International Conference on Engineering Psychology and Cognitive Ergonomics*. Springer, 3–14.

[5] Joshua Bardinelli, Yifan Zhang, Jianchang Su, Linpu Huang, Aidan Parilla, Rachel Jarvi, Sameer G. Kulkarni, and Wei Zhang. 2024. hyDNS: Acceleration of DNS Through Kernel Space Resolution. In *Proceedings of the ACM SIGCOMM 2024 Workshop on EBPF and Kernel Extensions* (Sydney, NSW, Australia) *(eBPF '24)*. Association for Computing Machinery, New York, NY, USA, 58–64. doi:10.1145/3672197.3673439

[6] Thomas Brisco. 1995. *DNS support for load balancing*. Technical Report.

[7] Marco Spaziani Brunella, Giacomo Belocchi, Marco Bonola, Salvatore Pontarelli, Giuseppe Siracusano, Giuseppe Bianchi, Aniello Cammarano, Alessandro Palumbo, Luca Petrucci, and Roberto Bifulco. 2022. hXDP: Efficient software packet processing on FPGA NICs. *Commun. ACM* 65, 8 (jul 2022), 92–100.

[8] Qizhe Cai, Shubham Chaudhary, Midhul Vuppalapati, Jaehyun Hwang, and Rachit Agarwal. 2021. Understanding host network stack overheads. In *Proceedings of the 2021 ACM SIGCOMM 2021 Conference*. 65–77.

[9] Ruining Chen and Guoao Sun. 2018. A Survey of Kernel-Bypass Techniques in Network Stack. In *Proceedings of the 2018 2nd International Conference on Computer Science and Artificial Intelligence* (Shenzhen, China) *(CSAI '18)*. Association for Computing Machinery, New York, NY, USA, 474–477. doi:10.1145/3297156.3297242

[10] Stuart Cheshire and Marc Krochmal. 2013. *DNS-based service discovery*. Technical Report.

[11] Mark Claypool and David Finkel. 2014. The effects of latency on player performance in cloud-based games. In *2014 13th Annual Workshop on Network and Systems Support for Games*. IEEE, 1–6.

[12] Edith Cohen and Haim Kaplan. 2003. Proactive caching of DNS records: Addressing a performance bottleneck. *Computer Networks* 41, 6 (2003), 707–726.

[13] Hongyu Gao, Vinod Yegneswaran, Jian Jiang, Yan Chen, Phillip Porras, Shalini Ghosh, and Haixin Duan. 2016. Reexamining DNS From a Global Recursive Resolver Perspective. *IEEE/ACM Transactions on Networking* 24, 1 (2016), 43–57. doi:10.1109/TNET.2014.2358637

[14] Yoann Ghigoff, Julien Sopena, Kahina Lazri, Antoine Blin, and Gilles Muller. 2021. BMC: Accelerating Memcached using Safe In-kernel Caching and Pre-stack Processing. In *18th USENIX Symposium on Networked Systems Design and Implementation (NSDI 21)*. USENIX Association, 487–501.

[15] Hadrien Hours, Ernst Biersack, Patrick Loiseau, Alessandro Finamore, and Marco Mellia. 2016. A study of the impact of DNS resolvers on CDN performance using a causal approach. *Computer Networks* 109, 200–210.

[16] Michail-Alexandros Kourtis, Georgios Xilouris, Vincenzo Riccobene, Michael J McGrath, Giuseppe Petralia, Harilaos Koumaras, Georgios Gardikis, and Fidel Liberal. 2015. Enhancing VNF performance by exploiting SR-IOV and DPDK packet processing acceleration. In *2015 IEEE Conference on Network Function Virtualization and Software Defined Network*. IEEE, 74–78.

[17] Ilias Marinos, Robert N.M. Watson, and Mark Handley. 2014. Network stack specialization for performance. *SIGCOMM Comput. Commun. Rev.* 44, 4 (Aug. 2014), 175–186. doi:10.1145/2740070.2626311

[18] Paul Mockapetris. 1987. *Domain names-concepts and facilities*. Technical Report.

[19] Racyus D. G. Pacífico, Matheus S. Castanho, Luiz F. M. Vieira, Marcos A. M. Vieira, Lucas F. S. Duarte, and José A. M. Nacif. 2021. Application Layer Packet Classifier in Hardware. In *2021 IFIP/IEEE International Symposium on Integrated Network Management (IM)*. IEEE, Bordeaux, France, 515–522.

[20] Racyus D. G. Pacífico, Lucas F. S. Duarte, Luiz F. M. Vieira, Barath Raghavan, José A. M. Nacif, and Marcos A. M. Vieira. 2024. eBPFlow: A Hardware/Software Platform to Seamlessly Offload Network Functions Leveraging eBPF. *IEEE/ACM Transactions on Networking* 32, 2 (2024), 1319–1332. doi:10.1109/TNET.2023.3318251

[21] Hao Shang and Craig E Wills. 2006. Piggybacking related domain names to improve DNS performance. *Computer Networks* 50, 11 (2006), 1733–1748.

[22] Ankit Singla, Balakrishnan Chandrasekaran, P Brighten Godfrey, and Bruce Maggs. 2014. The internet at the speed of light. In *Proceedings of the 13th ACM Workshop on Hot Topics in Networks*. 1–7.

[23] Tantalor93. [n. d.]. dnspyre: A DNS benchmarking tool. https://tantalor93.github.io/dnspyre/. Acessado em 17 de julho de 2025.

[24] Marcos AM Vieira, Matheus S Castanho, Racyus DG Pacífico, Elerson RS Santos, Eduardo PM Câmara Júnior, and Luiz FM Vieira. 2020. Fast packet processing with ebpf and xdp: Concepts, code, challenges, and applications. *ACM Computing Surveys (CSUR)* 53, 1 (2020), 1–36.