UNIVERSIDADE FEDERAL DO PAMPA

Patric Lincoln Ramires Izolan

ANÁLISE DE ESCALABILIDADE E DESEMPENHO DE APLICAÇÕES DE REDE EM DPDK

Alegrete
Outubro de 2023

Patric Lincoln Ramires Izolan

ANÁLISE DE ESCALABILIDADE E DESEMPENHO DE APLICAÇÕES DE REDE EM DPDK

Dissertação apresentada ao Programa de Pós-graduação Stricto Sensu em Engenharia Elétrica da Universidade Federal do Pampa, como requisito parcial para obtenção do Título de Mestre em Engenharia Elétrica.

Orientador: Fábio Diniz Rossi
Coorientador: Marcelo Caggiani Luizelli

Alegrete
Outubro de 2023

Ficha catalográfica elaborada automaticamente com os dados fornecidos
pelo(a) autor(a) através do Módulo de Biblioteca do
Sistema GURI (Gestão Unificada de Recursos Institucionais) .

I99a Izolan, Patric Lincoln Ramires
ANÁLISE DE ESCALABILIDADE E DESEMPENHO DE APLICAÇÕES DE REDE EM DPDK / Patric Lincoln Ramires Izolan.
64 p.

Dissertação(Mestrado)-- Universidade Federal do Pampa, MESTRADO EM ENGENHARIA ELÉTRICA, 2023.
"Orientação: Fábio Diniz Rossi".

1. DPDK. 2. Programabilidade de redes. 3. P4. I. Título.

Patric Lincoln Ramires Izolan

ANÁLISE DE ESCALABILIDADE E DESEMPENHO DE APLICAÇÕES DE REDE EM DPDK

Dissertação apresentada ao Programa de Pós-graduação Stricto Sensu em Engenharia Elétrica da Universidade Federal do Pampa, como requisito parcial para obtenção do Título de Mestre em Engenharia Elétrica.

Área de concentração: Otimização de Sistemas

Dissertação defendida e aprovada em: Alegrete, 26 de outubro de 2023.
Banca examinadora:

Fábio Diniz Rossi
Fábio Diniz Rossi

Marcelo Caggiani Luizelli
Unipampa

Heleno Carmo Borges Cabral
IFFAR

Paulo Silas Severo de Souza
IFFAR/Unipampa

Prof. Dr. Paulo Silas Severo de Souza

(IFFar)

Assinado eletronicamente por **FABIO DINIZ ROSSI**, **PESSOAL VOLUNTÁRIO**, em 26/10/2023, às 16:05, conforme horário oficial de Brasília, de acordo com as normativas legais aplicáveis.

Assinado eletronicamente por **Paulo Silas Severo de Souza**, **Usuário Externo**, em 26/10/2023, às 16:05, conforme horário oficial de Brasília, de acordo com as normativas legais aplicáveis.

Assinado eletronicamente por **Heleno Carmo Borges Cabral**, **Usuário Externo**, em 27/10/2023, às 20:08, conforme horário oficial de Brasília, de acordo com as normativas legais aplicáveis.

Assinado eletronicamente por **MARCELO CAGGIANI LUIZELLI**, **PROFESSOR DO MAGISTERIO SUPERIOR**, em 08/12/2023, às 06:55, conforme horário oficial de Brasília, de acordo com as normativas legais aplicáveis.

A autenticidade deste documento pode ser conferida no site https://sei.unipampa.edu.br/sei/controlador_externo.php?acao=documento_conferir&id_orgao_acesso_externo=0, informando o código verificador **1276777** e o código CRC **C9409952**.

RESUMO

As redes de comunicação desempenham um papel fundamental em nossa sociedade, suportando uma ampla variedade de serviços e aplicações. No entanto, à medida que essas redes crescem em complexidade e as demandas dos serviços evoluem, surge a necessidade de torná-las mais flexíveis e adaptáveis. A programabilidade de redes, impulsionada por tecnologias como P4, eBPF e DPDK, oferece a capacidade de personalizar o comportamento da rede, permitindo inovações e otimizações em tempo real. O P4 permite programar a camada de encaminhamento da rede, enquanto o eBPF estende a programabilidade até o kernel do sistema operacional, e o DPDK acelera operações de E/S para alto desempenho. A programabilidade de redes é essencial devido a várias razões, como a dinamicidade das aplicações, a necessidade de escalabilidade e eficiência, a importância da segurança e monitoramento, a adoção de novas tecnologias e a personalização de serviços. A capacidade de programar a rede de acordo com requisitos específicos é uma vantagem competitiva e contribui para infraestruturas de rede mais inteligentes e adaptáveis. Este trabalho se concentra na avaliação da escalabilidade e desempenho de aplicações desenvolvidas em P4 quando transferidas para execução no nível de usuário através do DPDK. O objetivo é identificar incompatibilidades e avaliar o desempenho dessas aplicações em um ambiente de programabilidade de plano de dados em DPDK. Essas informações podem servir como um guia para futuras implementações e contribuir para o avanço das tecnologias de programabilidade de redes.

Palavras-chave: DPDK, programabilidade de redes, P4.

ABSTRACT

Communication networks play a fundamental role in our society, supporting a wide range of services and applications. However, as these networks grow in complexity and service demands evolve, there is a need to make them more flexible and adaptable. Network programmability, driven by technologies like P4, eBPF, and DPDK, offers the ability to customize network behavior, allowing real-time innovations and optimizations. P4 allows programming the network forwarding layer, while eBPF extends programmability to the kernel of the operating system, and DPDK accelerates I/O operations for high performance. Network programmability is essential for several reasons, including the dynamism of applications, the need for scalability and efficiency, the importance of security and monitoring, the adoption of new technologies, and the customization of services. The ability to program the network according to specific requirements is a competitive advantage and contributes to smarter and more adaptable network infrastructures. This work focuses on evaluating the scalability and performance of applications developed in P4 when transferred for execution at the user level through DPDK. The goal is to identify incompatibilities and assess the performance of these applications in a data plane programmability environment in DPDK. This information can serve as a guide for future implementations and contribute to the advancement of network programmability technologies.

Keywords: DPDK, network programmability, P4.

LISTA DE ILUSTRAÇÕES

Figura 1 – Visão geral da arquitetura de SDN. 20
Figura 2 – Comparação entre o modelo de rede tradicional e SDN. 21
Figura 3 – Processamento na arquitetura PISA. 23
Figura 4 – Arquitetura de compiladores P4. 27
Figura 5 – Vazão e latência obtidas durante operações de leitura e escrita. 38
Figura 6 – Vazão e latência obtidas durante a recirculação de pacotes no pipeline. 40
Figura 7 – Desempenho mediante aplicações com múltiplas tabelas. 40
Figura 8 – Impacto com uso de funções hash. 41
Figura 9 – Desempenho no uso de operações aritméticas. 42
Figura 10 – Vazão e latência obtidas durante operações de leitura e escrita variando o número de registradores (pacotes de 64B). 44
Figura 11 – Vazão e latência obtidas durante operações de leitura e escrita para diferentes tamanhos de pacotes. 46
Figura 12 – Vazão e latência obtidas durante operações de leitura e escrita para diferentes tamanhos de registradores. 48
Figura 13 – Impacto na recirculação de pacotes em pipeline DPDK. 50
Figura 14 – Desempenho do pipeline DPDK no uso de múltiplas tabelas. 52
Figura 15 – Desempenho do pipeline DPDK no uso de funções criptográficas. . . . 54
Figura 16 – Desempenho do pipeline DPDK para consecutivas operações aritméticas. 55

SUMÁRIO

1 INTRODUÇÃO 13
1.1 Contexto e Motivação 15
1.2 Problema da Pesquisa 17
1.3 Objetivos e Contribuições 17
1.4 Organização do Trabalho 18

2 REFERENCIAL TEÓRICO 19
2.1 Redes Definidas por *Software* 19
2.2 Linguagem P4 22
2.2.1 Arquiteturas da linguagem P4 24
2.2.1.1 v1model 24
2.2.1.2 Portable Switch Architecture (PSA) 25
2.2.1.3 Portable NIC Architecture (PNA) 26
2.2.1.4 Compilador P4C 26
2.3 DPDK 27
2.4 Trabalhos Relacionados 32

3 ANÁLISE DE ESCALABILIDADE DO DPDK 35
3.1 Metodologia 35
3.1.1 Ambiente de Testes 35
3.1.2 Métrica Aplicada 35
3.2 Resultados 37
3.2.1 Desempenho com Configuração ST(i) 37
3.2.1.1 Desempenho durante Operações de Leitura e Escrita 37
3.2.1.2 Desempenho Durante Recirculação de Pacotes 39
3.2.1.3 Desempenho de Aplicações com Múltiplas Tabelas 40
3.2.1.4 Desempenho no Uso de Funções Criptográficas 41
3.2.1.5 Desempenho no Uso de Operações Aritméticas 42
3.2.2 Desempenho com Especificações ST(ii), ST(iii) e ST(iv) 43
3.2.2.1 Desempenho durante Operações de Leitura e Escrita 43
3.2.2.2 Desempenho Durante Recirculação de Pacotes 49
3.2.2.3 Desempenho de Aplicações com Múltiplas Tabelas 51
3.2.2.4 Desempenho no Uso de Funções Criptográficas 53
3.2.2.5 Desempenho no Uso de Operações Aritméticas 54
3.3 Discussão 56

4 CONSIDERAÇÕES FINAIS 61

REFERÊNCIAS **63**

1 INTRODUÇÃO

As redes de comunicação têm desempenhado um papel fundamental na nossa sociedade desde o seu surgimento. Seja para comunicação entre dispositivos, acesso à internet, transmissão de dados ou suporte a serviços críticos, as redes desempenham um papel cada vez mais crucial em nosso dia a dia. No entanto, a crescente complexidade e as demandas em constante evolução colocaram um desafio significativo para as infraestruturas de rede: a necessidade de flexibilidade e adaptabilidade para atender às crescentes demandas e requisitos de aplicativos. Nesse cenário, a programabilidade de redes emergiu como uma resposta essencial para a transformação das infraestruturas de rede atuais. A capacidade de programar e personalizar o comportamento da rede abre portas para uma série de inovações e otimizações que vão além das configurações estáticas tradicionais.

As redes de computadores, em sua forma mais tradicional, eram caracterizadas por dispositivos de hardware que desempenhavam funções de encaminhamento de pacotes e roteamento com base em regras de configuração pré-definidas. Essas configurações eram geralmente inflexíveis e exigiam um esforço significativo para serem adaptadas a novos requisitos ou cenários de uso. No entanto, à medida que a demanda por redes mais dinâmicas e personalizáveis cresceu, a necessidade de tornar as redes programáveis se tornou evidente.

Uma das principais tecnologias que impulsionou a programabilidade de redes é o P4 (Programming Protocol-Independent Packet Processors). O P4 é uma linguagem de programação projetada especificamente para definir o comportamento da camada de encaminhamento de dispositivos de rede. Com o P4, os engenheiros de rede podem especificar como os pacotes de dados devem ser processados e encaminhados em uma rede, sem depender das limitações impostas pelos protocolos de rede tradicionais. O P4 permite que os dispositivos de rede sejam reprogramados dinamicamente para atender a requisitos específicos de aplicativos ou cenários de tráfego. Isso significa que as redes podem ser adaptadas e otimizadas em tempo real, sem a necessidade de substituir hardware físico. Por exemplo, em uma rede que precisa priorizar o tráfego de vídeo em tempo real, o P4 pode ser usado para ajustar a lógica de encaminhamento para garantir baixa latência e qualidade de serviço para esse tipo de tráfego.

Enquanto o P4 se concentra na programabilidade da camada de encaminhamento de rede, o eBPF (extended Berkeley Packet Filter) estende a programabilidade até o próprio kernel do sistema operacional. Originalmente desenvolvido como parte do kernel Linux, o eBPF permite que os desenvolvedores escrevam código personalizado que é executado diretamente no kernel, permitindo a análise e a manipulação de pacotes em tempo real. O eBPF é frequentemente usado para criar funções personalizadas de filtragem, monitoramento e segurança em redes. Ele oferece a capacidade de tomar decisões com base no conteúdo dos pacotes de rede, direcionando-os para ações específicas, como inspeção de segurança, balanceamento de carga ou redirecionamento para interfaces de rede virtuais.

A flexibilidade do eBPF o tornou uma ferramenta poderosa para aprimorar a visibilidade e o controle nas infraestruturas de rede.

Outra tecnologia importante na programabilidade de redes é o DPDK (Data Plane Development Kit). Enquanto o P4 e o eBPF se concentram na programação da lógica de encaminhamento e processamento, o DPDK se destaca na aceleração de operações de E/S (Entrada e Saída) para alto desempenho. Ele oferece uma pilha de software otimizada para interfaces de rede de alto desempenho, permitindo o processamento de pacotes em taxas extremamente altas. O DPDK é amplamente utilizado em aplicativos que exigem processamento de pacotes de baixa latência e alto throughput, como firewalls de alto desempenho, balanceadores de carga e sistemas de telecomunicações. Sua capacidade de programar e otimizar o processamento de pacotes em hardware de uso geral torna-o uma escolha valiosa em ambientes onde o desempenho é crítico.

A programabilidade de redes oferece inúmeras vantagens e se tornou uma necessidade premente nas infraestruturas de rede atuais. Vamos explorar algumas das razões pelas quais as redes precisam ser programáveis:

- **Dinamismo das Aplicações:** As aplicações modernas são dinâmicas e variadas. Elas podem exigir níveis variados de largura de banda, qualidade de serviço e segurança, dependendo do contexto. Redes estáticas e inflexíveis não podem mais atender a essa demanda. Com a programabilidade, as redes podem se adaptar às necessidades específicas das aplicações em tempo real, fornecendo recursos sob demanda.

- **Escalabilidade e Eficiência:** À medida que a escala das redes cresce, a eficiência se torna fundamental. As redes programáveis podem ser otimizadas para o tráfego real, eliminando gargalos e minimizando o desperdício de recursos. Isso é especialmente importante em data centers e ambientes de nuvem, onde a eficiência de recursos é crítica para o desempenho e a economia.

- **Segurança e Monitoramento:** A segurança da rede é uma preocupação central em qualquer infraestrutura. A programabilidade permite a implementação de funções de segurança personalizadas, como detecção de intrusões, análise de tráfego malicioso e isolamento de ameaças em tempo real. Além disso, a capacidade de monitorar e analisar o tráfego em tempo real é crucial para identificar anomalias e responder rapidamente a incidentes de segurança.

- **Adoção de Novas Tecnologias:** A tecnologia de rede está em constante evolução, com o surgimento de novos protocolos e padrões. A programabilidade permite a adaptação rápida a essas mudanças, sem a necessidade de substituir hardware físico. Isso acelera a adoção de novas tecnologias e padrões, mantendo a infraestrutura de rede atualizada e compatível.

- **Personalização de Serviços:** As redes programáveis permitem a oferta de serviços personalizados aos clientes e usuários finais. Provedores de serviços podem ajustar o comportamento da rede de acordo com as necessidades de seus clientes, criando serviços diferenciados e adaptáveis.

Assim, a programabilidade de redes, impulsionada por tecnologias como P4, eBPF e DPDK, está revolucionando a forma como as infraestruturas de rede são projetadas e operadas. Ela oferece flexibilidade, adaptabilidade e eficiência que são essenciais para atender às crescentes demandas das aplicações modernas e das crescentes escalas de rede. À medida que a necessidade de redes programáveis continua a crescer, é fundamental que as organizações invistam em talentos e recursos para aproveitar ao máximo essas tecnologias. A capacidade de personalizar o comportamento da rede e adaptá-lo às necessidades específicas é uma vantagem competitiva que pode impulsionar a inovação e melhorar a eficiência em todos os setores.

A revolução da programabilidade de redes está apenas começando, e as possibilidades são empolgantes. À medida que as infraestruturas de rede se tornam mais inteligentes e adaptáveis, podemos esperar um futuro onde a rede não é mais uma limitação, mas sim uma facilitadora da inovação e do progresso tecnológico. Este trabalho avalia uma das vantagens da programabilidade de redes elencada anteriormente, que consiste na escalabilidade. Propomos avaliar a escalabilidade de soluções desenvolvidas utilizando DPDK, que consiste em uma aplicação em nível de usuário para realizar programabilidade de redes e que permite uma curva de aprendizado mais rápida do que o entendimento dos códigos eBPF e DPDK.

1.1 CONTEXTO E MOTIVAÇÃO

As redes de computadores consistem em infraestruturas de comunicação nas quais são empregados diversos dispositivos/ativos de redes distintos. Esses equipamentos podem ser implantados em diversos ambientes em uma infraestrutura de redes, suportando uma ampla variedade de casos de uso. Embora exista um movimento em direção à especialização dos dispositivos visando à realização de tarefas específicas, também existe uma direção tomada no sentido de tornar os dispositivos de rede menos especializados e mais gerais, visando reduzir custos e oferecer maior flexibilidade (ELENBOGEN, 1999).

Nesse contexto, o modelo de rede definida por software (*Software-Defined Network* — SDN) (SHALIMOV et al., 2013) emerge trazendo maior programabilidade e flexibilidade às infraestruturas de rede, separando o plano de controle do plano de dados. Em geral, o modelo de SDN separa as funções de controle e encaminhamento de dados do hardware físico dos elementos que compõem essa rede, criando uma infraestrutura de rede mais gerenciável e dinâmica. Isso ocorre através da possibilidade de se programar os dispositivos empregados na formação da rede, como placas de rede, roteadores, switches e demais

ativos que possam ser utilizados para a comunicação de dados com ou sem o seu propósito definido, além da possibilidade da aplicação desse modelo nos sistemas operacionais já empregados e servidores físicos utilizados na infraestrutura da rede.

Com o advento do SDN, surgiram os dispositivos de rede de propósito programável. Esses dispositivos promoveram uma mudança significativa no que se conhece de rede de dados, proporcionando uma redução considerável no tempo de espera de atualizações, correções ou mesmo novas funcionalidades. Além do que já foi mencionado, dispositivos de rede programáveis permitem aos operadores um caminho para implementação de novas *features* de forma rápida e conforme a necessidade.

A capacidade de programar a rede constitui um princípio fundamental no contexto de SDN. Ainda que essa noção não seja recente, o diferencial se encontra na sua aplicação no âmbito das SDN, onde os dispositivos de rede não são simplesmente gerenciados, mas também interagem ativamente. Essa interação estabelece uma comunicação bidirecional entre os planos de controle e dados, um modelo que difere substancialmente dos métodos tradicionais de gerenciamento de rede, que não envolvem um *feedback* dos dispositivos ativos.

A ideia de programabilidade está intimamente relacionada à capacidade de modificar, direta ou indiretamente, as tabelas de instruções de um componente de rede. Além disso, em infraestruturas que operam no modelo de SDN, a programabilidade não se limita às funcionalidades predefinidas do hardware em questão. Nesse contexto, novos conceitos de comunicação e transmissão de pacotes podem ser implementados sem que haja necessidade de modificar o plano de dados existente.

As redes SDN, com seu foco na programabilidade, permitem uma abordagem mais dinâmica e flexível na configuração e gestão de redes. Através dessa capacidade de programação, é possível ajustar o comportamento da rede conforme as necessidades específicas, seja para otimizar o roteamento, aplicar políticas de segurança personalizadas ou habilitar novos serviços de maneira ágil.

Para que se tenha flexibilidade no uso de dispositivos programáveis, a dependência com o fabricante em disponibilizar funcionalidades e atualizações do *software* precisa ser rompida. No entanto, as linguagens de programação utilizadas geralmente para tal propósito têm alta complexidade, além de serem variadas dependendo do fabricante. A necessidade de meios de abstrações para permitir a rápida programação e prototipação no SDN exige uma linguagem de programação de alto nível que alcance os diversos dispositivos programáveis, seja qual for o seu fabricante.

O P4 surge como uma linguagem de alto nível independente do protocolo do hardware para a programabilidade do plano de dados de dispositivos SDN (GOSWAMI; KULKARNI; PAULOSE, 2023) (NEVES et al., 2021). A maior necessidade de programação em redes SDN fez com que, além da linguagem de programação de alto nível específica de domínio, como o P4, surgissem outras opções para realizar a programabilidade do plano de

dados, como o DPDK. Diferente do P4, que é uma linguagem de programação, o DPDK é um acelerador de entrada/saída de pacotes, com o qual é possível programar o plano de dados.

Esses dois níveis de programabilidade do plano de dados apresentam distintas vantagens de uso em relação à infraestrutura utilizada pela rede. O P4, como linguagem de programação, tem alcance na maior parte dos dispositivos programáveis, o que não é verdade em relação ao DPDK. Por outro lado, o DPDK permite que recursos de processamento de uso geral sejam utilizados para o processamento de pacotes de alto desempenho. Embora sejam níveis de programabilidade do plano de dados distintos e antagonistas, uma possível portabilidade de aplicações entre os níveis ou mesmo o uso de uma programabilidade do plano de dados de forma heterogênea são pontos de destaque para uma rápida e harmoniosa migração do uso de SDN.

1.2 PROBLEMA DA PESQUISA

Embora o conceito de redes definidas por software não seja algo novo, as tecnologias que permitem a programabilidade de redes ainda não estão maduras. Muitas das limitações dessas tecnologias só são descobertas no momento de sua implementação, sejam limitações do hardware (como quantidade de registradores, memória e processamento), sejam limitações de software (como a falta de implementação de estruturas de repetição e tratamento de ponto flutuante). Assim, há de verificar os limites de tais tecnologias, para que futuros trabalhos possam utilizar essas informações como oráculo para novas implementações. Nesse contexto, ainda existem muitas dúvidas sobre a portabilidade para execução de códigos de programabilidade de redes, especialmente quando falamos desse tipo de desenvolvimento ao nível de usuário.

1.3 OBJETIVOS E CONTRIBUIÇÕES

Este trabalho tem por objetivo geral, analisar a conversão, escalabilidade, e por conseguinte o desempenho, de aplicações desenvolvidas em P4 para execução ao nível de usuário através do DPDK.

Sendo assim, podemos elencar alguns objetivos específicos deste trabalho:

1. Identificar as incompatibilidades para uma efetiva portabilidade de programas em linguagem P4, para sua execução no nível de programabilidade do plano de dados (PDP) em DPDK.

2. Avaliar o desempenho das aplicações transferidas para execução em pipeline no nível do usuário quanto a diversas métricas de desempenho.

1.4 ORGANIZAÇÃO DO TRABALHO

O restante deste trabalho está organizado como segue. No Capítulo 2, são discutidos os conceitos fundamentais que representam a base para a pesquisa realizada, incluindo uma visão geral sobre SDN, programabilidade de rede e DPDK. Após, no Capítulo 3, é realizado um detalhamento dos experimentos executados para analisar a capacidade de escalabilidade do DPDK. Por fim, no Capítulo 4 são feitas as considerações finais e indicações de potenciais trabalhos a serem abordados em pesquisas futuras.

2 REFERENCIAL TEÓRICO

Neste capítulo são apresentados os conceitos fundamentais que constituem a fundação deste trabalho, incluindo SDN, P4 e DPDK. Além disso, são discutidos alguns esforços de pesquisa relacionados, onde se destaca o fator de inovação deste trabalho.

2.1 REDES DEFINIDAS POR *SOFTWARE*

O constante aumento do tráfego nas redes IP tradicionais, juntamente com o crescimento da complexidade dos serviços oferecidos pela internet à sociedade, tem demandado um processamento de pacotes em taxas cada vez mais elevadas. Além disso, a natureza dinâmica das demandas desses serviços exige que as redes se adaptem rapidamente, mantendo níveis adequados de qualidade de serviço (QoS) e atendendo de forma satisfatória às solicitações, utilizando eficientemente os recursos disponíveis na infraestrutura de rede.

No entanto, as redes IP tradicionais (OEVER; BERALDO, 2018) foram desenvolvidas de forma estática, incorporando a implementação de funcionalidades e protocolos de comunicação diretamente no hardware dos dispositivos. Como resultado, as infraestruturas das redes IP tornaram-se cada vez mais complexas e desafiadoras de gerenciar de maneira ágil e adaptativa às necessidades em constante evolução. Os dispositivos utilizados em redes IP tradicionais são fornecidos pelos fabricantes como hardware dedicado e de propósito específico, muitas vezes com custos elevados. Esses dispositivos de função dedicada tendem a dificultar futuras expansões da rede ou a incorporação de novos recursos, serviços e funcionalidades. Isso ocorre devido à necessidade de substituir ou adicionar diversos desses dispositivos de rede, o que resulta em um processo dispendioso e complexo.

Alguns destes dispositivos de função dedicada podem ser vistos abaixo:

- **Roteadores:** Independente do nível hierárquico no qual um roteador é utilizado em uma rede IP, sua função é principalmente estabelecer conexão entre uma ou mais redes de computadores, além de ser responsável por encaminhar os pacotes de dados, traçando a melhor rota para os pacotes de dados cheguem ao seu destino.

- **Switches:** Pode desempenhar funções em camada 2 (L2) de encaminhamento de dados, ou no caso processamento de quadros (frames), processamento de pacotes em camada 3 (L3). O posicionamento de switches em uma rede IP é, em linhas gerais, similar ao de um roteador, mas o seu uso tem o objetivo de expandir uma rede ou mesmo conectar dispositivos de uso final.

- **Firewall:** Realizam operações de análise de tráfego, filtrando pacotes para impedir o acesso indevido a informações e componentes. Por exemplo, componentes de firewall visam prevenir que infraestruturas de rede sofram ataques à segurança.

Os dispositivos de rede com função dedicada têm o seu plano de controle aliado ao plano de dados. A complexidade no uso de dispositivos com função dedicada está na

demanda do acesso individual em cada ativo de uma rede IP. Por exemplo, é necessário um certo esforço para atualizar seu sistema ou (re)configurar seu plano de controle. No entanto, agilidade, flexibilidade e segurança, são pontos de alta requisição além de largura de banda em uma rede IP tradicional, realçando o quanto uma rede é complexa e expondo sua carência em inovação, e essa falta de inovação e ossificação em redes IP tradicionais acabou sendo um gargalo para a implementação de novas tecnologias.

Neste contexto, SDN surge como uma tecnologia que apresenta características dinâmicas, gerenciáveis, econômicas e adaptáveis, tornando-a ideal para atender às demandas de aplicações atuais (NIKOLICH, 2016). A arquitetura de SDN promove o desacoplamento do plano de controle de rede das funções de tratamento do pacote, possibilitando que o controle de rede seja programado fora dos dispositivos finais, e que a infraestrutura subjacente seja abstraída das aplicações e serviços de rede.

O paradigma SDN mantém o plano de controle em dispositivo distinto do plano de dados, e idealiza a separação destes planos de forma codificada em uma interface aberta. De forma clara, o plano de controle é o responsável por definir o comportamento da rede, à medida que o plano de dados é o responsável por aplicar o comportamento definido, individualmente em cada pacote. A separação entre os planos de dados e controle permite desenvolver ações de (re)configuração, mitigação de anomalias na rede ou prover serviços. O plano de controle, contem a lógica de controle em uma rede SDN através de um controlador centralizado logicamente, e fisicamente distribuído. A Figura 1 apresenta os principais componentes de infraestruturas de rede operando sob o paradigma de SDN.

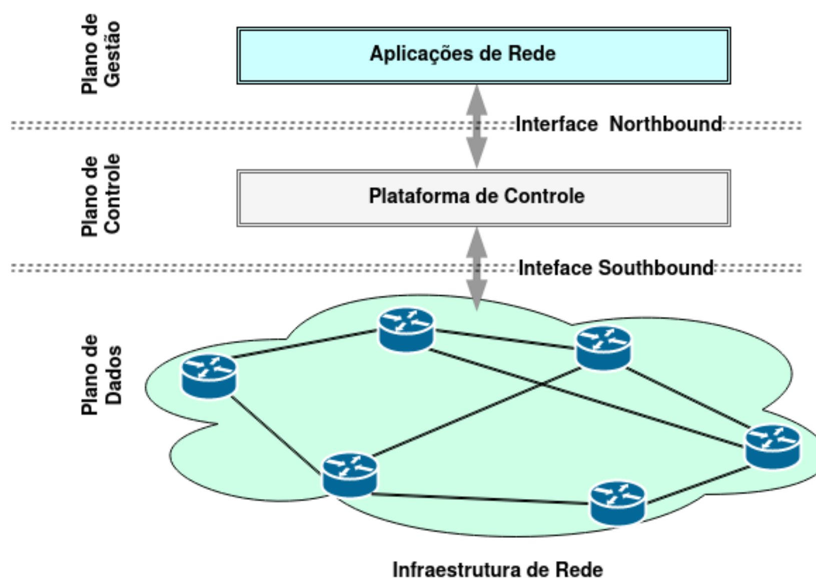


Figura 1 – Visão geral da arquitetura de SDN.

Dada a separação dos planos de dados e controle, infraestruturas SDN usam Interfaces de Programação de Aplicações (*Application Programming Interfaces* — APIs) para a comunicação dos componentes de rede. Neste contexto, o protocolo *OpenFlow* se

estabelece como uma importante interface no contexto de SDN, permitindo a comunicação entre dispositivos que mantém o plano de controle (chamados controladores) e dispositivos que executam o plano de dados. Os dispositivos de encaminhamento com suporte ao *OpenFlow* possuem tabelas de encaminhamento (também chamadas de tabelas de fluxo), preenchidas pelos controladores com instruções sobre as ações que devem ser tomadas em um determinado subconjunto de pacotes da rede.

O comportamento de dispositivos do plano de dados em uma rede SDN pode variar conforme o fluxo de pacotes. Deste modo, determinado dispositivo de encaminhamento pode atuar como um roteador (encaminhando pacotes), como firewall (filtrando pacotes), ou mesmo como balanceador de carga (equilibrando o processamento de pacotes de um determinada aplicação). Como redes SDN são projetadas visando alto nível de reconfigurabilidade, mudanças definidas por um controlador SDN nas tabelas de roteamento são aplicadas nos dispositivos de encaminhamento em tempo de execução. A Figura 2 ilustra algumas diferenças entre o modelo de redes tradicional e SDN.

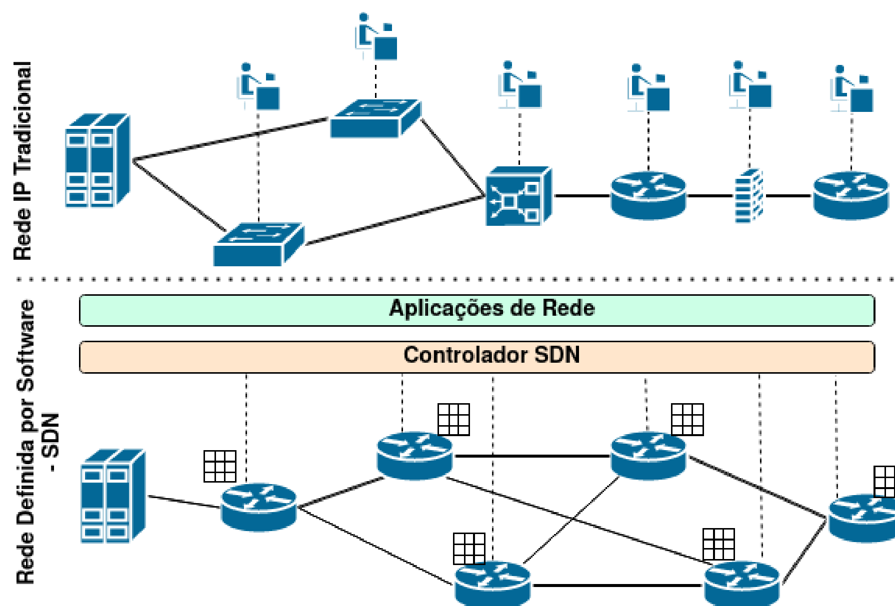


Figura 2 – Comparação entre o modelo de rede tradicional e SDN.

A arquitetura SDN é formada por quatro pilares fundamentais. No primeiro pilar, ocorre o desacoplamento dos planos de controle e dados. Em outras palavras, funções de controle são removidas de dispositivos de encaminhamento. No segundo pilar, as decisões de encaminhamento não são mais baseadas apenas no destino dos pacotes, mas sim no fluxo de rede, proporcionando maior flexibilidade nas rotas. No terceiro pilar, estabelece-se a presença de entidades do plano de controle, os chamados controladores SDN. No quarto pilar, que representa o principal princípio de SDN, a rede se torna programável por meio de artefatos de software executados no controlador SDN, e que interage com os dispositivos do plano de dados subjacentes. Essa capacidade de programação permite que a rede seja configurada e ajustada de forma dinâmica, adaptando-se rapidamente às necessidades

específicas das aplicações, tornando a SDN uma abordagem altamente flexível e eficiente para gerenciar redes modernas.

A programabilidade do plano de dados, concebível no modelo de SDN, possibilita a criação de infraestruturas de rede altamente flexíveis, adaptáveis e personalizadas, permitindo uma maior eficiência em atividades de gerenciamento. A alta reconfigurabilidade permitida pelo modelo de SDN torna-se especialmente útil em ambientes com requisitos complexos ou em constante mudança, como em *data centers* de nuvem e em redes de provedores de serviços.

2.2 LINGUAGEM P4

Os dispositivos que compõem a rede, como switches ou roteadores, geralmente são criados em uma abordagem chamada de *bottom-up*, onde decisões de projeto tomadas durante o desenvolvimento do equipamento determinam de forma estática quais funcionalidades serão suportadas. Tais dispositivos geralmente possuem chips chamados Circuitos Integrados de Aplicação Específica (*Application-Specific Integrated Circuit* — ASIC), compostos apenas por blocos de função fixa em seu pipeline interno de processamento. Chips compostos apenas por blocos de função fixa não permitem fácil reconfiguração de suas funcionalidades em tempo de execução, o que pode ser observado como uma limitação em termos de flexibilidade. A popularização do modelo de SDN motivou discussões sobre o formato *bottom-up*, já que a flexibilidade promovida pelo modelo de SDN é inerentemente limitada em cenários onde os dispositivos não podem ser reconfigurados em tempo de execução. Neste contexto, o formato *top-down* surge como uma alternativa promissora, onde o software determina o que deve ser realizado pelo hardware.

Em julho de 2014, um artigo intitulado *"Programming Protocol-Independent Packet Processors"* (em português: "Programando Processadores de Pacotes Independentes de Protocolos"), propôs um modelo de programação de dispositivos de rede embasada em 3 pilares: (i) (re)configuração dos dispositivos de rede em tempo de execução; (ii) dispositivos sem vínculos a qualquer protocolo de rede especifico; (iii) compatibilidade independente do alvo. Deste modo, foi descrita pela primeira vez a linguagem P4 (BOSSHART et al., 2014), como uma linguagem de programação que permite programar dispositivos de encaminhamento (plano de dados) em redes que empregam o modelo de SDN.

Com o uso da linguagem de programação P4, dispositivos programáveis apresentam duas características básicas: (i) definição do seu plano de dados no ato de carregamento do código P4 no dispositivo; e (ii) plano de controle e plano de dados se comunicam utilizando do mesmo canal com a possibilidade de modificar o programa P4 a qualquer momento pelo controlador SDN. O P4 teve como referência a arquitetura PISA (*Protocol-Independent Switch Architecture*), arquitetura de hardware com capacidade na casa de Tb/s de processamento de pacotes. Ao contrário dos tradicionais processadores ASICs, o PISA oferece um hardware baseado em um pipeline reconfigurável por meio de um modelo

Match-Action, provendo grande flexibilidade sem comprometer o desempenho.

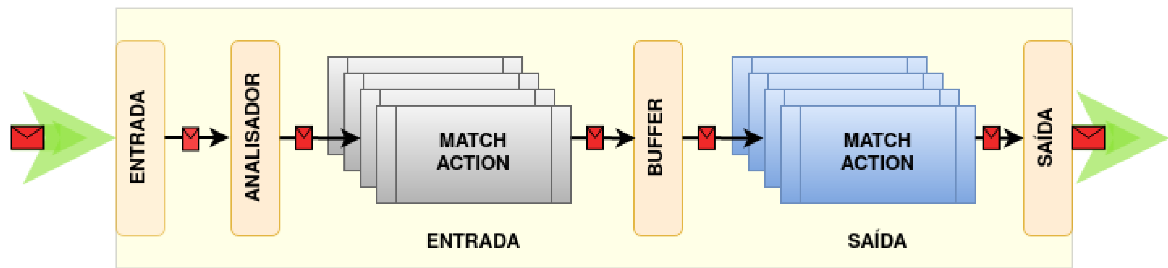


Figura 3 – Processamento na arquitetura PISA.

A linguagem P4 permite que o programador analise e manipule (se necessário) os cabeçalhos de qualquer protocolo conhecido ou personalizado em um fluxo de pacotes (por exemplo, Ethernet, IP e TCP). Conforme apresentado na Figura 3, quando um pacote alcança um dispositivo P4, como um switch ou roteador, ocorre uma etapa inicial de análise dos cabeçalhos, seguindo os critérios já estabelecidos no programa. Nesse processo, campos relevantes são identificados e extraídos. Após a análise inicial dos cabeçalhos, entram em ação as tabelas, que comparam os campos extraídos e executam as ações apropriadas conforme as instruções previamente definidas pelo controlador. As tabelas são preenchidas com informações do controlador, e é a partir dessa ação que ocorre o tratamento adequado dos pacotes. A sugestão na especificação da linguagem P4, é subdividir as tabelas em: (i) *ingress*, que contém a definição de portas de saída e fila para o pacote e (ii) *egress*, que contém as ações de manipulação de campos do cabeçalho do pacote. A primeira versão da linguem P4, denominada de P4-14, permite a programabilidade de aplicações para o plano de dados utilizando uma estrutura de cinco pontos:

1. **Headers:** Discrimina os campos com os dados internos do pacote para a etapa de análise realizada pelo parser.

2. **Parser:** Realiza a análise e extração dos valores contidos nos campos do pacote.

3. **Tables:** Executa o processamento do pacote com base em correspondências e ações.

4. **Actions:** Composto de primitivas de uso para uma determinada função (por exemplo, cópia, remoção ou adição de cabeçalho).

5. **Control:** Estabelece o controle de fluxo das tabelas.

Conforme aumentava a utilização da versão P4-14, diversas limitações na linguagem sobrevinham durante seu uso, como, por exemplo, um baixo suporte para modulação de programas, falta de meios para descrever vários alvos e arquiteturas e uma fraca semântica, foram as limitações de maior destaque. O suporte para diferentes alvos e arquiteturas demostrou ser a limitação mais importante, sendo essa a principal correção da versão

subsequente da linguagem, a versão P4-16. A separação das especificações de arquitetura do corpo da linguagem principal torna a linguagem independente da arquitetura. Estrutura, recursos e interfaces do pipeline estão encapsulados em uma descrição de arquitetura, e no caso do alvo possuir funções específicas em sua arquitetura, essas são acessíveis por meio de bibliotecas fornecida pelo fabricante do dispositivo. Os principais componentes da linguagem, são estruturados em um pequeno conjunto de instruções, por meio de uma biblioteca utilizada na maioria dos programas P4.

Os dispositivos programáveis, estão disponíveis em hardware especializado e também em software. Esses apresentam um pipeline de processamento de pacotes que consistem em componentes P4 programáveis e de função fixa. A exata estrutura desses pipelines é específica e descrita por um modelo de arquitetura P4 correspondente para esse dispositivo. Os algoritmos em P4 serão executados pelos componentes programáveis no pipeline desses dispositivos. No entanto, como será sua interação com os componentes de função fixa, deve ser determinado pelo programador durante o desenvolvimento da aplicação P4. O programa P4 deve ser implementado considerando um modelo especifico de arquitetura, e essa composição constitui uma aplicação para o plano de dados completa.

Dessa forma, a versão P4-16 utiliza um conceito de camada intermediária entre a linguagem de programação e os alvos, por meio de arquiteturas P4. Essas arquiteturas são modelos de referência para os recursos programáveis, e a visão lógica do pipeline P4 de processamento do alvo almejado. Ao desenvolver uma aplicação P4 para uma determinada arquitetura, esse mesmo código pode ser aplicado em outros dispositivos programáveis de mesma arquitetura P4. Externs é como se denomina das funcionalidades que uma arquitetura P4 pode fornecer de modo adicional, além das funções padrão do núcleo da linguagem, bem como, registradores, medidores, computação de hash, geradores de números aleatórios, contadores de bytes e pacotes, dentre outros. Podemos utilizar essas funcionalidades externas instanciando de forma explicita através de seu método construtor. O pipeline de processamento P4 permite apenas a manipulação do cabeçalho dos pacotes, porém funções externas podem operar na carga útil (dados dos pacotes).

2.2.1 ARQUITETURAS DA LINGUAGEM P4

Na linguagem P4-16 existe uma série de arquiteturas que permitem a implementação e o processamento eficiente dos pacotes de rede em alvos P4 de hardware ou software. Essas arquiteturas são projetadas para atender a diferentes requisitos e a portabilidade entre os dispositivos programáveis. As arquiteturas de relevância, atualmente, apresentam-se a seguir.

2.2.1.1 V1MODEL

A arquitetura v1model consiste em um dos principais modelos de processamento de pacotes disponíveis na linguagem P4-16. Ela estabelece um conjunto abstrato de etapas

e componentes para o processamento de pacotes em dispositivos de rede programáveis, permitindo aos desenvolvedores definirem de forma flexível o comportamento dos pacotes em uma rede definida por software. Em destaque as principais características da arquitetura.

- Divisão em Etapas de Processamento: A arquitetura v1model divide o processamento de pacotes em duas etapas principais: ingress (entrada) e egress (saída). Essa divisão permite que os pacotes sejam tratados de forma específica em cada etapa, o que é essencial para a implementação de funcionalidades complexas e personalizadas.
- Tabelas de Fluxo (Match-Action Tables): A arquitetura v1model enfatiza o uso de tabelas de fluxo (match-action tables) para o processamento de pacotes. As tabelas de fluxo contêm regras que descrevem como os pacotes devem ser manipulados com base em suas características, como endereços, MAC, IP, portas etc. Isso permite que os desenvolvedores definam o comportamento dos pacotes de maneira granular e flexível.
- Processamento de Pacotes Independente de Protocolo: Ao contrário de outras arquiteturas que podem ser específicas de certos protocolos de rede, a v1model é projetada para ser independente de protocolo. Isso significa que ela não se baseia em protocolos de rede específicos, permitindo que os desenvolvedores a utilizem em diferentes contextos sem restrições.
- Flexibilidade na Definição de Ações: A arquitetura v1model permite que os desenvolvedores definam ações personalizadas que podem ser aplicadas aos pacotes. Isso inclui a criação de ações predefinidas e a capacidade de definir ações personalizadas, o que amplia as possibilidades de processamento de pacotes.

2.2.1.2 PORTABLE SWITCH ARCHITECTURE (PSA)

A arquitetura Portable Switch Architecture (PSA) é um modelo significativo dentro do contexto da linguagem P4-16. Seu desenvolvimento teve como objetivo ampliar a portabilidade e a flexibilidade dos programas P4, permitindo que esses programas sejam executados em uma variedade de dispositivos de rede, independentemente das peculiaridades de hardware subjacentes. As características de destaque na arquitetura PSA tem como:

- Abstração de Hardware: A principal característica da arquitetura PSA é a abstração de detalhes específicos de hardware nos dispositivos de rede. Isso possibilita que os programas P4 sejam escritos sem a necessidade de levar em consideração as particularidades técnicas dos dispositivos onde serão implantados.
- Interface Padronizada: A PSA estabelece uma interface padronizada que permite aos programas P4 interagirem com o hardware de maneira uniforme em diferentes

dispositivos de rede. Isso assegura que os programas P4 escritos para a arquitetura PSA possam ser executados em diversos dispositivos, tornando a programação mais flexível e reutilizável.

- Versatilidade: A arquitetura PSA foi projetada para se adaptar a uma variedade de dispositivos de rede, desde comutadores e roteadores até equipamentos especializados. Isso capacita os desenvolvedores a criar programas P4 que funcionem de maneira consistente em uma ampla gama de dispositivos.

- Fomento à Inovação: Ao abstrair o hardware, a PSA abre espaço para a experimentação e inovação. Desenvolvedores podem se concentrar na criação de novas funcionalidades e serviços de rede, em vez de se preocuparem com a implementação específica de hardware.

2.2.1.3 PORTABLE NIC ARCHITECTURE (PNA)

A arquitetura PNA é um modelo essencial no contexto da linguagem P4. Ela oferece uma estrutura padronizada e capacidades comuns para dispositivos de controlador de interface de rede (NIC) que processam pacotes entre uma ou mais interfaces e um sistema host. As principais características da arquitetura PNA são:

- Pipeline Programável: A arquitetura PNA incorpora um pipeline programável que pode ser utilizado para criar diferentes "caminhos de pacotes"entre as diversas portas do dispositivo (como interfaces de rede ou o próprio sistema host ao qual está conectado).

- Biblioteca de Tipos e Externs P4-16: A PNA inclui uma biblioteca de tipos, como metadados intrínsecos e padrão, e externs P4-16, como contadores, medidores e registradores. Isso amplia a funcionalidade e as capacidades do programa P4 ao ser executado em dispositivos compatíveis com a arquitetura PNA.

- Portabilidade e Compatibilidade: A PNA foi projetada para modelar recursos comuns de uma ampla classe de dispositivos NIC. Ela estabelece diretrizes de codificação e APIs padronizadas, permitindo que os desenvolvedores criem programas P4 portáteis que funcionem de maneira consistente em vários dispositivos NIC que estejam em conformidade com a PNA.

2.2.1.4 COMPILADOR P4C

Aplicações desenvolvidas em linguagem P4 têm a necessidade da tradução das instruções de *software* para um código de máquina (binário) especifico para o dispositivo alvo. Essa tradução é executada por compiladores P4. Compiladores P4 utilizam em sua maioria do modelo de duas camadas que consiste em um front-end com suporte genérico e

um back-end especifico para o alvo P4. Independente do alvo P4, a análise semântica e sintática do programa P4 é realizada pelo front-end do compilador e transformada para uma representação intermediaria (IR). O back-end especifico do alvo P4 executa esse IR gerado pelo front-end, o traduzindo essa representação para o código binário específico do alvo P4, a Figura 4 ilustra esse processo.

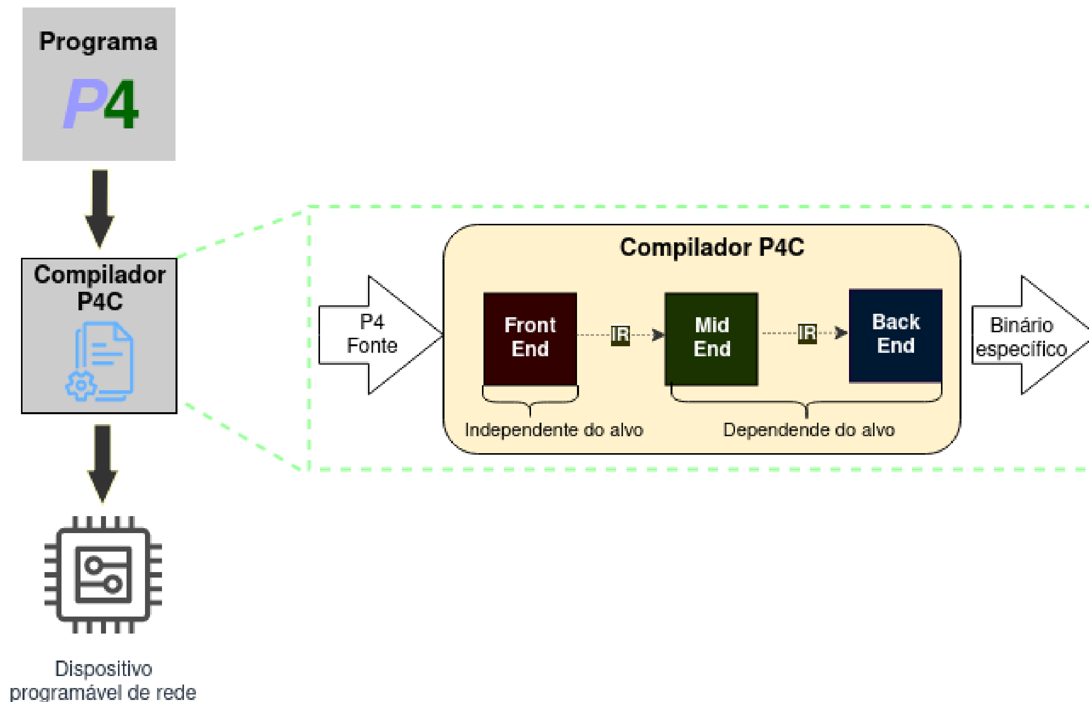


Figura 4 – Arquitetura de compiladores P4.

A ferramenta de compilação referência para a linguagem de programação P4 é o compilador P4C. O P4C aceita códigos em linguagem P4 nas versões P4-14 e P4-16. O P4C possui back-ends de referência para alvos P4 baseado em software como o bmv2, eBPF e uBPF, além de um back-end para testes. Alvos P4 de fabricação proprietária tem seu compilador desenvolvido e fornecido pelo seu fabricante, mantendo o seu front-end com suporte genérico, visando garantir a compatibilidade da linguagem P4.

2.3 DPDK

O Kit de desenvolvimento para plano de dados (Data Plane Development Kit - DPDK) (OVERVIEW, 2023a) é um projeto totalmente de código aberto que opera no espaço do usuário do sistema operacional. Consiste em um projeto multi-fornecedor e multi-arquitetura, e visa alcançar alto desempenho de E/S e atingir altas taxas de processamento de pacotes, alguns dos recursos mais importantes na área de rede. O DPDK ignora as camadas mais pesadas da pilha de rede do Kernel do Linux e se comunica diretamente com o hardware de rede, e por esse motivo, é possível atingir velocidades superiores se comparado com aplicações comuns do kernel.

Outro fator que advoga em direção a um bom desempenho, consiste no uso de grandes fatias de memória, essa adaptada para rápidas operações em massa, que armazena os pacotes em filas sem nenhum bloqueio. Ao usar páginas grandes (de 2MB até 1GB de tamanho), é necessário um número menor de paginação de memória quando comparado com o uso de páginas de memória padrão (lembrado que muitas plataformas usam apenas 4k). Como resultado, o número de erros do Translation Lookaside Buffer (TLB) é reduzido significativamente, proporcionando um aumento considerável no desempenho. Outro fator que eleva seu desempenho consiste em otimizações de baixo nível que são realizadas no código, algumas delas relacionadas ao alinhamento das linhas do cache de memória, visando atingir o uso ideal do cache, pré-busca e assim por diante.

O DPDK iniciou como um pequeno projeto de software na Intel, e algum tempo depois evoluiu para um projeto líder de código aberto sob responsabilidade da Linux Foundation. Além disso, a utilização de hardwares de uso geral por sistemas de infraestrutura de redes, computação em nuvem e telecomunicações, impulsionou sua utilização. Outro ponto que vale chamar a atenção consiste na evolução de plataformas de servidores, que foi diretamente impulsionada pelo crescimento de processadores com múltiplos núcleos.

O design mais comum de uso por servidores é o de soquete duplo, ou seja, 2 CPU multi-core. No entanto, outros componentes em silício colaboram com a evolução dos servidores, como exemplo, de memórias e interfaces de rede, onde ocorre aumento considerável em capacidade. Outro atrativo está no custo-benefício dos atuais servidores em concentrar vários CPUs e uma grande quantidade de memória em um único dispositivo. Esse rápido desenvolvimento tecnológico torna a plataforma de servidor a opção preferencial para implementar a infraestrutura definida por software (SDI), uma plataforma comum para fornecer tarefas de computação, rede e armazenamento. Data centers definidos por software contam com o uso de uma infraestrutura de modo virtual, sendo essas ferramentas em software desempenhando serviços antes realizado por hardware especifico como, switchs, firewall, balanceadores de carga, routers e outros.

O modelo SDI de data center, exige maior desempenho, agilidade e escalabilidade, no entanto, limitações do sistema operacional do servidor impõem limitações para essas demandas. Atualmente os servidores de um, data center podem disponibilizar de NICs de capacidade de até 100 Gbps, sendo o mais comum interface de rede de 10 Gbps/25 Gbps. Um servidor que possui NICs que suportam uma entrada/saída de pacotes em uma taxa linear de até 100 Gbps é gerenciando pelo sistema operacional (OS). Portanto, é de responsabilidade do sistema operacional gerenciar todos os recursos do sistema e fornecer um ambiente de execução para aplicações, em um servidor de data center isso engloba os serviços, sistemas e aplicações do qual utilizam dos recursos deste servidor. Desta forma, obter um alto desempenho em um alto trafego de pacotes implica em sobrecarga do sistema operacional, além de considerar o modelo de interrupções aplicado no processamento de pacotes.

O modelo de interrupções de processamento de pacotes feito pelo sistema operacional é forma que o OS desempenha a transferência de dados entre a interface de rede e o software da aplicação. O processamento de pacotes no modo de interrupções tem as seguintes etapas.

- **Interrupções e pilha de rede:** Quando um pacote chega à interface de rede de um computador, o controlador de rede dispara uma interrupção para o sistema operacional para notificar a chegada de um novo pacote. As interrupções são mecanismos de hardware que permitem que dispositivos externos, como placas de rede, solicitem atenção do processador do computador. Quando ocorre a interrupção, o sistema operacional interrompe a execução da aplicação atual e passa o controle para o tratamento da interrupção. Essa interrupção envolve uma série de passos, incluindo a identificação da interface de rede que gerou a interrupção, a leitura do pacote da interface e o encaminhamento do pacote para a pilha de rede do sistema operacional.

- **Pilha de rede:** A pilha de rede é uma coleção de protocolos e funções usados pelo sistema operacional para processar e manipular pacotes de rede. Ela é organizada em camadas, onde cada camada é responsável por tarefas específicas relacionadas ao processamento de pacotes. Quando o pacote chega à pilha de rede, passa por várias etapas de processamento, como análise do cabeçalho do pacote para determinar o protocolo usado, verificação de erros, roteamento, tradução de endereços (NAT), controle de congestionamento, etc. Dependendo do protocolo e do destino do pacote, este pode ser entregue à aplicação de destino diretamente ou repassado para outra camada para processamento adicional.

- **Interrupções de processamento de pacotes:** O sistema operacional implementa técnicas para minimizar o tempo gasto no tratamento de pacotes de rede e otimizar o desempenho geral do sistema. Uma das técnicas importantes é a chamada "interrupção de processamento de pacotes"(packet processing offload), que envolve transferir parte do processamento dos pacotes da CPU para componentes de hardware especializados quando possível.

Dessa forma, podemos dizer que o modelo de interrupção de processamento de pacotes é a modo com que o sistema operacional orquestra a fatia de tempo do processamento de pacotes pela CPU no hardware subjacente. Esse pesado processo realizado pelo sistema operacional no processamento de pacotes de rede, sem dúvida, implica para um maior latência e baixa taxa de transferência. O DPDK surge como uma opção para agregar desempenho em latência e taxa de transferência no processamento de pacotes, utilizando recursos de hardware de arquitetura x86.

A necessidade de suportar um tráfego de pacotes de entrada/saída em taxa de linha compatível com as, atuais interface de rede (de 10Gbps até 100Gbps) do mercado fez

com que o DPDK fosse utilizado em diversas frentes, como em storages de alto desempenho, comunicação entre contêineres, virtualização, funções virtuais de rede (NFV), etc. O DPDK é conhecido como um acelerador de pacotes com uma tecnologia de rede de desvio de kernel, ou seja, ele ignora a pesada pilha de rede do sistema operacional e sua execução é em espaço do usuário (user-space). Operar no espaço de usuário significa que ele evita o uso tradicional do modo de processamento de pacotes de rede feito pelo sistema operacional (modo de interrupção) visando obter um desempenho superior.

Ao adotar o modo de espaço de usuário, o DPDK contorna as camadas do sistema operacional que seriam normalmente concorridas pelo tráfego de rede, reduzindo a sobrecarga de chamadas ao sistema e proporcionando um processamento mais eficiente. As etapas realizadas pelo DPDK no modo de espaço do usuário são.

- **Inicialização:** O DPDK é inicializado pela aplicação de usuário (por exemplo, um switch, roteador ou firewall) através da configuração dos dispositivos de rede de entrada/saída e da preparação de buffers de recepção e transmissão.

- **Gerenciamento de Memória:** O DPDK utiliza pools de memória pré-alocada (memory pools) para evitar fragmentação e para otimizar o acesso aos buffers de pacotes. Os pools de memória são criados durante a inicialização e são usados para armazenar os pacotes recebidos e transmitidos.

- **Recebimento de Pacotes:** Quando um pacote chega na interface de rede, o DPDK o captura diretamente da NIC usando mecanismos específicos de acesso direto à memória (Direct Memory Access - DMA). Em seguida, o pacote é colocado em um buffer apropriado do pool de memória.

- **Processamento de Pacotes:** Os pacotes são processados de forma eficiente pelo DPDK, utilizando múltiplos núcleos de CPU. Cada núcleo é designado para processar pacotes de uma fila específica, permitindo paralelismo e distribuição da carga.

- **Transmissão de Pacotes:** Os pacotes processados são transmitidos pelo DPDK de volta para a NIC, utilizando novamente mecanismos de acesso direto à memória (DMA). O DPDK evita copiar os pacotes para os buffers de transmissão, reduzindo ainda mais a sobrecarga.

- **Tratamento de Interrupções:** Enquanto em modo de usuário, o DPDK lida com as interrupções de forma diferente em comparação ao modo kernel. Em vez de utilizar o mecanismo de interrupções padrão do sistema operacional, o DPDK utiliza threads de interrupção (interrupt threads) para gerenciar e tratar as interrupções da NIC, minimizando o tempo gasto em interrupções e melhorando o desempenho.

De modo resumido, o DPDK opera no espaço do usuário, permitindo o acesso direto ao hardware da NIC e o processamento eficiente de pacotes de rede. Essa abordagem

oferece um desempenho significativamente melhor em comparação com as pilhas de rede tradicionais do sistema operacional, tornando-o uma opção onde aplicações requerem alto rendimento e latência baixa, de versões em software de serviços como, switches, roteadores, gateways, firewalls e outras soluções de rede de alto desempenho.

O DPDK permite que aplicações de rede de alto desempenho utilizem ao máximo a capacidade de processamento dos atuais processadores. Disponibilizando uma estrutura de desenvolvimento para o plano de dados contendo um conjunto de bibliotecas e drivers otimizados para processamento de pacotes em servidores. O DPDK mimetiza a maioria das funções de uma rede em software e seu funcionamento tem como base uma arquitetura de pilha de software para o rápido processamento de pacotes.

- **Aplicações:** Sua principal responsabilidade é definir a lógica de processamento de pacotes, permitindo que os dados sejam manipulados de forma eficiente e rápida. Essas aplicações interagem diretamente com as bibliotecas do DPDK, aproveitando assim as funcionalidades de aceleração de hardware disponíveis. Com isso, aplicações podem tirar proveito das otimizações de desempenho oferecidas pelo framework. Isso inclui a capacidade de acessar diretamente recursos de hardware e de usar as técnicas de descarregamento de NIC, possibilitando um processamento mais ágil e eficaz dos pacotes de rede.

- **Bibliotecas** O DPDK oferece um conjunto abrangente de bibliotecas que desempenham um papel fundamental ao encapsular funcionalidades de baixo nível necessárias para interagir diretamente com o hardware de rede. Essas bibliotecas são componentes-chave que permitem que aplicações de rede se beneficiem das otimizações de desempenho e aceleração proporcionadas pelo DPDK.

  As principais bibliotecas incluem:

  - Librte_ethdev: Essa biblioteca fornece funções para inicializar e configurar dispositivos Ethernet, bem como para realizar operações de leitura e escrita nos buffers de dados recebidos e enviados
  - Librte_mbuf: Essa biblioteca é responsável pelo gerenciamento dos buffers de pacotes, permitindo a manipulação eficiente dos dados durante o processamento.
  - Librte_ring: Essa biblioteca oferece uma estrutura de dados em anel (ring) de baixa latência, amplamente utilizada para comunicação e sincronização entre threads.
  - Librte_timer: Essa biblioteca oferece a funcionalidade de temporização, sincronizando as diferentes tarefas nas diferentes threads.
  - Librte_mempool: Essa biblioteca é responsável por alocar e liberar de forma eficiente os buffers de memória para pacotes.

- **Drivers PMD:** Os drivers PMD (Poll Mode Drivers) são componentes específicos do DPDK projetados para se adaptar a diferentes dispositivos de rede. Sua função essencial é proporcionar uma interface de alto desempenho que permite o acesso otimizado às funcionalidades de hardware desses dispositivos suportados. Ao trabalharem em modo de polling, esses drivers possibilitam que aplicações interajam de forma mais eficiente com o hardware de rede, evitando a sobrecarga causada por interrupções frequentes.
- **NICs:** As placas de interface de rede desempenham um papel fundamental na arquitetura do DPDK, pois são os dispositivos responsáveis por conectar o sistema ao ambiente de rede. Para que o DPDK possa operar eficientemente, é essencial que essas NICs sejam compatíveis e suportem o acesso direto ao plano de dados. Ao serem compatíveis com o DPDK, as NICs tornam-se capazes de trabalhar em conjunto com os drivers PMD específicos, otimizados para aproveitar ao máximo suas capacidades de processamento. Esses drivers PMD permitem que aplicações de rede interajam diretamente com o hardware do NIC, evitando camadas desnecessárias de abstração e proporcionando um acesso direto e eficiente aos dados.

Em resumo, o DPDK oferece uma arquitetura de software altamente eficiente para acelerar o processamento de pacotes em servidores. Sua abordagem de pilha de software otimizada, combinada com drivers PMD e o acesso direto ao hardware em NICs compatíveis, possibilita que os pacotes de rede possam ser recebidos e enviados de maneira mais rápida e eficiente, sem a sobrecarga de interrupções realizada pelo sistema operacional.

2.4 TRABALHOS RELACIONADOS

A evolução das redes de comunicação é uma necessidade premente, uma vez que o aumento constante do tráfego de dados e a complexidade crescente dos serviços disponibilizados pela internet impõem desafios significativos. As redes tradicionais, que incorporam a implementação de funcionalidades e protocolos de comunicação diretamente no hardware dos dispositivos, têm se mostrado inadequadas para lidar com essas demandas em constante evolução.

A abordagem tradicional resulta em infraestruturas de rede complexas e de difícil gerenciamento. Os dispositivos utilizados nesse contexto são, em sua maioria, fornecidos pelos fabricantes como hardware dedicado e de propósito específico, o que acarreta custos elevados e dificuldades para futuras expansões ou atualizações da rede. A necessidade de substituir ou adicionar dispositivos específicos para incorporar novos recursos ou serviços representa um desafio adicional, tanto em termos de custo quanto de complexidade operacional.

Nesse sentido, a programabilidade de redes emerge como uma solução promissora. Essa abordagem envolve a separação das funções de controle e encaminhamento de dados,

permitindo que o plano de controle seja programado e ajustado dinamicamente, enquanto o plano de dados permanece inalterado. Isso proporciona uma maior flexibilidade na configuração e gestão das redes, permitindo a introdução de novos serviços, otimização de roteamento e adaptação rápida às mudanças nas demandas de tráfego.

As tecnologias como P4, eBPF e DPDK desempenham um papel fundamental nesse cenário. O P4 permite a definição personalizada do processamento de pacotes, o eBPF possibilita a inserção de código executável diretamente nos dispositivos de rede, e o DPDK oferece um alto desempenho no processamento de pacotes. Essas tecnologias abrem caminho para uma programabilidade mais eficaz e personalizada das redes, contribuindo para um melhor desempenho e eficiência. No entanto, apesar dos avanços significativos proporcionados por essas tecnologias, ainda há desafios a serem superados. A definição clara dos limites de desempenho do DPDK, como mencionado anteriormente, é um exemplo. Além disso, é essencial considerar a escalabilidade e a interoperabilidade dessas tecnologias em ambientes de rede complexos e em constante evolução.

No estudo realizado por Begin et al. (BEGIN et al., 2018), foi apresentado um modelo analítico de filas para avaliar o desempenho de um vSwitch DPDK. Nesse contexto, o vSwitch é representado como um sistema de polling complexo, no qual os pacotes são processados em lotes por núcleos da CPU. Cada núcleo processa vários pacotes de suas filas de entrada antes de passar para a próxima etapa. Em resumo, os autores desenvolveram um framework que divide o sistema de polling em subsistemas de enfileiramento, cada um correspondendo a um núcleo específico da CPU. Essa abordagem contribui para a redução da complexidade do modelo associado ao vSwitch baseado em DPDK.

Por outro lado, Shen et al. (SHEN et al., 2018) propuseram uma alternativa aos dispositivos de teste de segurança de rede proprietários. Eles abordaram um método de otimização do desempenho de E/S de rede virtualizada em uma plataforma de hardware genérico. Isso foi alcançado por meio da Virtualização de I/O de Raiz Única (SRIOV), que divide a placa física de rede em funções de rede virtual (VF). Essas VFs podem ser atribuídas a máquinas virtuais (VM) diferentes. Utilizando a tecnologia SRIOV e o Data Plane Development Kit, os pesquisadores demonstraram a capacidade de compartilhar e virtualizar uma única placa de rede física por meio do hardware, atingindo taxas de transferência de até 10 Gbps de E/S de rede.

Vladislavic et al. (VLADISLAVIć; HULJENIć; OžEGOVIć, 2017) combinaram o Open vSwitch (OVS) com o DPDK para melhorar o desempenho e a eficácia no processamento de pacotes. Eles implementaram essa combinação em uma rede de contêineres, explorando dois cenários distintos. No primeiro cenário, executaram dois contêineres em um único host e utilizaram o OVS-DPDK como uma ponte virtual para conectá-los. No segundo cenário, configuraram o sistema em dois hosts distintos. Em ambos os casos, realizaram medições de taxa de transferência, uso da CPU e avaliação do desempenho do OVS e do OVS-DPDK.

Kourtis et al. (KOURTIS et al., 2015) realizaram uma avaliação das Inspeções Profundas de Pacotes (DPI) do tráfego de rede na forma de uma função de rede virtualizada (VNF). Eles destacaram os benefícios do uso do SR-IOV com DPDK para VNFs de alto desempenho. Os resultados obtidos demonstraram que a combinação de SR-IOV e DPDK resulta em um aumento significativo na taxa de transferência de pacotes, em comparação com o processamento de pacotes utilizando a pilha de rede do kernel Linux nativo.

Apesar da crescente quantidade de estudos que exploram o potencial e as funcionalidades do DPDK, ainda persiste uma lacuna significativa em relação à definição clara dos limites de desempenho que o DPDK pode oferecer. Com base nessa observação, o presente trabalho se propõe a realizar uma análise de escalabilidade do DPDK durante a execução de aplicações de rede. Essa análise busca identificar e discutir as limitações técnicas existentes, bem como os potenciais pontos de melhoria. O objetivo não é apenas proporcionar um panorama mais preciso do estado atual da tecnologia, mas também abrir caminho para futuras iniciativas e pesquisas na área. A compreensão aprofundada das capacidades e desafios do DPDK é fundamental para o desenvolvimento contínuo de infraestruturas de rede de alto desempenho.

Portanto, a pesquisa e o desenvolvimento contínuo na área de programabilidade de redes são fundamentais para atender às crescentes demandas por serviços de comunicação de alta qualidade. A capacidade de programar e personalizar a infraestrutura de rede de acordo com requisitos específicos é crucial para o sucesso das redes modernas. Nesse contexto, a análise de escalabilidade do DPDK e a compreensão de suas limitações e potenciais melhorias desempenham um papel importante no avanço da tecnologia e na construção de infraestruturas de rede mais eficientes e adaptáveis.

3 ANÁLISE DE ESCALABILIDADE DO DPDK

Nesta seção, analisamos a escalabilidade do DPDK através da condução de experimentos que avaliam a latência e vazão[1] de aplicações P4 em um pipeline DPDK. Primeiramente, nós detalhamos a metodologia adotada (§3.1). Após, discutimos os resultados obtidos (§3.2).

3.1 METODOLOGIA

3.1.1 AMBIENTE DE TESTES

Em nossos experimentos, utilizamos um ambiente composto por dois servidores Dell PowerEdge T440. Cada servidor estava equipado com um processador Intel(R) Xeon(R) Silver 4214R CPU @ 2.40GHz e 32 GB de RAM. Um dos servidores executa o sistema operacional Ubuntu 20.04.5 LTS e utiliza uma placa de rede Intel(R) Ethernet Connection XL710 10GbE. Este servidor atuou como dispositivo programável, onde as aplicações P4 foram executadas. O segundo servidor é responsável por gerar o tráfego de pacotes contra o servidor de teste. O servidor de geração de tráfego utiliza de uma placa de rede Intel Ethernet Connection 82599ES 10-Gigabit SFI/SFP+.

Adotamos uma aplicação P4 disponibilizada no GitHub-P4Lang como base e adaptamos o arquivo para atender às necessidades de nossos cenários de avaliação. As aplicações em P4 utilizadas passaram por compilação utilizando a ferramenta P4C juntamente com o backend DPDK. Durante a compilação do arquivo P4, as instruções de controle do plano de dados foram convertidas para um formato compatível com o carregamento e execução em alvos DPDK. A utilização do P4C com o backend DPDK permite que os programas P4 sejam carregados e executados em um pipeline DPDK. No entanto, é importante destacar que essa conversão não é totalmente funcional, pois existem recursos de linguagem, arquitetura e limitações específicas do alvo de destino que podem impedir a aplicação de certas aplicações P4 em alvos DPDK.

3.1.2 MÉTRICA APLICADA

Realizamos uma análise completa do desempenho de programas PDP em um pipeline DPDK, levando em conta propriedades como latência e taxa de transferência, além de possíveis limitações em desempenho. Para isso, elaboramos diversos cenários de teste, levando em consideração os aspectos:

- (i) Número de operações em registradores:
- Cenário 1-a - Ler, Escrever, Ler/Escrever em registrador de tamanho 32 bits, e trafego com pacotes de tamanho 64B.

[1] No contexto deste trabalho, os termos “vazão” e “taxa de transferência” são usados de forma intercambiável.

- Cenário 1-b - Tráfegos com pacote de tamanho 64B, 128B, 256B, 512B, 1024B e 1500B, registradores com 32 bits de espaço, ação Ler/Escrever.
- Cenário 1-c - Registradores com espaços 16, 32 e 64 bits, ação de Leitura, em um tráfego intenso de pacotes com 64B de tamanho.
- (ii) Número de recirculações de pacotes:
- Cenário 2 - Recirculações de pacotes variando do seu tamanho de 64B, 128B, 256B, 512B, 1024B e 1500B.
- (iii) Número de acessos a tabelas de correspondência e ação:
- Cenário 3 - Pacotes de tamanho 64B, 128B, 256B, 512B, 1024B e 1500B, correspondido em sequência por tabelas match-action.
- (iv) Número de funções criptográficas aplicadas:
- Cenário 4 - Trafego de pacotes com 64B de tamanho, com chamadas consecutivas de funções criptográficas.
- (v) Número de operações aritméticas realizadas:
- Cenário 5 - Consecutivas operações aritméticas, com tráfego de pacotes com 64B de tamanho.

Realizamos várias sessões de testes (STs), e em cada sessão, todos os cenários são executados possuindo as mesmas configurações e métricas. No entanto, em cada ST realizada possui o número de lcores(AFFINITY, 2023) de alocação distinto. Com isso almejamos avaliar o impacto dessas variações sobre o desempenho de aplicações em pipeline DPDK. Conduzimos as sessões de testes, aplicando os cenários nas diferentes STs elaboradas da seguinte forma:

- ST(i): 2 lcores alocados.
- ST(ii): 4 lcores alocados.
- ST(iii): 8 lcores alocados.
- ST(iv): 16 lcores alocados.

Nos resultados obtidos para cada cenário nas diferentes sessões de testes, realizamos comparações entre resultados de cenários semelhantes ao ST(i), focando especialmente nas métricas de latência e taxa de transferência dos programas PDP avaliados. Dessa forma, tomamos como base um programa P4 de exemplo do repositório GitHub P4lang(LPM, 2023). O código do programa P4 base contém o mínimo de instruções para realizar o

encaminhamento de pacotes em camada 2 e 3 entre portas físicas de rede. Para cada aspecto de avaliação, alteramos o código P4 de referência, modelando-o com as propriedades de avaliação desejadas. Ao finalizar cada modelagem de código, compilamos o nosso código-fonte P4 resultante, mediante uso do compilador P4C(LANGUAGE, 2023) utilizando o backend DPDK(API, 2023), o que nos resulta em um arquivo de extensão ".spec", compatível com pipeline DPDK.

O emprego dos nossos códigos em pipeline DPDK são realizados por meio de CLI (Interface de Linha de Comando) contendo os parâmetros da camada de abstração do ambiente (EAL)(OVERVIEW, 2023b) mediante do caminho do arquivo obtido, e consequente a compilação do código-fonte P4. A efetiva execução dos cenários de teste é obtida via uma chamada CLI da aplicação pipeline DPDK(APPLICATION, 2023), acompanhado de parâmetros EAL obrigatórios, como alocação de hardware (cores) junto do caminho do roteiro CLI. Ao longo dos experimentos, a aplicação em execução no pipeline processa os pacotes de um tráfego de rede gerados pela ferramenta MooGen(GENERATOR, 2023).

A coleta dos dados sobre taxa de transferência e latência foi realizada em tempo de execução para cada um dos cenários de teste. Os valores de latência (a diferença entre os timestamps de entrada e saída do pacote processado pelo PDP) e os valores de throughput em pacotes por segundo são oferecidos diretamente pela ferramenta MooGen.

3.2 RESULTADOS

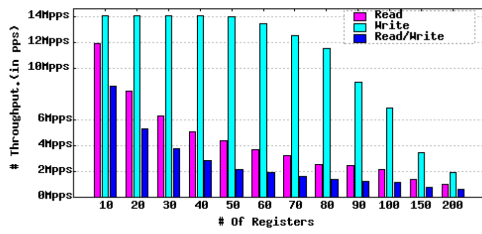
Esta seção apresenta e discute os resultados obtidos durante a análise de escalabilidade do DPDK. Inicialmente, avaliamos o desempenho do DPDK em uma configuração de teste base (isto é, ST(i)). Na sequência, comparamos os resultados com as demais configurações de teste (isto é, ST(ii)–ST(iv)).

3.2.1 DESEMPENHO COM CONFIGURAÇÃO ST(I)

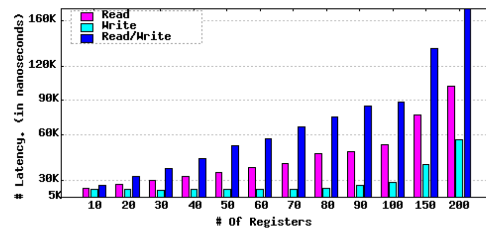
3.2.1.1 DESEMPENHO DURANTE OPERAÇÕES DE LEITURA E ESCRITA

Iniciamos nossa análise dos resultados tratando do custo da execução de várias operações sobre os registradores pelo programa PDP em pipeline DPDK. Entre cada um dos cenários de ST(i), variamos o número de operações realizadas em sequência pela nossa aplicação no pipeline de 10 a 200 registros, além de variarmos o tamanho das palavras de 16 a 64 bits. Julgamos que os registradores estão localizados no pipeline de entrada, podendo serem lidos, escritos ou lidos/escritos. Com isso, buscamos identificar algum impacto na latência e taxa de transferência, além de alguma possível limitação existente. A figura 5 ilustra a taxa de transferência e latência média obtida.

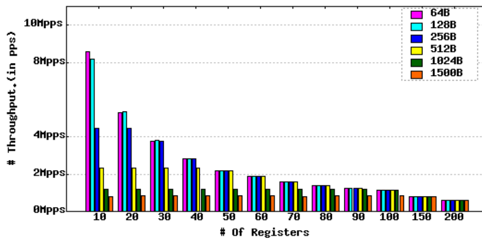
Tráfego intenso de pacotes: As figuras 5(a) e 5(b) representam a taxa obtida nos quesitos transferência e latência em um fluxo de pacotes de 64 Bytes (pacotes pequenos)



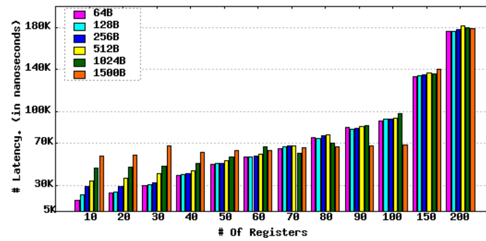
(a) Vazão média variando o número de registradores (pacotes de 64B)



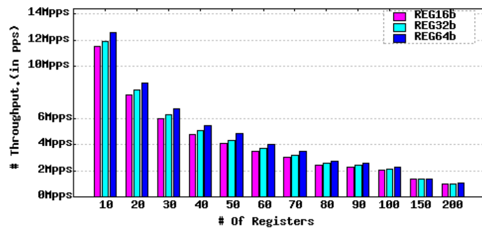
(b) Latência média variando o número de registradores (pacotes de 64B)



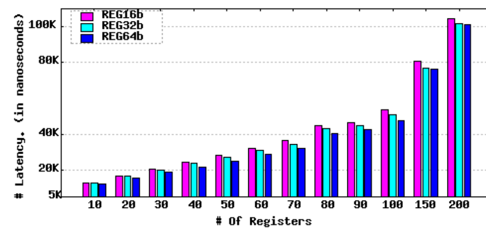
(c) Vazão média para diferentes tamanhos de pacotes



(d) Latência média para diferentes tamanhos de pacotes



(e) Vazão média para diferentes tamanhos de registradores



(f) Latência média para diferentes tamanhos de registradores

Figura 5 – Vazão e latência obtidas durante operações de leitura e escrita.

e registradores de largura de 32 bits. Observamos que conforme exista aumento no número de registradores, também aumenta o quantitativo de instruções no pipeline DPDK, e a degradação na transferência e latência é constante nas ações de leitura e leitura/escrita. Em ações de escrita, observamos degradação quase que insignificante em termos de transferência, executando até 50 leituras e na latência até 60 leituras, isso tudo para um fluxo de pacotes de 10 Gbit/s (14.18 Mpps).

Tráfegos com pacotes de tamanho variado: Com a alternância no tamanho de pacotes, registradores com ação de leitura e escrita e largura fixa de 32 bits, observa-se nas figuras 5(c) e 5(d), que pacotes de maior tamanho, maior é o quantitativo de operações de registro suportada. Vimos que com pacotes com tamanho de 512B, não houve degradação com até 40 registradores lidos e escritos, já com pacotes com seu tamanho de 1024B é sustentado sem degradação até 100 registradores, e com pacotes de 1500B de tamanho, houve degradação do tráfego de pacotes a partir de 150 registradores. Já no quesito latência, a figura 5(d), nos mostra que quanto maior for o pacote, maior é sua latência de tráfego,

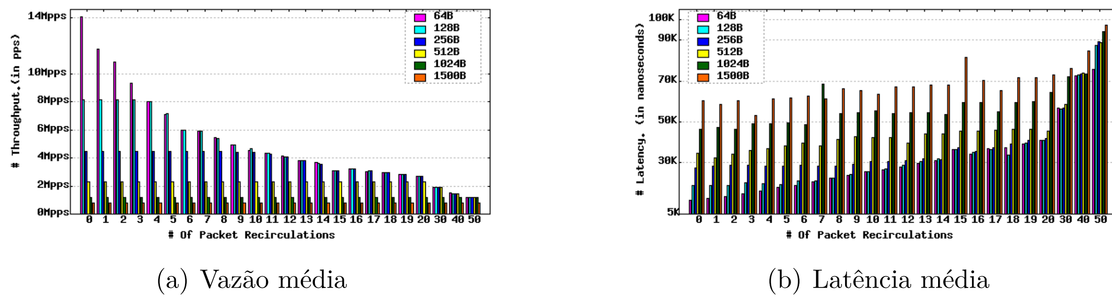
por exemplo, pacotes com tamanho de 1500B, o aumento da latência foi de até 3x (de 57732ns a 179051ns), e para pacotes pequenos (64B), o aumento de latência foi de até 12x.

Tráfego intenso de pacotes com registradores de tamanho variado: Neste cenário avaliamos o desempenho da aplicação, em um tráfego de pacotes com tamanho de 64B e registradores com ação de leitura variando o seu tamanho (16, 32 e 64 bits). Devido a limitações da ferramenta de compilação e conversão, não foi possível utilizar registradores de tamanho maior que 64 bits.Observamos nas figuras 5(e) e 5(f), que houve degradação na taxa de transferência de pacotes em todos os quantitativos de registradores empregados no teste, onde para uma taxa de tráfego de 14.18Mpps injetada pelo gerador Moogen, as menores degradações foram com execuções de 10 leituras de registrador, onde um registrador com tamanho de 16 bits teve degradação de 19%, o registrador com tamanho de 32 bits degradou 16% do tráfego gerado e com registrador de 64 bits a degradação foi de 11%.

3.2.1.2 DESEMPENHO DURANTE RECIRCULAÇÃO DE PACOTES

Nessa avaliação fizemos uso do arquivo de exemplo de recirculação de pacotes disponibilizado na API DPDK. Recirculação de pacotes integra a lista de recursos não suportados pelo backend DPDK do compilador P4C. A linguagem P4 não suporta estruturas com base em interações (loops), a recirculação de pacotes é utilizada para contornar essa limitação em programas P4. Em resumo, a ação de encaminhar um pacote de volta ao pipeline de entrada após seu processamento, o que se assemelha a uma estrutura com base em loop, descreve o funcionamento da recirculação de pacotes. Para nosso cenário alteramos o arquivo de exemplo de recirculação de pacotes, para que atendesse nossa necessidade, de variar de 0 a 50 o número de recirculações em cada pacote, de mesmo modo, variamos o tamanho dos pacotes (de 64B a 1500B) do tráfego gerado. Ao analisar as figuras 6(a) e 6(b), onde é ilustrado o comportamento da taxa de tráfego dos pacotes, observou-se que quanto maior o número de recirculações de pacotes, menor é a taxa linear mantida no tráfego de pacotes pequenos, por exemplo, pacotes de 128B a taxa linear é mantida até 4 recirculações. No oposto, pacotes maiores como, por exemplo, de tamanho 512B a taxa em linha é mantida com até 20 recirculações de pacotes, com pacotes de 1024B a taxa em linha se mantém com até 50 recirculações.

O fato a ser considerado para a redução da taxa de transferência em recirculações de pacotes é que, conforme os pacotes são recirculados, outros pacotes estão sendo enviados para o plano de dados que os enfileira na espera de seu processamento no pipeline. Na avaliação de latência, observamos considerável aumento conforme pacotes são recirculados. Recirculações de pacotes pequenos (64B) de 0 até 50 recirculações obtiveram aumento de latência de até 6.5x (11627ns até 75601ns). Com fluxo de pacotes de maior tamanho como de 1500B, observou-se aumento de latência de até 1.6x (60582ns até 97447ns).

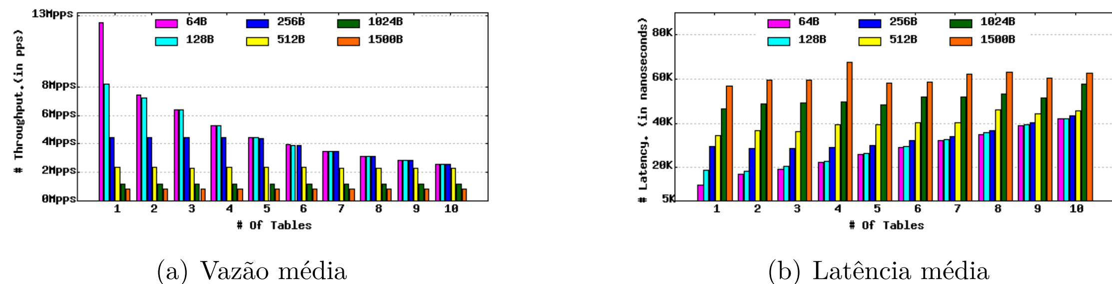


(a) Vazão média (b) Latência média

Figura 6 – Vazão e latência obtidas durante a recirculação de pacotes no pipeline.

3.2.1.3 DESEMPENHO DE APLICAÇÕES COM MÚLTIPLAS TABELAS

Nesse cenário, conduzimos uma análise do impacto no uso de múltiplas tabelas de correspondência e ação em pipeline DPDK. Nosso objetivo principal foi investigar como o desempenho é afetado pelo número de tabelas em programas P4. Durante o experimento, manipulamos o número de tabelas utilizadas, variando de 1 a 10, e garantimos que cada pacote fosse correspondido sequencialmente em todas as tabelas. Em cada correspondência, uma ação é acionada para ler um único dado de 32 bits da tabela e armazená-lo em uma estrutura de metadados do pacote. Além disso, também exploramos a variação do tamanho dos pacotes, indo de 64B a 1500B, e o número de tabelas no pipeline de entrada.Na figura 7(a), observamos uma degradação constante na taxa de transferência para fluxos de pacotes pequenos à medida que o número de tabelas aumenta.



(a) Vazão média (b) Latência média

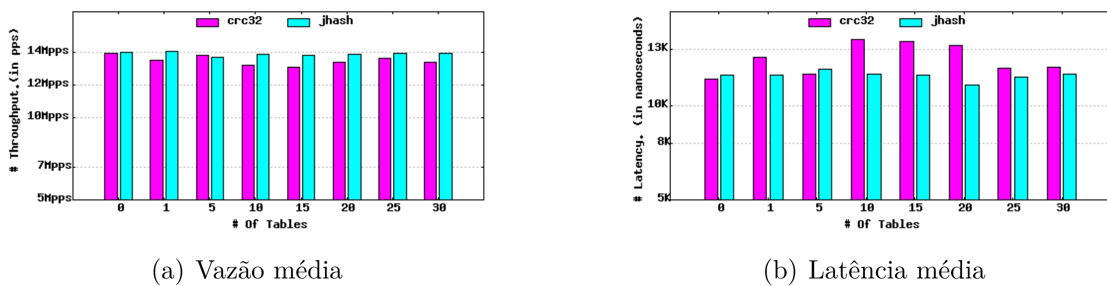
Figura 7 – Desempenho mediante aplicações com múltiplas tabelas.

Para fluxos com pacotes de 64B, a taxa de transferência diminuiu em 15%, enquanto para fluxos com pacotes de 128B, a degradação foi de 42%, considerando apenas uma tabela de correspondência e ação. Com pacotes a partir de 256B, observamos um comportamento linear na taxa de transferência até 5 tabelas, independentemente do tamanho do pacote. No entanto, a taxa linear é mantida para os demais tamanhos de pacotes utilizados, mesmo com 10 tabelas no fluxo. Na figura 7(b), podemos observar o aumento progressivo na latência à medida que o número de tabelas aumenta. O maior acréscimo de latência foi de 71% para pacotes de 64B no crescente número de tabelas. Por

outro lado, o menor acréscimo de latência, de 10%, foi observado entre 1 e 10 tabelas para pacotes de 1500B.

3.2.1.4 DESEMPENHO NO USO DE FUNÇÕES CRIPTOGRÁFICAS

Continuamos nossa avaliação de desempenho, agora com aplicações que executam funções hash em um pipeline DPDK. Para este experimento, utilizamos um arquivo modelo da API DPDK, adaptado-o para nossas necessidades, devido à falta de suporte para funções hash no compilador P4C, com o uso do backend DPDK.



(a) Vazão média (b) Latência média

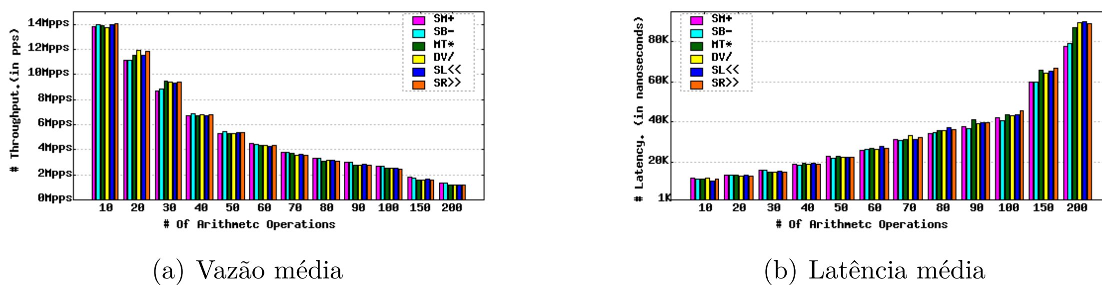
Figura 8 – Impacto com uso de funções hash.

Embora o DPDK suporte outros algoritmos de funções hash, a aplicação de exemplo usada no experimento enfatiza seu pleno funcionamento com os dois algoritmos usados neste teste. Na Figura 8(a), observamos que, ao aplicar o algoritmo crc32, há uma variação irregular na taxa de transferência à medida que o número de ações hash aumenta. Por exemplo, em um fluxo de pacotes de tamanho 64B e sem ações hash, a taxa de transferência de 14,18 Mpps injetada pelo Moogen sofre uma degradação de apenas 1,7% ao ser processada pela aplicação. Essa degradação aumenta quando uma ação hash é aplicada (4,7%) e diminui quando são aplicadas cinco ações hash (2,5%), em comparação com a aplicação de uma única ação hash. Ao aplicar o algoritmo jhash, observa-se um padrão semelhante ao descrito acima, porém de forma mais suave.

Em um fluxo de 14,18 Mpps e sem a aplicação de funções hash jhash, a taxa de transferência sofre uma degradação de apenas 1%, enquanto com uma ação hash a degradação é de 0,8%, e com a aplicação de cinco ações hash, a taxa de transferência degradada é de 3%. Observa-se que, a partir de cinco ações hash com o algoritmo jhash, foi obtida uma taxa de transferência linear, mesmo com o aumento do número de ações hash executadas. Na Figura 8(b), são ilustrados os valores de latência. Observa-se que os maiores valores de latência ocorrem nas ações hash com o uso do algoritmo crc32. Comparando o menor valor de latência obtido com o maior valor, há um acréscimo de 15,4%. Já nas ações hash com o algoritmo jhash, esse acréscimo é de 7%.

3.2.1.5 DESEMPENHO NO USO DE OPERAÇÕES ARITMÉTICAS

Neste cenário, avaliamos nossa aplicação P4 em um pipeline DPDK executando operações aritméticas consecutivas. O compilador P4C, juntamente com o backend-DPDK, otimiza o código a ser compilado, removendo qualquer instrução que não interaja diretamente com o fluxo de dados. Considerando que nosso programa P4 apenas encaminha o tráfego de rede entre as interfaces físicas, para avaliar o impacto das operações aritméticas no desempenho, instruímos que essas operações utilizassem os valores dos campos TTL do cabeçalho IPv4 e ETHER_TYPE do cabeçalho Ethernet. Os resultados obtidos são apresentados na figura 9.



(a) Vazão média

(b) Latência média

Figura 9 – Desempenho no uso de operações aritméticas.

As figuras 9(a) e 9(b) ilustram o comportamento da taxa de tráfego à medida que o número de operações aritméticas aumenta. Observamos uma degradação maior na taxa de transferência ao aumentar de 10 para 20 operações de soma e subtração (média de 20%), em comparação com as demais operações aritméticas no mesmo número de operações (média de 16%). O mesmo padrão se repete ao aumentar de 20 para 30 operações executadas, com uma média de degradação de 21% para operações de soma e subtração, em comparação com as demais operações aritméticas (média de 19%). Ao aumentar de 30 para 40 operações aritméticas, é observada uma degradação semelhante entre os operadores aritméticos em relação à taxa de transferência afetada.

A média de degradação exercida pelas operações foi de 26% na taxa de transferência. Esse padrão se repete para os demais números de operações aritméticas executadas no experimento, ou seja, há uma oscilação insignificante nos valores da taxa de transferência entre os operadores aritméticos e uma queda constante na transferência de pacotes à medida que o número de operações executadas pelo programa P4 aumenta.

Na Figura 9(b), é observado um aumento constante na latência conforme o número de execuções aritméticas aumenta. No entanto, a relação entre os operadores aritméticos não é relevante em termos de latência exercida por cada operação. Com a execução de 10 a 100 operações aritméticas, houve um aumento de 376% na latência, resultando em uma média de aumento de 37% a cada acréscimo de 10 execuções aritméticas no programa P4.

3.2.2 DESEMPENHO COM ESPECIFICAÇÕES ST(II), ST(III) E ST(IV)

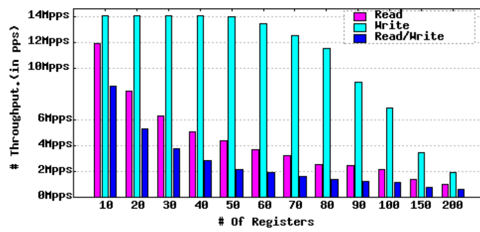
Nesta seção, conduzimos uma análise abrangente nos resultados gerados por cada cenário das diferentes STs realizadas. Com os dados obtidos em cada cenário das STs, realizamos comparações entre resultados em cenários semelhantes de ST(i), onde seus resultados de desempenho usaremos com referência para os resultados das demais sessões.

3.2.2.1 DESEMPENHO DURANTE OPERAÇÕES DE LEITURA E ESCRITA

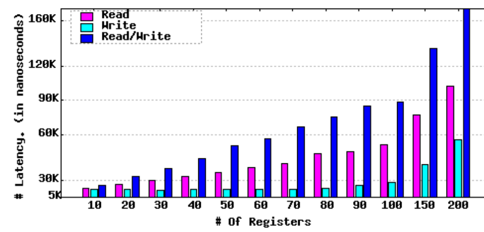
Desempenho com tráfego intenso de pacotes: A figura 10(a) e 10(b) ilustra o desempenho do programa P4 em execução no pipeline DPDK, submetido a um tráfego intenso de pacotes de tamanho de 64 Bytes, e largura de 32 bits dos registradores, e utilizando da configuração ST(i) de 2 lcores de alocação. Na atual seção esses dados se darão como referência para os resultados gerados no uso das demais configurações utilizadas. Iniciamos nossa análise examinando os resultados alcançados por esse cenário em ST(ii), os representados nas figuras 10(c) e 10(d) e englobam a taxa de transferência e a latência. Ao comparar os resultados da figura 10(c) com os da figura 10(a), observamos uma redução na taxa de transferência quando utilizamos a configuração ST(ii). Na figura 10(a), é possível notar uma taxa linear de transferência em ações de escrita de registros até 50 execuções (13,98 Mpps), enquanto na figura 10(c), essa taxa linear é mantida até 40 execuções (13,14 Mpps em 50 execuções).

Em relação à latência, a Figura 10(b) ilustra um aumento de até 9 vezes (12.972 ns a 101.887 ns) em ações de leitura de registrador entre 10 e 200 execuções. Já para ações de escrita, o aumento é de até 5 vezes (11.966 ns a 55.160 ns) para 10 a 200 execuções. Quando consideramos ações de leitura e escrita, o aumento da latência chega a 11 vezes (15.636 ns a 175.161 ns) em 10 a 200 ações. Os dados coletados durante os testes com a configuração ST(ii) são apresentados na figura 10(d), que ilustra os valores correspondentes à latência. Nesse caso, para ações de leitura entre 10 e 200 execuções, observamos um aumento de até 8 vezes (12.885 ns a 110.643 ns). No caso das ações de escrita, a latência aumentou até 4 vezes (11.702 ns a 51.356 ns) para 10 a 200 execuções. Já para ações de leitura e escrita, o aumento da latência alcançou até 12 vezes (16.234 ns a 193.854 ns) em 10 a 200 execuções. Nas figuras 10(e) e 10(f), é ilustrado o resultado gerado pelo teste utilizando configuração ST(iii), ou seja, 8 lcores.

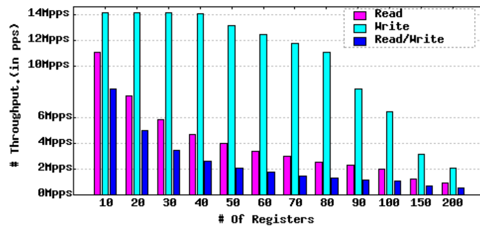
Em comparação com os resultados apresentados na Figura 10(a), a Figura 10(e) demostra em sua taxa linear de transferência em ações de escritas em registros, degradação em 40 execuções. Podemos citar que nos testes apresentados na Figura 10(a), essa degradação ocorre com 60 execuções. Em termos de latência, a ilustração é feita pela figura 10(f) onde observamos aumento onde, ações de leitura de registradores para 10 até 200 execuções, o crescimento foi de até 9x (12941ns a 115272ns). Com execuções de escrita em registradores, até 5x (11141ns a 53363ns) foi a alta da latência em 10 até 200 ações.



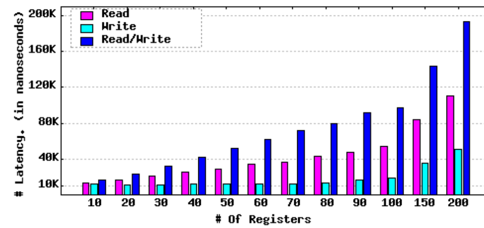
(a) Vazão com especificação ST(i)



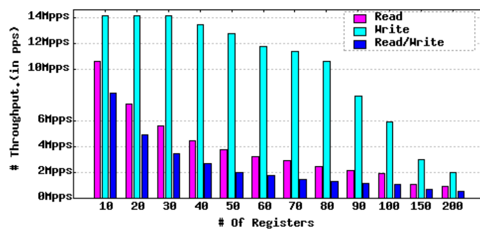
(b) Latência com especificação ST(i)



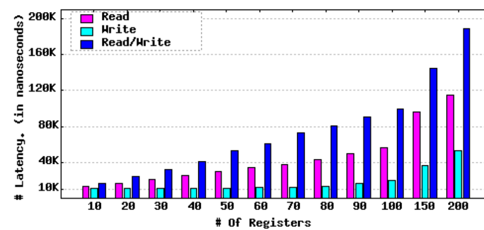
(c) Vazão com especificação ST(ii)



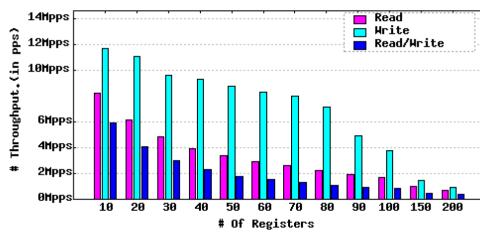
(d) Latência com especificação ST(ii)



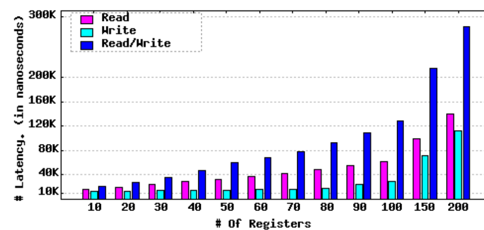
(e) Vazão com especificação ST(iii)



(f) Latência com especificação ST(iii)



(g) Vazão com especificação ST(iv)



(h) Latência com especificação ST(iv)

Figura 10 – Vazão e latência obtidas durante operações de leitura e escrita variando o número de registradores (pacotes de 64B).

Para ações de leitura e escrita em registradores, o aumento ficou em até 12x (16119ns a 189299ns) entre as 10 até 200 execuções.

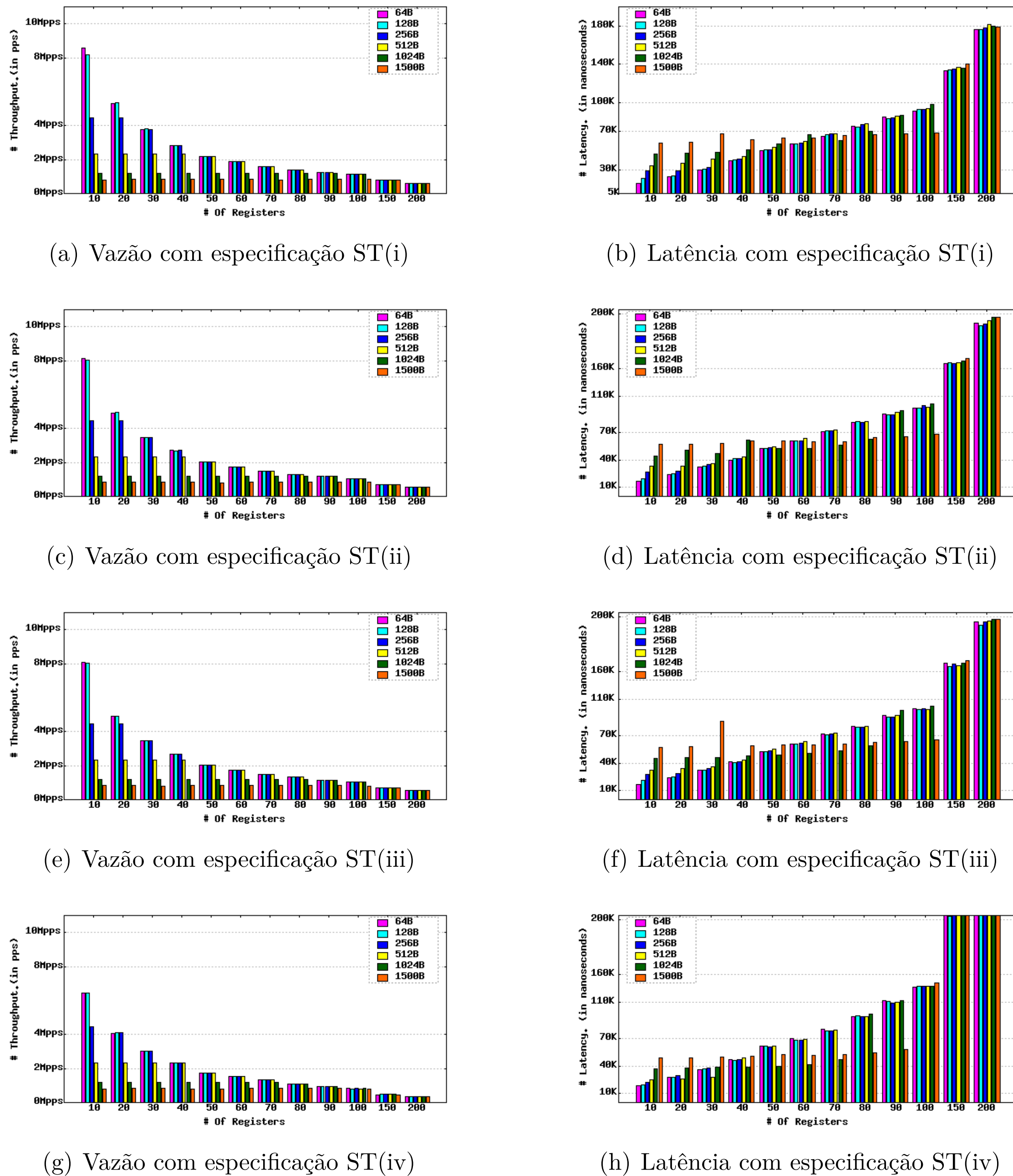
Como demostrado nas Figuras 10(g) e 10(h), os valores de taxa de transferência e latência para o teste executado com a configuração ST(iv) de 16 lcores, não obtém uma taxa linear em transferência, e nota-se analisando a Figura 10(g), considerável queda do valor de transferência em comparação com 10(a). Com o fluxo de pacotes pequenos (64 Bytes) em 10 Gbit/s(14.18 Mpps), em ações de leitura em registradores no pipeline, o valor de transferência alcançado foi de 8.19 Mpps em 10 execuções. Com escrita em registradores para 10 execuções a transferência alcançada é de 11.66 Mpps, e leitura e escrita em

registradores, 5.89 Mpps é o valor da transferência em 10 execuções. Para latência os acréscimos foram de até 9x (15667ns a 139730ns) em ação de leitura de registradores, com ações de escrita em registradores, até 9x (12359ns a 112036ns) de aumento na latência, e para leitura e escrita uma alta de até 14x (20396ns a 284128ns) entre 10 até 200 execuções.

Desempenho com pacotes de tamanhos diferentes: A Figura 11(c) e 11(d) apresentam o desempenho da aplicação PDP ao realizar ações de leitura e escrita em registradores com 32 bits de espaço. Nesse contexto, a aplicação em questão processa pacotes de um fluxo de tráfego injetado em um pipeline. Para cada ocasião, variamos o tamanho dos pacotes (64B, 128B, 256B, 512B, 1024B e 1500B) do fluxo de tráfego aplicado. Iniciamos nossa análise comparando os resultados obtidos, como mostrado na Figura 11(c), com os resultados de referência apresentados em 11(a). Observamos que o cenário em ST(ii) apresentou um desempenho inferior em relação à taxa de transferência para todos os tamanhos de pacotes utilizados nos fluxos de dados injetados no pipeline, ao comparar os resultados do mesmo cenário em ST(i). Apesar dessa degradação na taxa de transferência, foi constatado que ambos os cenários mantiveram períodos retilíneos de taxa de transferência de forma semelhante. Os períodos lineares na taxa de transferência ocorreram durante o processamento de pacotes com tamanho de 256B ao realizar 10 e 20 operações no pipeline, entre 10 e 40 execuções para pacotes de tamanho 512B, e durante 10 a 100 leituras e escritas em registradores com pacotes de tamanho 1024B e 1500B.

Para os dados de latência, constatamos que, à medida que o número de execuções aumenta, a latência também aumenta. Outro ponto relevante que notamos é que a latência tende a ser maior para pacotes com tamanho maior. Isso ocorre porque pacotes maiores exigem mais recursos e tempo para serem transmitidos e processados pelo sistema. Portanto, é natural que a latência seja mais alta para pacotes com maiores dimensões. Curiosamente, ao analisarmos a latência específica para pacotes de tamanho 1500 bytes, observamos um comportamento interessante. Entre 10 e 100 ações, a latência manteve-se constante, ou seja, não houve aumento significativo. Isso sugere que a taxa de latência para pacotes de 1500B permaneceu estável dentro dessa faixa de quantidade de ações. Os dados do cenário em ST(iii) representados na Figura 11(e), constatamos uma semelhança com os resultados do cenário em ST(ii), particularmente em relação à taxa de transferência.

A variação dos resultados entre os cenários ST(ii) e ST(iii) revelou-se insignificante, a ponto de considerarmos que o cenário obteve resultados idênticos em termos de taxa de transferência. Com base nisso, conduzimos uma investigação comparativa dos resultados do cenário ST(iii), ilustrado na Figura 11(e), com os dados do cenário semelhante ST(i) apresentados na Figura 11(a). Verificamos novamente um desempenho inferior em termos de taxa de transferência para todos os tamanhos de pacotes utilizados nos fluxos de dados injetados no pipeline. Além disso, identificamos períodos de taxa de transferência com comportamento semelhante: para o processamento de pacotes com tamanho de 256B, entre 10 e 20 operações no pipeline; para pacotes de tamanho 512B, entre 10 e 40 execuções; e



(a) Vazão com especificação ST(i)

(b) Latência com especificação ST(i)

(c) Vazão com especificação ST(ii)

(d) Latência com especificação ST(ii)

(e) Vazão com especificação ST(iii)

(f) Latência com especificação ST(iii)

(g) Vazão com especificação ST(iv)

(h) Latência com especificação ST(iv)

Figura 11 – Vazão e latência obtidas durante operações de leitura e escrita para diferentes tamanhos de pacotes.

para leituras e escritas em registradores com pacotes de tamanho 1024B e 1500B, entre 10 e 100 execuções. No entanto, em relação à latência, notamos uma semelhança entre os resultados ilustrados nas Figuras 11(d) e 11(f), com uma observação especial para a taxa linear de processamento de pacotes de tamanho 1500B.

No cenário ST(iii), o comportamento da latência se mantém linear entre 40 e 100 execuções, enquanto no cenário ST(ii), a linearidade na latência ocorre de 10 a 100 execuções. Com essa avaliação, concluímos que, à medida que o número de execuções aumenta, a latência também cresce. Além disso, a latência tende a ser maior para pacotes

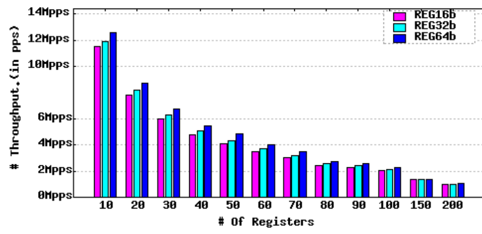
com tamanho maior. Vale ressaltar que no processamento de pacotes de tamanho 1500B, observamos uma latência linear entre 40 e 100 execuções de ação, isso no equiparo dos resultantes dos cenários em ST(iii) e ST(i). Para esse cenário, em ST(iv), os valores registrados em latência e taxa de transferência, são ilustrados na Figura 11(g) e 11(h).

Há uma degradação na vazão, comparando a Figura 11(g) com os resultados do cenário em ST(iv), vistos pela Figura 11(a). Para o processamento de pacotes pequenos (64B) no pipeline do cenário em ST(iv), o valor alcançado em transferência é de 6.46 Mpps ao executar 10 ações por pacote. O cenário ST(i) obtêm uma vazão de 8.56 Mpps (25% mais pacotes processados) para processamento de pacotes de tamanho 64B, executando 10 ações por pacote no pipeline. Constatamos transferência com taxas continuas ao observar o processamento de pacotes de maior tamanho. Pacotes de tamanho 512B mantiveram uma taxa linear em transferência entre 10 e 40 execuções, pacotes de 1024B de tamanho, a transferência tem comportamento semelhante quando está executando entre 10 até 70 ações. Porém, pacotes de 1500B a taxa linear em transferência se mantém no realizar ações de 10 a 100 vezes por pacote. Em termos de latência, é observado um maior custo conforme o número de ações aumenta com pacotes de tamanho menor que 1500B. Para o quantitativo de 10 execuções até 90, pacotes com 1500B de tamanho, tem seu processamento no pipeline com uma latência linear.

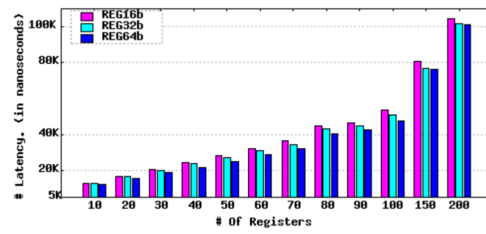
Desempenho com registradores de tamanhos diferentes: Nossa próxima avaliação consiste no cenário onde o pipeline executa uma aplicação PDP efetuando leitura em registradores variando seu tamanho (16b, 32b e 64b). A aplicação no pipeline processa pacotes pequenos (64B) de um intenso trafego injetado pelo MooGen. Os dados de ST(i) são utilizados como referência em nossa avaliação. Os resultados apresentados na Figura 12(a) e 12(b) pode ser visto degradação na taxa de transferência para 10 ações executadas no pipeline de 19% com registrador de tamanho 16b, 16% com registrador de 32b e 11% para registrador de 64b de espaço. A degradação na taxa de transferência é maior conforme aumenta o número de execuções da ação no pipeline.

Em termos de latência, se observa um maior custo ao ampliar o número de execuções da ação no pipeline. Para o cenário na sessão de testes ST(ii), observamos uma degradação em transferência, em comparação com os resultados apresentados na Figura 12(c) (que ilustra o resultado alcançado pelo cenário em ST(ii)) e os resultados de referência na Figura 12(a). O resultado em taxa de transferência do cenário em ST(ii) são de: em registrador de tamanho 16b, 10.54 Mpps (8% menor), registrador de 32b de tamanho, 11.07 Mpps (7% menor), e com 64b de espaço 11.76 Mpps (7% menor).

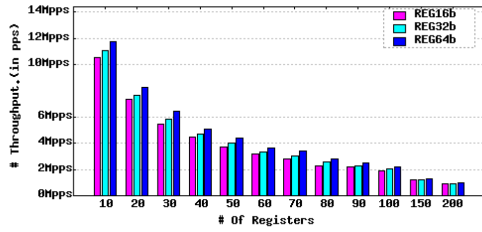
Em termos de latência, não é observada alterações significativas nos resultados, equiparado com os valores de latência de referência. O cenário na sessão de testes ST(iii), os resultados registrados em taxa de transferência e latência são expostos na Figura 12(e) e 12(f). Confrontando os resultados da Figura 12(e), com os dados da Figura 12(a) de referência, notamos um maior custo do cenário em ST(iii) na taxa de transferência.



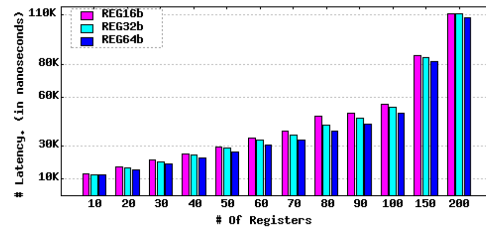
(a) Vazão com especificação ST(i)



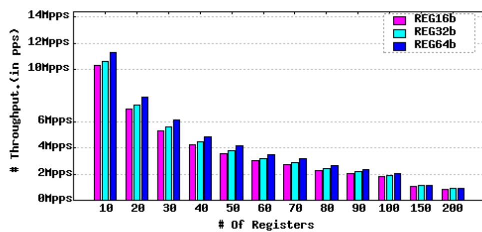
(b) Latência com especificação ST(i)



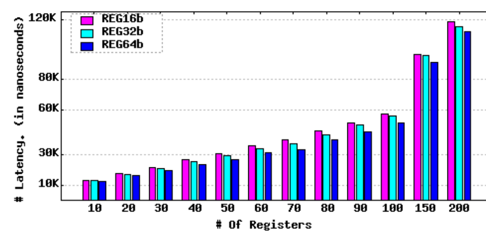
(c) Vazão com especificação ST(ii)



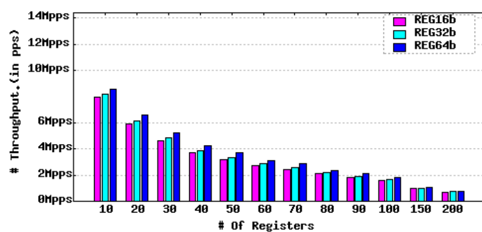
(d) Latência com especificação ST(ii)



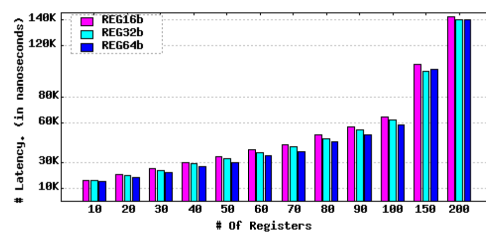
(e) Vazão com especificação ST(iii)



(f) Latência com especificação ST(iii)



(g) Vazão com especificação ST(iv)



(h) Latência com especificação ST(iv)

Figura 12 – Vazão e latência obtidas durante operações de leitura e escrita para diferentes tamanhos de registradores.

Para o caso de uso do cenário com registrador de tamanho 16b obtivemos 10.27 Mpps em taxa de transferência, sendo 10% menor do registrado pelo cenário em ST(i) (11.49 Mpps) para o mesmo caso. O valor em transferência do cenário para o caso de registrador de largura 32b alcançou 10.63 Mpps, contra 11.88 Mpps (10% maior) do valor de referência. A latência para os três caso do cenário em ST(iii) apresenta resultados pouco maiores dos obtidos em ST(i). Por meio das Figuras 12(b) e 12(f), observamos um comum acréscimo da latência entre os casos de uso do cenário. O custo de latência teve aumento entre 2% a 10%, do que o custo referencia.

As Figuras 12(g) e 12(h), ilustram os valores alcançados pelo cenário na sessão

de testes ST(iv). Os resultados em taxa de transmissão nos três casos de uso do cenário, podemos ver valores alcançados abaixo de 10.00 Mpps, para o menor número de ações executadas no pipeline. Em latência, observamos um maior custo para os casos de uso do cenário, demostrados pela Figura 12(h), em relação dos resultados na Figura 12(b).

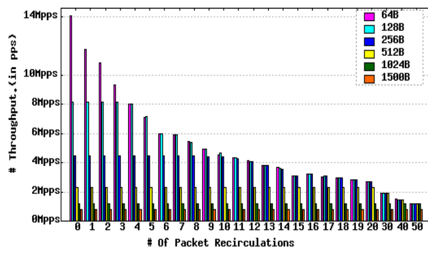
3.2.2.2 DESEMPENHO DURANTE RECIRCULAÇÃO DE PACOTES

Neste experimento, analisamos os dados obtidos em diferentes cenários da aplicação PDP, que envolve a recirculação de pacotes no pipeline. A recirculação de pacotes é uma técnica usada na programação do plano de dados em redes SDN, onde é necessário repetir o processamento de entrada em um pacote após a conclusão do processamento de saída, semelhante a um loop em programação. Exploramos diferentes casos de uso nesse cenário, processando pacotes de rede com tamanhos variados (64B, 128B, 256B, 512B, 1024B e 1500B). Para cada caso, avaliamos o número de recirculações, variando de 0 a 50, e registramos os resultados de taxa de transferência e latência.

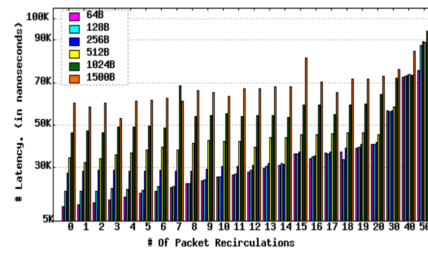
Utilizaremos os resultados de ST(i) como referência para avaliar os dados obtidos por cada cenário nas demais sessões de testes (STs). Apreciamos os resultados alcançados para o caso de uso do cenário em recircular pacotes, na sessão de testes ST(ii). Os resultados em taxa de transferência e latência, visualizamos no esboço da Figura 13(c) e 13(d). Nossa observação inicia em taxa de transferência, comparando os dados da Figura 13(c) com os dados ilustrados em 13(a).

Não encontramos alterações significantes entre os resultados obtidos pelo cenário para as sessões de testes ST(i) e ST(ii), portanto consideramos o igual alcance em taxa de transferência, bem como, dos comportamentos semelhantes entre os casos de uso do cenário, para tamanhos diferentes dos pacotes injetados no pipeline. Para latência, é observada uma variação entre os resultados apresentados na Figura 13(d) em comparação à Figura 13(b). Os valores de latência utilizados de referência apresentam um maior custo ao recircular pacotes de tamanho 64B, 128B e 512B, 1x no pipeline. No quantitativo de cinco recirculações do pacote de tamanho 64B, o valor obtido pelo cenário em ST(i) é menor (17915 ns) que o registrado (18919 ns) em ST(ii), e com pacotes 128B e 512B de tamanho, consideramos iguais os valores obtidos por apresentar variação menor que 150 ns.

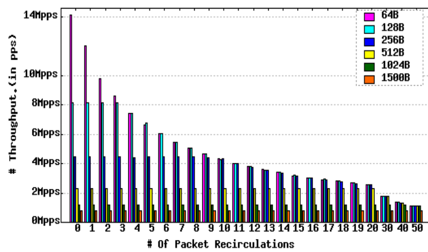
Um ponto a ser observado em latência é que não se obteve um padrão na variação dos resultados, perante as alterações no número de recirculações imposta aos pacotes no pipeline. Na avaliação em dados resultantes dos casos de uso do cenário em ST(iii). Atentamos para a similaridade dos resultados apresentados pela Figura 13(e) em comparação com a Figura 13(a) no aspecto de taxa de transmissão. Ao verificarmos os resultantes em latência do cenário em ST(iii), e ao confrontarmos com os resultados apresentados na Figura 13(f) e 13(b), podemos ver um equilíbrio entre os valores apresentados, porém com uma visível oscilação em termos de latência que ocorre nos casos onde os pacotes do fluxo



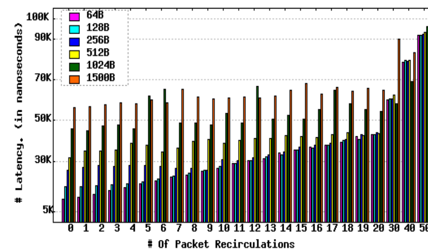
(a) Taxa média de transferencia em configuração ST(i)



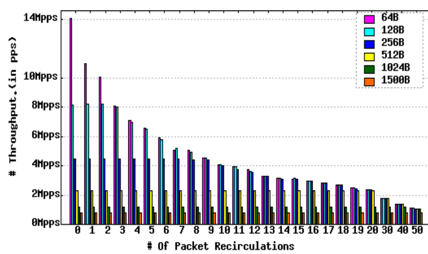
(b) Latência média em configuração ST(i)



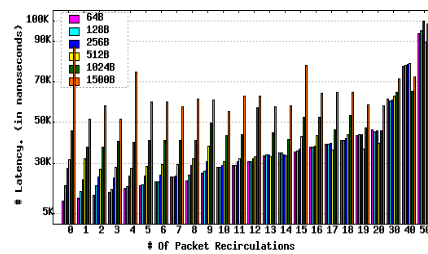
(c) Taxa média de transferencia em configuração ST(ii)



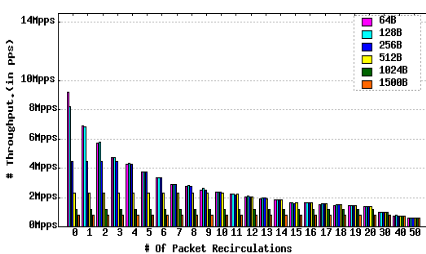
(d) Latência média em configuração ST(ii)



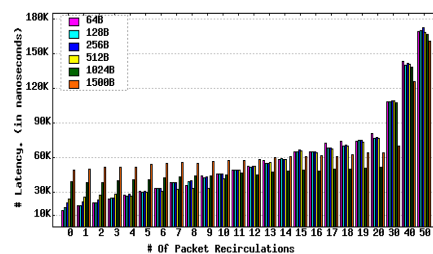
(e) Taxa média de transferencia em configuração ST(iii)



(f) Latência média em configuração ST(iii)



(g) Taxa média de transferencia em configuração ST(iv)



(h) Latência média em configuração ST(iv)

Figura 13 – Impacto na recirculação de pacotes em pipeline DPDK.

injetado no pipeline são de tamanhos maiores de 512B.

Para a sessão de testes ST(iv), o cenário apresenta os resultados na Figura 13(g) em taxa de transferência e para latência na Figura 13(h). Podemos notar que a taxa de transferência não ultrapassa 10.00 Mpps, em nenhum dos casos de uso do cenário, além da menor permanência das taxas lineares de transferência. No quesito latência, observamos

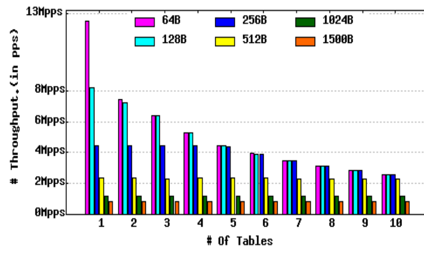
uma gradual linha crescente de latência nos casos de uso com pacotes pequenos (64B), e acompanha essa crescente linha em latência para os casos com pacotes de tamanho 128B, 256B e 512B. Os pacotes maiores, ou seja, de tamanho 1024B e 1500B, uma linha crescente, perpetua com variações mínimas (>= à 1300 ns) de latência para até 20 recirculações de um pacote.

3.2.2.3 DESEMPENHO DE APLICAÇÕES COM MÚLTIPLAS TABELAS

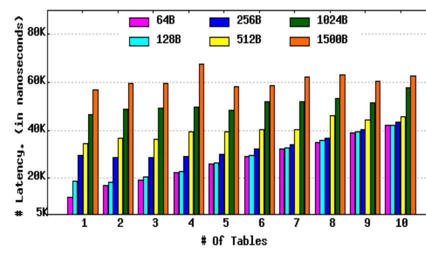
Nessa avaliação, conduzimos uma análise do impacto no uso de múltiplas tabelas de correspondência e ação por aplicações PDP em pipeline DPDK. Buscamos investigar como o desempenho é afetado pelo número de tabelas de uma aplicação PDP, para as diferentes sessões de testes (STs) realizadas. No decorrer dos experimentos em cada ST, manipulamos o número de tabelas utilizadas, variando de 1 a 10, e garantimos que cada pacote fosse correspondido sequencialmente em todas as tabelas. Para cada correspondência atendida, uma ação é acionada para ler um único dado de 32 bits da tabela e armazená-lo em uma estrutura de metadados do pacote. Além disso, também exploramos a variação do tamanho dos pacotes, indo de 64B a 1500B, e o número de tabelas no pipeline de entrada. Os resultados de ST(i) são utilizados como referência em nossas avaliações dos resultados das demais STs. A figura 14(a) e 14(b) apresentam os dados de referência em taxa de transferência e latência.

Os dados obtidos com os casos de uso do cenário em ST(ii), são expostos pela Figura 14(c) para taxa de transmissão alcançada e 14(d) para o valor de latência. Analisamos os dados de transferência e comparamos com os resultados da Figura 14(a). Observamos uma degradação na taxa de transferência, para o caso de uso do cenário processando pacotes de rede de tamanho pequeno (64B), e 1 tabela de correspondência e ação. O valor degradado pelo cenário na sessão de testes ST(ii) é de 11% (11.10 Mpps) menor, do valor (12.53 Mpps) obtido em ST(i) com o mesmo caso de uso do cenário. Observamos taxas semelhantes de transmissão em casos de uso com tráfego de pacotes de tamanho igual ou maior que 256B. No processamento de pacotes com 256B de tamanho, a taxa linear se mantém, para o uso de 1 até 4 tabelas de correspondência e ação da aplicação PDP, como mostra a Figura 14(c).

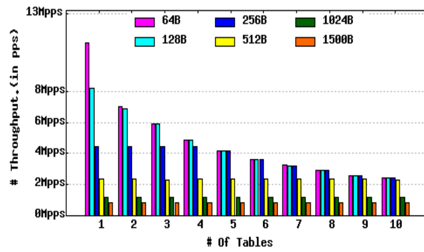
Nos casos de uso do cenário processando pacotes de tamanho 1024B e 1500B, os valores se mantêm alinhados com 1 até 10 tabelas de correspondência e ação do pipeline. Os dados de referência para os mesmos casos de uso, observamos na Figura 14(a), diferença apenas no alinhamento de resultados no caso de uso com pacotes de 256B de tamanho. A taxa de transferência se mantém com comportamento semelhante com 1 até 5 tabelas de correspondência e ação no pipeline. Os resultados em latência, comparamos os valores registrados pelo cenário em ST(ii), ilustrados na Figura 14(d), com os dados de referência da Figura 14(b). Em latência observamos similaridade dos resultados alcançados pelo cenário na sessão de testes ST(ii), com os dados de referência.



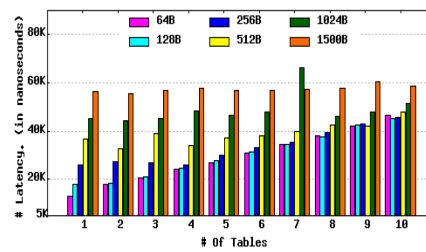
(a) Taxa média de transferencia em configuração ST(i)



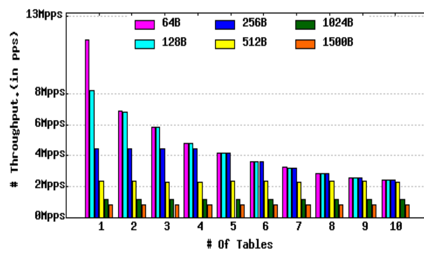
(b) Latência média em configuração ST(i)



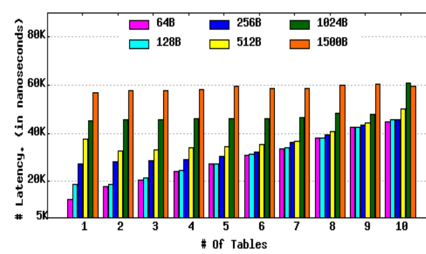
(c) Taxa média de transferencia em configuração ST(ii)



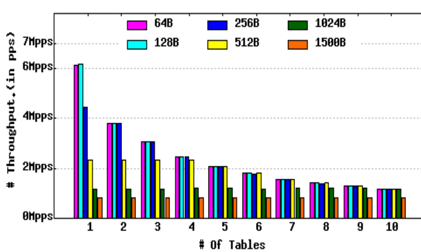
(d) Latência média em configuração ST(ii)



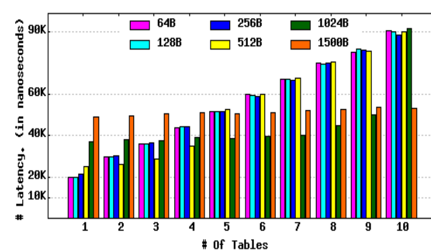
(e) Taxa média de transferencia em configuração ST(iii)



(f) Latência média em configuração ST(iii)



(g) Taxa média de transferencia em configuração ST(iv)



(h) Latência média em configuração ST(iv)

Figura 14 – Desempenho do pipeline DPDK no uso de múltiplas tabelas.

No conjunto geral dos resultados apresentados na Figura 14(d), os valores obtidos são melhores do que os dados de referência vistos na Figura 14(b). Nossa avaliação nos resultados obtidos pelo cenário na sessão de teste ST(iii) estão ilustrados na Figura 14(e) e 14(f). Os resultados alcançados pelo cenário em ST(iii) em taxa de transferência e latência apresentam uma variação mínima para os valores alcançados em ST(ii) por esse cenário.

Com essa similaridade dos valores alcançados pelo cenário, os pontos relevantes observados em nossa avaliação são os mesmos já descritos na avaliação dos resultados registrados pelo cenário em ST(ii). Para os casos de uso do cenário em ST(iv), avaliamos os valores dos dados atingidos em latência e taxa de transferência. Os valores registrados para ST(iv) por esse cenário, são apresentados na Figura 14(g) e 14(h).

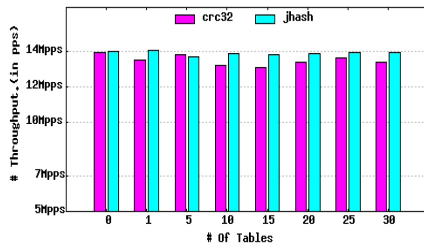
No comparar os dados apresentados na Figura 14(g) com os dados de referência em 14(a), observamos uma degradação acentuada na taxa de transmissão. O caso de uso do cenário no processamento de pacotes com 64B de tamanho apresentou uma queda maior que 50% em todas as variações aplicadas em número de tabelas no pipeline. Os resultados de latência dos valores atingidos com os casos de uso do cenário em ST(iv). A Figura 14(h) mostra um menor custo em latência no processar pacotes de rede de tamanho 1024B e 1500B, comparando com os dados referência ilustrados na Figura 14(b). Para os demais casos de uso observamos um maior custo do cenário em latência de até 78%.

3.2.2.4 DESEMPENHO NO USO DE FUNÇÕES CRIPTOGRÁFICAS

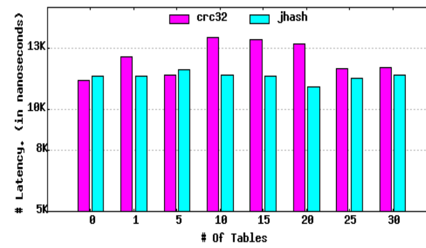
Nossa próxima avaliação de desempenho consiste em um cenário utilizando um programa PDP de uso de funções de criptografia. Funções criptográficas são amplamente utilizadas como, por exemplo, funções hash para buscar elementos em bases de dados, verificar a integridade de arquivos ou armazenar e transmitir senhas de usuários. Neste cenário usamos de instruções hash para calcular uma assinatura sobre um conjunto de n-tuplas de campos lidos do cabeçalho do pacote de rede. Variamos o número de hash de 0 a 30 execuções no pipeline para um processamento de pacotes pequenos (64B), na finalidade de avaliar o impacto em latência e taxa de transferência. Usaremos os dados obtidos nos casos de uso do cenário em ST(i), como referência para avaliarmos os demais resultados alcançados pelo cenário nas outras STs.

O cenário na sessão de testes ST(ii) registra os melhores resultados em latência e taxa de transferência comparado com os valores de referência. A Figura 15(c) apresenta uma perda inferior a 1% para todos os casos de uso do cenário em taxa de transferência. Para um tráfego de pacotes de 14.18 Mpps aplicado ao pipeline, no cenário em ST(i), observamos degradação de 5% em taxa de transmissão executando 30 hash's crc32. Com ações hash jhash, em 30 execuções não é visto perdas significantes de transferência.

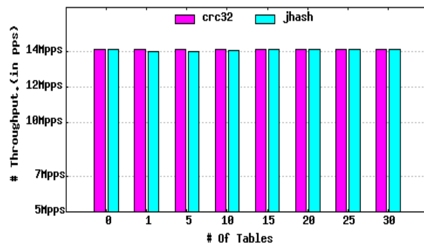
Os resultados em latência, a Figura 15(d) apresenta melhor desempenho na sessão de teste ST(ii), diante dos valores de referência da Figura 15(b). Na sessão de testes ST(iii), nas Figuras 15(e) e 15(f), podemos observar que os resultados alcançados demostram semelhança dos valores registrados em ST(ii). Essa semelhança obtida entre ST(iii) e ST(ii) pelo cenário é visto em todos os casos de uso, em taxa de transferência e latência. Durante a sessão de testes ST(iv), é visto que o cenário não acompanha o desempenho dos demais STs. As Figuras 15(g) e 15(h) apresentam uma considerável degradação em taxa de transferência, e significante aumento nos valores de latência em todos os possíveis



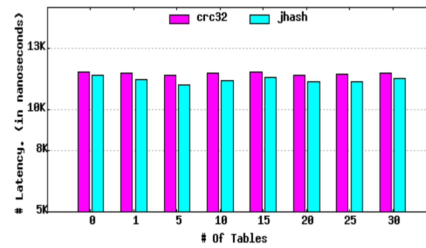
(a) Taxa média de transferencia em configuração ST(i)



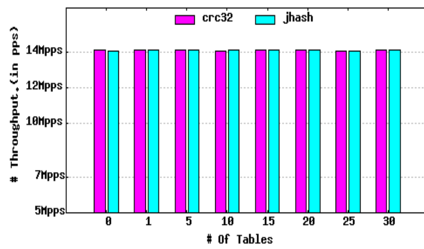
(b) Latência média em configuração ST(i)



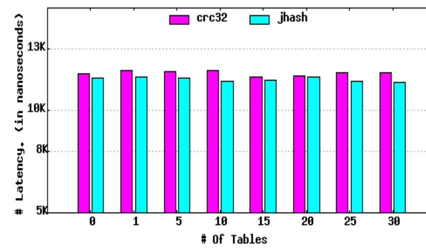
(c) Taxa média de transferencia em configuração ST(ii)



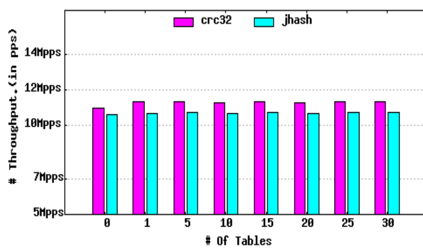
(d) Latência média em configuração ST(ii)



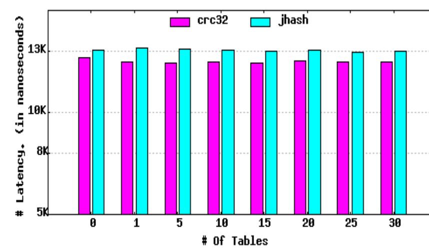
(e) Taxa média de transferencia em configuração ST(iii)



(f) Latência média em configuração ST(iii)



(g) Taxa média de transferencia em configuração ST(iv)



(h) Latência média em configuração ST(iv)

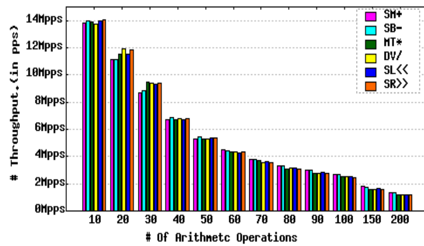
Figura 15 – Desempenho do pipeline DPDK no uso de funções criptográficas.

casos de uso neste cenário.

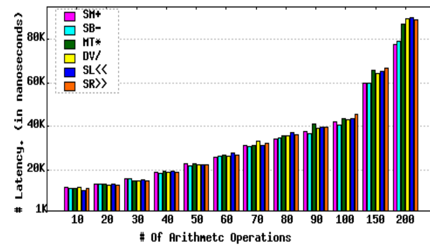
3.2.2.5 DESEMPENHO NO USO DE OPERAÇÕES ARITMÉTICAS

No último cenário a ser analisado, avaliamos o impacto ao aplicar consecutivas operações aritméticas (+,-,*,/,%,« e ») no processamento intensivo com pacotes de 64B

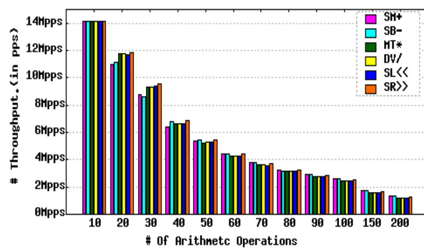
de tamanho. Variamos de 10 até 200 operações consecutivas em cada caso de uso aplicado ao cenário. Nas Figuras 16(a) e 16(b) apresentam os dados registrados pelo cenário em sessão de teste ST(i), para latência e taxa de transferência.



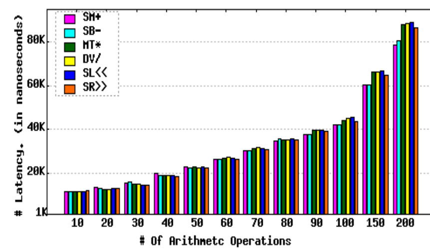
(a) Taxa média de transferência em configuração ST(i)



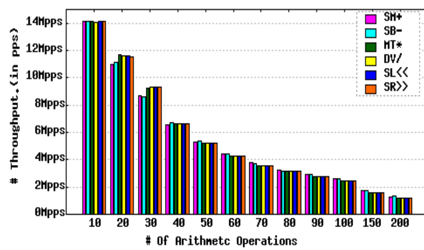
(b) Latência média em configuração ST(i)



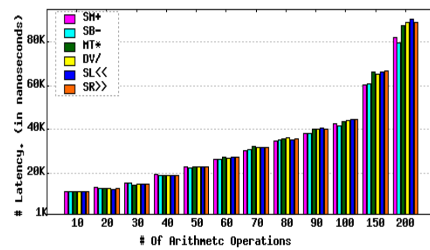
(c) Taxa média de transferência em configuração ST(ii)



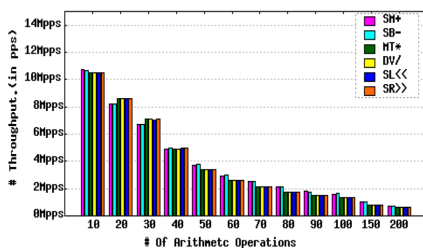
(d) Latência média em configuração ST(ii)



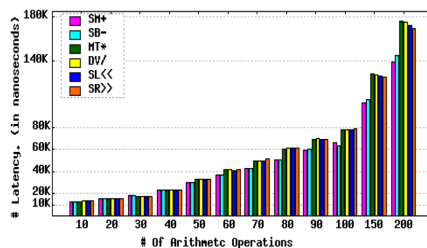
(e) Taxa média de transferência em configuração ST(iii)



(f) Latência média em configuração ST(iii)



(g) Taxa média de transferência em configuração ST(iv)



(h) Latência média em configuração ST(iv)

Figura 16 – Desempenho do pipeline DPDK para consecutivas operações aritméticas.

Para os casos de uso com operações aritméticas no pipeline, observamos o fato do cenário não registrar valores desiguais em latência e taxa de transmissão em STs (i),

(ii) e (iii). Na Figura 16(a), os valores de referência apresenta uma degradação na taxa de transferência para operações de soma (13.83 Mpps de 14.18 Mpps) e divisão (13.70 Mpps de 14.18 Mpps) ao realizar 10 operações. Com um tráfego de 10 Gbits (14.18 Mpps), o cenário em ST(ii) e ST(iii), não apresentam redução no desempenho em transferência com 10 operações executadas.

Em termos de latência, nas três STs o cenário alcança resultados aceitáveis (até 15000ns), isso ao executar 10 operações. No entanto, o aumento no número de operações aritméticas faz com que o custo em latência e taxa de transferência aumentem exponencialmente. Por exemplo, operações de soma, ao executar consecutivamente 40 vezes no pipeline, obteve degradação, em transferência, 48% maior do que, com 10 execuções de adição. Em latência, as 40 operações de adição resultam em acréscimo de 36%, comparando com o valor de latência no realizar 10 vezes uma soma.

O cenário de operações aritméticas apresenta considerável aumento do custo de latência e taxa de transferência na sessão de testes ST(iv). Nas Figuras 16(g) e 16(h) observamos que o ganho em transferência do cenário em ST(iv) é 24% menor que o valor obtido nas demais STs. Os valores de latência com 10 operações no pipeline, o cenário em ST(iv), registrou latência 9% maior do que o resultado de referência (11874ns). Com o número de operações 10 vezes maior (ou seja, 100 operações), nosso valor de referência é de 42154ns para operações de adição. O resultado logrado em ST(iv), para 100 vezes uma soma é 36% maior (65930ns).

3.3 DISCUSSÃO

Nesta seção, argumentamos as lições aprendidas no decorrer de nosso trabalho. O P4 permite que os desenvolvedores escrevam programas PDP descrevendo o comportamento de processamento de pacotes em dispositivos de rede programáveis, independente do hardware desejado. No entanto, depende de um compilador capaz de traduzir esses programas P4 em código apropriado para o hardware de destino, seja um switch de rede, um FPGA (Field-Programmable Gate Array) ou mesmo um pipeline DPDK.

O P4C realiza várias etapas de compilação e otimização para transformar os programas P4 em código executável. Uma dessas etapas é gerar o código de saída específico do hardware (alvo) de destino. Para tal feito, o compilador P4C utiliza de backends para suportar diferentes arquiteturas de hardware e ambientes de execução. Esses backends podem ser disponibilizados pelo fabricante de hardwares programáveis, ou mesmo por projetos interessados em ser uma opção de alvo programável.

Em nosso trabalho, utilizamos na maior parte de nossos cenários de testes, uma aplicação PDP escrita em linguagem P4 para um alvo DPDK. Para que isso fosse possível, utilizamos o P4C em conjunto do backend DPDK obtendo o código executável compatível com o alvo desejado. No entanto, levamos em consideração certas limitações da ferramenta. Citamos abaixo alguns dos problemas identificados:

- O destino DPDK não suporta instruções nos blocos de saída, sendo necessário limitar as funcionalidades do programa nos blocos de entrada.

- As estruturas de dados declaradas no programa, devem conter as variáveis com tamanho múltiplo de 8 bits até o máximo de 64 bits.

- Alguns recursos e instâncias externas não são suportadas com exemplos de, hash, packet recirculation, random, basic checksum e outros.

Todas as atuais limitações podem ser vistas no github da ferramenta. O DPDK, pode ser considerado um software acelerador de pacotes de rede. Atualmente NICs de alta velocidade, como 10GbE, 25GbE, 40GbE e 100GbE tem amplo uso por modernos servidores em data center. A finalidade do DPDK, portanto, é prover alto desempenho e escalabilidade no processamento de pacotes de rede. A possibilidade de aplicar o DPDK em hardware de uso geral faz com que essa ferramenta seja amplamente utilizada na infraestrutura de nuvem e rede de telecomunicações. No entanto, otimizações utilizando DPDK podem ser inviáveis, por incompatibilidade de hardware, especificamente com NICs. Onde haja compatibilidade, para obter o melhor desempenho, requisitos de hardware e memória devem ser consideradas.

A seguir, listamos alguns pontos relevantes:

- **Canais de Memória:** Os modernos controladores de memória portam vários canais capazes de armazenar ou carregar dados em paralelo. Dependendo do controlador e da sua configuração, o número de canais e a maneira como é distribuída a memória por estes, é diversificada. Cada canal de memória possui um limite de largura de banda, assim caso as operações de acesso à memória seja feita apenas em um canal, teremos então um possível gargalo. Garantir de que cada canal de memória possua pelo menos uma memória DIMM de tamanho igual ou maior que 4 GB, tem dos efeitos mais diretos no desempenho. Deve-se considerar determinar a frequência de trabalho da memória, pois isso irá impactar na largura de banda de acesso. Ao executar uma aplicação DPDK por meio da EAL, a linha de comando deve sempre ter o número de canais de memória especificados para os lcores, embora o DPDK disponibilize do pool de memória, sendo isso um alocador de objeto de tamanho fixo para armazenar objetos livres, em um padrão baseado em anel. Também deve-se levar em consideração outros aspectos, como um cache de objeto por núcleo e um auxiliar de alinhamento dos objetos, garantindo que o preenchimento dos canais de DRAM ou DDR sejam distribuídos igualmente. Todo esse esforço é perdido em um cenário com canais de memória únicos. O impacto no desempenho pode ser observado nos resultados obtidos por nossos cenários de testes, conforme maior o número de lcores das sessões STs realizadas. Dessa forma, o baixo quantitativo de

canais de memória de um setup de teste impõem resistência no desempenho mesmo com vários lcores de processamento.

- **Placas de interface de rede (NIC):** Um ponto limitante é o suporte na NIC ao DPDK. Quando a NIC é compatível, o desempenho está relacionado com o firmware da interface de rede e sua versão (como por exemplo, o uso de slots PCIe de terceira geração de x8 ou x16, principalmente se a NIC for de 2x 10GbE ou mais). Além disso, se faz desejável que a NIC esteja no mesmo soquete da CPU.

- **Configuração de BIOS:** Ajustes de configurações no BIOS do sistema pode melhorar o desempenho por permitir otimização ou eficiência energética. Revisar a política de "CPU Power and Performance", usar o "Turbo Boot" (para CPU Intel) ou "Turbo Core" (para CPU AMD) e desativar opções de virtualização ao usar funções físicas da interface de rede são pontos razoáveis de análise.

- **Inicialização do sistema Linux:** O uso de sistemas, Linux, é recomendado o uso do arquivo GRUB para inicializar configurações de reserva como, quantidade de páginas grandes e a escolha dos núcleos da CPU para uso exclusivo pelo DPDK.

Podemos inferir que existe necessidade que o desenvolvedor possua conhecimento ou reúna informações sobre o hardware alvo da otimização. Analisar as dimensões do hardware possibilita desenvolver um software com melhor desempenho. O conjunto de bibliotecas e drivers otimizados, o uso eficiente de memória com páginas grandes e o desvio da pesada pilha de rede do kernel Linux, tem por garantir um melhor desempenho de programas DPDK. No entanto, além das nativas otimizações, DPDK também permite otimizar via parâmetros EAL e utilizar ou não as técnicas de codificação, que podem impactar no desempenho.

- **Correlação latência - throughput:** O equilíbrio nativo realizado pelo DPDK entre latência e taxa de transferência em uma aplicação pode ser personalizada pelo desenvolvedor. O padrão exercido pelo DPDK para um maior rendimento, está na tentativa de integrar o custo individual no processar pacotes, executando um processamento de pacotes em rajadas (o valor padrão das rajadas é de 32 pacotes). A aplicação processa cada pacote recebido e armazena na fila TX da porta de rede correspondente, que fica no aguardo dos demais 31 pacotes para serem transmitidos. Esse comportamento é esperado para uma alta taxa de transferência, onde distribui em 32 pacotes o custo das atualizações das filas RX e TX. O processamento em rajadas oculta de forma satisfatória o maior custo MMIO (Mapeamento de Memória de Entrada/Saida) para gravações em dispositivos PCIe. O procedimento de rajadas atribui melhor desempenho ao custo de uma maior latência, onde o primeiro pacote processado de uma rajada de 32 pacotes aguarda que os outros 31 pacotes sejam

recebidos. É possível obter uma baixa latência, mesmo sob pesada carga de processamento, alterando o valor de pacotes de uma rajada. Se alterarmos o valor padrão de intermitência de 32 pacotes para 1, permite que um único pacote seja processado por vez, e que por sua vez provê de uma menor latência. No entanto, uma menor latência ao processar e transmitir um único pacote tem por agregar uma menor taxa de transferência.

- **Alinhamento de memória na arquitetura x86:** Conforme a configuração de memória em hardware de arquitetura x86, podemos aprimorar de forma significativa o desempenho de aplicações DPDK, ao inserir preenchimentos específicos entre os objetos armazenados na memória. Essa inserção tem por objetivo garantir que o início de cada objeto inicie em um canal e classificação distinta na memória, garantindo o igual carregamento dos canais. Isso é desejável em buffers de pacotes ao realizar encaminhamento L3 ou classificar fluxo, onde apenas os 64 bytes do início são acessados. Dessa forma é provável um maior desempenho distribuindo os endereços iniciais dos objetos em todos os canais de memória de uso.

4 CONSIDERAÇÕES FINAIS

A programabilidade de redes, que se refere à capacidade de ajustar e personalizar o comportamento da infraestrutura de rede de acordo com as necessidades específicas, desempenha um papel vital na evolução das redes de comunicação. Essa abordagem oferece uma flexibilidade significativa na configuração e no gerenciamento das redes, permitindo a introdução eficiente de novos serviços, otimização de roteamento e adaptação ágil às mudanças nas demandas de tráfego. Tecnologias como P4, eBPF e DPDK têm desempenhado um papel fundamental na realização da programabilidade de redes, proporcionando maior eficiência e desempenho.

Esse trabalho propõe uma extensa avaliação de escalabilidade e desempenho de aplicações PDP executadas em pipeline DPDK, nos permitiu entender e quantificar limitações e dos benefícios deste uso. Consideramos satisfatório os resultados obtidos por nossos testes, para taxa de transferência e latência, uso intensivo de processamento e espaço em memória, em hardware não específico. Foi possível avaliar o desempenho de aplicações PDP, variando o número de lcores de processamento, onde identificamos o ponto restritivo no desempenho em um setup de hardware com poucos canais de memória.

Um setup de poucos canais de memória demonstrou ser uma limitação significativa, podendo ser observado o sucedido em todas as sessões de testes realizadas.No aplicar escalabilidade variando o numero de lcores de processamento observamos nos resultados obtido uma perda no desempenho na taxa de transferência e uma maior latência no executar as aplicações com maior número de lcores entre as sessão de teste.A concorrência dos cores de processamento para utilizar dos canais de memória disponíveis, faz com que o custo em tempo para o acesso em memória seja maior em um cenário multicore.No uso de programas escritos em linguagem P4 e compilados para um alvo DPDK, o backend DPDK com compilador P4C atendeu em 71% em termos de compatibilidade de código entre esses dois níveis (P4 e DPDK).

Em resumo, a pesquisa e o desenvolvimento contínuo no campo da programabilidade de redes são cruciais para enfrentar os desafios impostos pelo aumento constante do tráfego de dados e pela complexidade crescente dos serviços de rede. As vantagens oferecidas por tecnologias como o DPDK são evidentes, pois possibilita uma programação mais eficaz e personalizada das infraestruturas de rede, contribuindo para um desempenho superior e uma maior eficiência operacional. No entanto, é importante reconhecer que ainda existem desafios a serem superados, como a definição clara dos limites de desempenho do DPDK e a consideração da escalabilidade e interoperabilidade em ambientes de rede complexos. Superar esses desafios requer pesquisa contínua e colaboração entre a comunidade de redes e a indústria.

Em última análise, a capacidade de programar e personalizar a infraestrutura de rede de acordo com requisitos específicos é fundamental para o sucesso das redes modernas. À medida que a demanda por serviços de comunicação de alta qualidade continua a crescer,

a programabilidade de redes desempenha um papel central na construção de infraestruturas de rede mais eficientes, adaptáveis e preparadas para o futuro. Como trabalhos futuros, nos motiva da possibilidade de avaliar programas PDP em pipeline DPDK, levando em consideração o impacto no desempenho, no utilizar parâmetros EAL habiliatando e/ou ajustando funcionabilidades do hardware ou do software, técnicas e recursos de codificação descritos na documentação da ferramenta e se possível, diferentes setup com hardwares que disponibilize mais canais de memórias e outras interfaces de rede disponives no mercado, ou até mesmo um setup virtualizado.Com todas as aplicações escritas em linguagem P4 compiladas via P4C com backend DPDK.

REFERÊNCIAS

AFFINITY, D. lcore. *DPDK lcore affinity*. 2023. <https://doc.dpdk.org/guides/prog_guide/env_abstraction_layer.html#eal-pthread-and-lcore-affinity>. [Online; accessed 20-June-2023]. Citado na página 36.

API p4c-dpdk translate P4-16 programs to D. *p4c-dpdk translate P4-16 programs to DPDK API*. 2023. <https://github.com/p4lang/p4c/tree/main/backends/dpdk>. [Online; accessed 20-June-2023]. Citado na página 37.

APPLICATION, D. R. the. *DPDK Running the Application*. 2023. <https://doc.dpdk.org/guides/testpmd_app_ug/run_app.html>. [Online; accessed 20-June-2023]. Citado na página 37.

BEGIN, T. et al. An accurate and efficient modeling framework for the performance evaluation of dpdk-based virtual switches. *IEEE Transactions on Network and Service Management*, v. 15, n. 4, p. 1407–1421, Dec 2018. ISSN 1932-4537. Citado na página 33.

BOSSHART, P. et al. P4: Programming protocol-independent packet processors. *SIGCOMM Comput. Commun. Rev.*, Association for Computing Machinery, New York, NY, USA, v. 44, n. 3, p. 87–95, jul 2014. ISSN 0146-4833. Citado na página 22.

ELENBOGEN, B. S. Computer network management: Theory and practice. In: *The Proceedings of the Thirtieth SIGCSE Technical Symposium on Computer Science Education*. New York, NY, USA: Association for Computing Machinery, 1999. (SIGCSE '99), p. 119–121. ISBN 1581130856. Citado na página 15.

GENERATOR, M. P. *Moogen Packet Generator*. 2023. <https://github.com/emmericp/MoonGen>. [Online; accessed 20-June-2023]. Citado na página 37.

GOSWAMI, B.; KULKARNI, M.; PAULOSE, J. A survey on p4 challenges in software defined networks: P4 programming. *IEEE Access*, IEEE, v. 11, p. 54373–54387, May 2023. Citado na página 16.

KOURTIS, M.-A. et al. Enhancing vnf performance by exploiting sr-iov and dpdk packet processing acceleration. In: *2015 IEEE Conference on Network Function Virtualization and Software Defined Network (NFV-SDN)*. [S.l.: s.n.], 2015. p. 74–78. Citado na página 34.

LANGUAGE, R. compiler for P. *Reference compiler for P4 language*. 2023. <https://github.com/p4lang/p4c>. [Online; accessed 20-June-2023]. Citado na página 37.

LPM, S. l2l3. *Simple l2l3 lpm*. 2023. <https://github.com/p4lang/p4-dpdk-target/blob/main/examples/psa/simple_l2l3_lpm/simple_l2l3_lpm.p4>. [Online; accessed 20-June-2023]. Citado na página 36.

NEVES, M. et al. Dynamic property enforcement in programmable data planes. *IEEE/ACM Transactions on Networking*, IEEE, v. 29, n. 4, p. 1540–1552, April 2021. Citado na página 16.

NIKOLICH, A. Sdn research challenges and opportunities. In: *Proceedings of the Sixth ACM Conference on Data and Application Security and Privacy*. New York, NY, USA: Association for Computing Machinery, 2016. (CODASPY '16), p. 254. ISBN 9781450339353. Citado na página 20.

OEVER, N. ten; BERALDO, D. Routes to rights: Internet architecture and values in times of ossification and commercialization. *XRDS*, Association for Computing Machinery, New York, NY, USA, v. 24, n. 4, p. 28–31, jul 2018. ISSN 1528-4972. Citado na página 19.

OVERVIEW, D. A. *DPDK Architecture Overview*. 2023. <https://doc.dpdk.org/guides/prog_guide/overview.html>. [Online; accessed 20-June-2023]. Citado na página 27.

OVERVIEW, D. E. A. L. *DPDK Environment Abstraction Layer Overview*. 2023. <https://doc.dpdk.org/guides/prog_guide/env_abstraction_layer.html>. [Online; accessed 20-June-2023]. Citado na página 37.

SHALIMOV, A. et al. Advanced study of sdn/openflow controllers. In: *Central and Eastern European Software Engineering Conference in Russia*. New York, NY, USA: Association for Computing Machinery, 2013. (CEE-SECR '13). ISBN 9781450326414. Citado na página 15.

SHEN, H. et al. A method for performance optimization of virtual network i/o based on dpdk-sriov. In: *2018 IEEE International Conference on Information and Automation (ICIA)*. [S.l.: s.n.], 2018. p. 1550–1554. Citado na página 33.

VLADISLAVIć, ; HULJENIć, D.; OžEGOVIć, J. Enhancing vnf's performance using dpdk driven ovs user-space forwarding. In: *2017 25th International Conference on Software, Telecommunications and Computer Networks (SoftCOM)*. [S.l.: s.n.], 2017. p. 1–5. Citado na página 33.