

INSTITUTO FEDERAL DO ESPÍRITO SANTO
PROGRAMA DE PÓS-GRADUAÇÃO EM COMPUTAÇÃO APLICADA

GIANCARLO O. DOS SANTOS

**MTS-POLKA: DIVISÃO DE TRÁFEGO MULTICAMINHOS EM PROPORÇÃO DE
PESO COM ROTEAMENTO NA FONTE**

GIANCARLO O. DOS SANTOS

**MTS-POLKA: DIVISÃO DE TRÁFEGO MULTICAMINHOS EM PROPORÇÃO DE
PESO COM ROTEAMENTO NA FONTE**

Dissertação apresentada ao Programa de Pós-Graduação em Computação Aplicada do Instituto Federal do Espírito Santo, Campus Serra, como requisito parcial para a obtenção do título de Mestre em Computação Aplicada.

Orientadora: Prof.^a Dr.^a Cristina Klippel Dominicini

Orientador: Prof. Dr. Gilmar L. Vassoler

Dados Internacionais de Catalogação na Publicação (CIP)

S237m Santos, Giancarlo Oliveira dos
2024 MTS-PoLKA : divisão de tráfego multicaminhos em proporção de
peso com roteamento na fonte / Giancarlo Oliveira dos Santos. - 2024.
68 f.; il.; 30 cm

Orientador: Prof.^a Dr.^a Cristina Klippel Dominicini ; Prof. Dr. Gilmar L. Vassoler.

Dissertação (mestrado) - Instituto Federal do Espírito Santo,
Programa de Pós-graduação em Computação Aplicada, 2024 .

1. Datacenter. 2. Multipath Traffic Splitting (MTS). 3. Polynomial Key-based Architecture (PoLKA). 4. Divisão de tráfego. 5. Roteamento da fonte. I. Dominicini, Cristina Klippel. II. Instituto Federal do Espírito Santo. III. Título.

CDD 004

Bibliotecário: Valmir Oliveira de Aguiar - CRB6/ES 566

GIANCARLO OLIVEIRA DOS SANTOS

MTS-POLKA: DIVISÃO DE TRÁFEGO MULTICAMINHOS EM PROPORÇÃO DE PESO COM ROTEAMENTO NA FONTE

Dissertação apresentada ao Programa de Pós-Graduação em Computação Aplicada do Instituto Federal do Espírito Santo, Campus Serra, como requisito parcial para a obtenção do título de Mestre em Computação Aplicada.

Aprovado em 15 de outubro de 2024.

COMISSÃO EXAMINADORA

Prof.^a Dr.^a Cristina Klippel Dominicini
Instituto Federal do Espírito Santo
Orientadora

Prof. Dr. Gilmar L. Vassoler
Instituto Federal do Espírito Santo
Orientador

Prof. Dr. Leandro Colombi Resendo
Instituto Federal do Espírito Santo
Membro Interno

Prof. Dr. Vinícius Fernandes Soares Mota
Universidade Federal do Espírito Santo
Membro Externo

FOLHA DE ROSTO Nº 72/2024 - SER-CGEN (11.02.32.01.08.02)

(Nº do Protocolo: NÃO PROTOCOLADO)

(Assinado digitalmente em 20/10/2024 12:04)

CRISTINA KLIPPEL DOMINICINI

PROFESSOR DO ENSINO BASICO TECNICO E TECNOLOGICO

SER-CGEN (11.02.32.01.08.02)

Matrícula: 2154146

(Assinado digitalmente em 21/10/2024 16:47)

GILMAR LUIZ VASSOLER

DIRETOR

SER (11.02.32)

Matrícula: 1544688

(Assinado digitalmente em 20/10/2024 15:09)

LEANDRO COLOMBI RESENDO

PROFESSOR DO ENSINO BASICO TECNICO E TECNOLOGICO

SER-CGEN (11.02.32.01.08.02)

Matrícula: 1687072

Visualize o documento original em <https://sipac.ifes.edu.br/documentos/> informando seu número: 72, ano: 2024, tipo: FOLHA DE ROSTO, data de emissão: 20/10/2024 e o código de verificação: a1542def8a

Dedico este trabalho à minha família, pela base sólida de amor e apoio incondicional; à minha noiva, pela paciência, carinho e incentivo em cada passo desta jornada; à minha filha, fonte de inspiração e alegria diária; aos meus orientadores, pelo conhecimento compartilhado e orientação fundamental; e aos meus amigos, pela companhia e compreensão nos momentos mais desafiadores. A cada um de vocês, minha eterna gratidão.

AGRADECIMENTOS

Gostaria de expressar minha profunda gratidão a Deus, cuja luz e força me acompanharam em cada etapa desta jornada. Sua presença constante me proporcionou coragem nas adversidades e sabedoria nas decisões cruciais. Agradeço à minha noiva e à minha filha, que foram fontes constantes de apoio e motivação. O amor e a alegria que elas trazem para a minha vida são inestimáveis, sempre me lembrando do propósito maior por trás de meus esforços.

Agradeço também à minha família pelo amor incondicional e encorajamento. Cada gesto de carinho e palavra de apoio foram fundamentais para que eu seguisse em direção aos meus objetivos; vocês são minha base e minha maior inspiração.

Em memória dos meus avós, sou profundamente grato por tudo o que fizeram por mim. Sua sabedoria e amor moldaram quem sou hoje, e cada momento ao lado deles trouxe lições valiosas sobre a vida e a importância da família. Sou eternamente grato por sua influência positiva na minha trajetória.

Um agradecimento especial vai para minha orientadora, Cristina, cuja orientação foi crucial para eu alcançar este estágio. Sua presença constante, apoio inabalável e cobranças construtivas me motivaram a buscar melhorias e a superar meus limites. Valorizo cada conselho, cada correção e, acima de tudo, por ter acreditado em mim e me incentivado a não desistir nos momentos desafiadores.

Ao meu co-orientador, Gilmar, expresso minha gratidão por sua orientação e apoio ao longo da minha trajetória acadêmica. Sua crença em meu potencial e seu conhecimento foram essenciais para que eu superasse obstáculos e alcançasse novos patamares. Sem seu incentivo, não teria conseguido os resultados que celebro hoje.

Aos meus colegas de jornada, Isis, Domingos e Luciano, agradeço pela camaradagem e pelas trocas enriquecedoras ao longo do mestrado. A colaboração que cultivamos foi fundamental para o sucesso da minha pesquisa, tornando esta experiência ainda mais significativa.

Por fim, reconheço com gratidão todos os professores que desempenharam um papel crucial na minha formação. Agradeço especialmente a Rodolfo Villaça e Rafael Silva Guimarães pelo apoio e sugestões durante esta jornada. Sua dedicação e expertise foram essenciais para meu crescimento e suas contribuições continuarão a ressoar em minha trajetória.

Que este agradecimento reflita a imensa gratidão que sinto por todos que tornaram esta jornada possível.

"O bater das asas de uma borboleta no Brasil pode causar um tornado no Texas."
(Provérbio Popular sobre o Efeito Borboleta)

RESUMO

A crescente demanda por eficiência em redes de *datacenters* motivou a pesquisa sobre métodos inovadores de otimização de tráfego. As abordagens tradicionais, como *ECMP* (*Equal-Cost Multi-Path*) e *WCMP* (*Weighted-Cost Multi-Path*), apresentam limitações significativas na adaptação a flutuações de tráfego, resultando em subutilização de recursos e ineficiências operacionais. Já o *MP-TCP* (*MultiPath TCP*), que divide cada fluxo em vários subfluxos na camada *TCP* para melhorar a utilização da largura de banda e a resiliência, enfrenta desafios relacionados à complexidade na coordenação de subfluxos e ao aumento da sobrecarga de controle. Diante desse cenário, a dissertação propõe o *MTS-PolKA*, uma solução inovadora que combina *Multipath Traffic Splitting* (MTS) com o protocolo de roteamento *PolKA* (Polynomial Key-based Architecture). A versão modificada, *M-PolKA*, em que o *M* refere-se a *Multipath*, permite a exploração dinâmica de multicaminhos por meio de rótulos (*routeIDs* e *weightID*) inseridos nos cabeçalhos dos pacotes, possibilitando ajustes em tempo real sem a necessidade de reconfigurações complexas nos *switches*. Essa capacidade de dividir e distribuir o tráfego de maneira eficiente entre diferentes rotas maximiza o uso da infraestrutura de rede e melhora o desempenho geral. A proposta se baseia no roteamento na fonte, empregando um Protocolo *M-PolKA* modificado e um sistema numérico de resíduos, o que possibilita o roteamento de fonte sem armazenamento de estado e sem alterações nos *hosts* finais. O *MTS-PolKA* destaca-se pela agilidade na reconfiguração de caminhos e pesos, permitindo que a origem do tráfego selecione perfis de divisão de forma eficiente e flexível. Os resultados obtidos através de experimentos demonstram a eficácia do *MTS-PolKA*, evidenciando melhorias significativas no desempenho e na eficiência das redes de *datacenters*. A solução não apenas otimiza a distribuição de tráfego, mas também oferece flexibilidade e estabilidade, apresentando-se como uma contribuição valiosa para a engenharia de tráfego em ambientes complexos. Assim, a pesquisa reafirma a importância de inovações na engenharia de tráfego para atender às crescentes demandas da área.

Palavras-chave: Datacenter. Multicaminhos. Divisão de tráfego. Roteamento na fonte.

ABSTRACT

The growing demand for efficiency in data center networks has driven research into innovative traffic optimization methods. Traditional approaches, like ECMP (Equal-Cost Multi-Path) and WCMP (Weighted-Cost Multi-Path), have significant limitations in adapting to traffic fluctuations, leading to resource underutilization and operational inefficiencies. On the other hand, MP-TCP (MultiPath TCP), which breaks each flow into multiple sub-flows at the TCP layer to enhance bandwidth usage and resilience, faces challenges related to the complexity of coordinating sub-flows and increased control overhead. In light of this scenario, the thesis proposes *MTS-PolKA*, an innovative solution that combines Multipath Traffic Splitting (MTS) with the PolKA (Polynomial Key-based Architecture) routing protocol. The modified version, M-PolKA, where the M stands for Multipath, enables the dynamic exploration of multiple paths through labels (routeIDs and weightID) inserted in the packet headers, allowing for real-time adjustments without the need for complex reconfigurations on switches. This ability to effectively split and distribute traffic across different routes maximizes network infrastructure utilization and improves overall performance. The proposal is based on source routing, employing a modified M-PolKA Protocol and a numerical residue system, facilitating source routing without state storage and without changes to the end hosts. *MTS-PolKA* stands out for its agility in (re)configuring paths and weights, enabling traffic sources to efficiently and flexibly select splitting profiles. Results obtained through experiments demonstrate the effectiveness of *MTS-PolKA*, highlighting significant improvements in performance and efficiency in data center networks. The solution not only optimizes traffic distribution but also offers flexibility and stability, making it a valuable contribution to traffic engineering in complex environments. Thus, the research reaffirms the importance of innovations in traffic engineering to meet the increasing demands in the field.

Keywords: Data center. Multipath. Traffic splitting. Source routing.

LISTA DE FIGURAS

Figura 1 – Divisão assimétrica de tráfego multicaminho em topologias de árvore multi-raiz do tipo <i>Clos Tier 2</i> . (a) <i>ECMP</i> : cada fluxo é atribuído a um único caminho de custo igual. (b) <i>WCMP</i> : cada fluxo é atribuído a um único caminho, e a distribuição de fluxo entre os caminhos pode ser configurada por tabelas de grupos ponderados em cada <i>switch</i> (estrelas vermelhas). Fluxos elefante não podem ser divididos entre múltiplos links. (c) <i>MTS-PolKA</i> : Fluxos podem ser divididos entre vários caminhos de acordo com os pesos definidos pela fonte (estrela azul) e inseridos como um rótulo no cabeçalho do pacote. Modificações de caminho requerem apenas uma mudança neste rótulo, e os <i>switches</i> precisam apenas de tabelas estáticas.	15
Figura 2 – Problema: Como configurar os <i>switches</i> para permitir a distribuição ponderada de um fluxo em multicaminhos com granularidade de pacotes?	17
Figura 3 – Arquitetura SDN	21
Figura 4 – Principais componentes de um <i>switch Openflow</i>	22
Figura 5 – Switches Tradicionais e Switches Programáveis (P4)	23
Figura 6 – Arquitetura PISA. Adaptado de (BOSSHART et al., 2014)	24
Figura 7 – Fluxo de implementação do programa P4. Adaptado de (P4.org, 2021)	24
Figura 8 – Exemplo de roteamento na fonte sem uso de tabelas	26
Figura 9 – Exemplo do roteamento <i>PolKA</i> original	27
Figura 10 – Exemplo do roteamento M-PolKA original	29
Figura 11 – Cabeçalho M-PolKA com comprimento fixo	30
Figura 12 – Pipeline do Núcleo P4 para M-PolKA	31
Figura 13 – Encaminhamento multicaminho - WCMP	33
Figura 14 – Trabalhos relacionados: exemplo ilustrando as filas de links para um cenário com fluxos de tamanhos comparáveis e links simétricos (Adaptado de (HUANG; LYU et al., 2021))	35
Figura 15 – Trabalhos relacionados: exemplo ilustrando as filas de links para um cenário com fluxos de elefantes e ratos e links assimétricos	36
Figura 16 – Arquitetura <i>MTS-PolKA</i>	39
Figura 17 – Exemplo de funcionamento do <i>MTS-PolKA</i> com perfil de divisão de tráfego selecionado pela origem com um rótulo no cabeçalho do pacote. Ambas as tabelas são configuradas previamente pelo controlador	41
Figura 18 – <i>Pipeline</i> na linguagem P4 dos <i>switches</i> de núcleo do <i>MTS-PolKA</i>	43
Figura 19 – Ingresso de um pacote na rede com <i>routeID</i> = 110110 e <i>weightID</i> = 001011	45
Figura 20 – Mostrando a operação de módulo e as portas ativas resultantes	45
Figura 21 – Cálculo do <i>profileID</i> e a distribuição do tráfego entre as portas ativas	46
Figura 22 – Seleção do perfil de peso (<i>profileID</i>) na <code>table static profiles</code>	46
Figura 23 – Tabelas Estáticas	47

Figura 24 – Tabelas estáticas <code>table static profiles</code> e <code>table multipath</code>	48
Figura 25 – Soma do <i>hash</i> do <i>timestamp</i> com o índice de entrada da tabela <code>table static profiles</code>	48
Figura 26 – Definição da Porta de saída	49
Figura 27 – Definição da Porta de saída	49
Figura 28 – Aplicação do algorítimo de seleção das portas	50
Figura 29 – Teste 1: perfil 0	53
Figura 30 – Teste 1: perfil 100	54
Figura 31 – Teste 1: perfil 1011	54
Figura 32 – Teste 2: Resultado	55
Figura 33 – Teste 3: Cenário	57
Figura 34 – Teste 3: Resultado.	58
Figura 35 – Teste 4: Cenário (<i>MTS-PoLKA</i> configuração de divisão de tráfego).	60
Figura 36 – Teste 4: Comparação do Tempo de Conclusão do Fluxo com trabalhos relacionados.	60
Figura 37 – Teste 4: Comparação com trabalhos relacionados à retransmissão de TCP para o fluxo elefante 1.	61
Figura 38 – Teste 4: Comparação de latências com trabalhos relacionados.	63

LISTA DE TABELAS

Tabela 1 – Trabalhos Relacionados de Divisão de Tráfego em Redes de Datacenter 34

SUMÁRIO

1	INTRODUÇÃO	14
1.1	PROBLEMA	15
1.2	PROPOSTA	18
1.3	OBJETIVOS	20
1.3.1	Objetivo Geral	20
1.3.2	Objetivos Específicos	20
1.4	ORGANIZAÇÃO DO TRABALHO	20
2	REFERENCIAL TEÓRICO	21
2.1	ARQUITETURA DE REDE DEFINIDA POR SOFTWARE	21
2.1.1	Primeira Geração SDN: Protocolo OpenFlow	22
2.1.2	Segunda Geração SDN: Linguagem P4	23
2.2	ROTEAMENTO NA FONTE	25
2.2.1	List-based Source Routing (LSR)	26
2.2.2	KeyFlow	26
2.2.3	PolKA	27
2.2.4	M-PolKA	29
3	TRABALHOS RELACIONADOS	32
4	PROPOSTA	38
4.1	ROTEAMENTO M-POLKA	38
4.2	ARQUITETURA DO <i>MTS-POLKA</i>	39
4.2.1	Plano de Controle	40
4.2.2	Plano de Dados	41
4.3	EXEMPLO DETALHADO SOBRE O FUNCIONAMENTO DO <i>MTS-POLKA</i>	44
4.3.1	Introdução ao Funcionamento	44
4.3.2	Ingresso do Pacote	44
4.3.3	Definição de Rota	45
4.3.4	Seleção de Perfil	45
4.3.5	Distribuição de Tráfego	46
5	PROVA DE CONCEITO	51
5.1	PROTÓTIPO	51
5.2	EXPERIMENTOS	51
5.2.1	Teste 1: Validação Funcional	52
5.2.2	Teste 2: Reconfiguração Ágil	54
5.2.3	Teste 3: Múltiplos fluxos e concorrência	56
5.2.4	Teste 4: Comparação com Trabalhos Relacionados sobre o Tempo de Conclusão de Fluxo para Fluxos Elefante e Análise de Latência para Fluxos Rato	59
5.2.5	Limitações do protótipo	62

6	RESULTADOS E DISCUSSÕES	64
7	CONCLUSOES E TRABALHOS FUTUROS	65
	REFERÊNCIAS	66

1 INTRODUÇÃO

Nos últimos anos, os *datacenters* se tornaram os pilares da infraestrutura de computação da Internet. Inúmeros aplicativos de processamento distribuído, como colaboração social, pesquisa e computação de alto desempenho, são executados diariamente em *datacenters* de grande escala que contêm mais de 100.000 servidores (DIXIT; PRAKASH et al., 2013). Os *datacenters* modernos são comumente organizados em topologias de árvore multi-raiz, como *Fat-Tree* e *Clos*, para fornecer vários caminhos com o mesmo custo entre esses servidores (HUANG; LYU et al., 2021). As operadoras de rede geralmente garantem vários caminhos para atender às demandas de tráfego de seus aplicativos, seja em *datacenters* ou redes de longa distância.

No entanto, utilizar toda a largura de banda disponível requer uma maneira eficaz de equilibrar o tráfego, especialmente quando a carga nesses caminhos pode se alterar rapidamente devido às flutuações de tráfego. Portanto, conseguir balancear a carga em uma pequena escala de tempo, com base nas condições de tráfego dos caminhos, torna-se crucial para obter um bom desempenho neste gerenciamento (HSU; TAMMAMA et al., 2020). Assim, para que a arquitetura de rede de *datacenter* alcance um alto desempenho, a chave é a engenharia de tráfego, em que o controlador de rede ou os agentes distribuídos devem ser capazes de selecionar caminhos e balancear a carga entre eles, de forma ágil, para adaptar-se a padrões de tráfego com comportamentos dinâmicos (JYOTHI; DONG; GODFREY, 2015).

Para resolver estes problemas, na última década, algumas tecnologias de redes têm sido propostas para garantir maior confiabilidade e flexibilidade. Diante disto, surgiu a ideia de “redes programáveis” ou redes definidas por software (SDN, do inglês *Software Defined Networking*) (HU et al., 2018b). Usando *SDN*, as funções de rede são implementadas removendo decisões de controle, como roteamento, de dispositivos de *hardware* e permitindo funções programáveis no *hardware*, usando um protocolo padronizado. O balanceamento de Carga é um método para melhorar o desempenho da rede, disponibilidade, minimizando atrasos e evitando rede congestionamento. Dessa forma, os mecanismos de balanceamento de carga em ambientes *SDN* podem ser aplicados de forma flexível em qualquer momento da operação da rede. Antes da falha da rede, isso implica que estratégias de balanceamento de carga e roteamento podem ser implementadas proativamente para evitar congestionamentos ou sobrecargas que poderiam levar a falhas. Além disso, esses mecanismos permitem uma rápida reação após a falha de um *link* e ajudam a evitar congestionamentos e sobrecargas no plano de dados. Essa adaptabilidade é crucial para garantir a eficiência e a continuidade do serviço na rede. (NKOSI; LYSKO; DLAMINI, 2018).

Em síntese, uma engenharia de tráfego eficiente em redes de *datacenter* requer: i) distribuir os fluxos de forma ágil, explorando os multicaminhos, proporcionalmente à largura de banda disponível em cada caminho, e ii) ajustar dinamicamente, também de forma ágil,

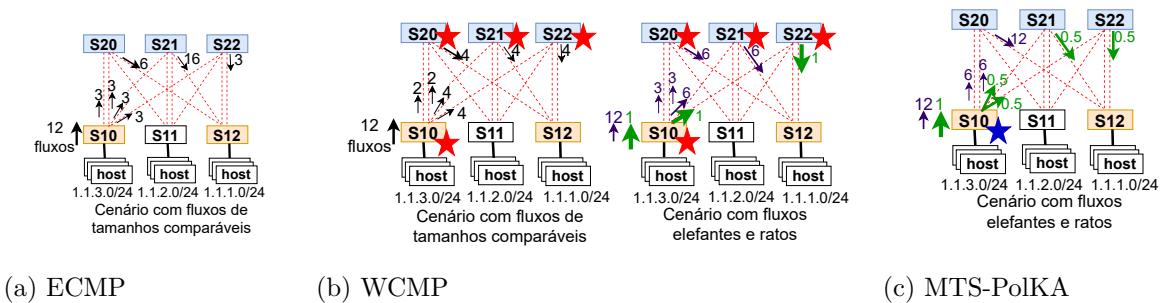
os fluxos por esses caminhos.

Em síntese, uma engenharia de tráfego eficiente em redes de *datacenter* requer: i) distribuir os fluxos de forma ágil, explorando os multicaminhos, proporcionalmente à largura de banda disponível em cada caminho, e ii) ajustar dinamicamente, também de forma ágil, os fluxos por esses caminhos. Assim, a divisão de tráfego é um recurso comumente necessário em redes modernas para balancear o tráfego em vários caminhos ou servidores de rede (ROTTENSTREICH; KANIZO et al., 2018). A proposta aborda a divisão de tráfego multicamino com base em rótulos, trazendo uma solução inovadora que permite um平衡amento de carga eficiente em redes de *datacenter*, alinhada às demandas de flexibilidade e escalabilidade mencionadas neste estudo (SANTOS et al., 2024).

1.1 PROBLEMA

Os algoritmos tradicionais de roteamento baseados em *ECMP* (*Equal Cost Multiple Path*) consideram apenas topologias simétricas e regulares, devido à sua estratégia de roteamento estática baseada apenas em caminhos com a mesma capacidade de banda e mesma quantidade de saltos (HSU; TAMMAMA et al., 2020). Esse requisito não é realista para a maioria dos cenários atuais de redes de *datacenter*, pois pode ser violada pelas seguintes condições (ZHOU; TEWARI et al., 2014): i) mudanças rápidas nas condições de tráfego; ii) ocorrência de falhas nos nós ou enlaces da rede; iii) existência de componentes de rede heterogêneos; e iv) topologias assimétricas.

Figura 1 – Divisão assimétrica de tráfego multicamino em topologias de árvore multi-raiz do tipo *Clos Tier 2*. (a) *ECMP*: cada fluxo é atribuído a um único caminho de custo igual. (b) *WCMP*: cada fluxo é atribuído a um único caminho, e a distribuição de fluxo entre os caminhos pode ser configurada por tabelas de grupos ponderados em cada *switch* (estrelas vermelhas). Fluxos elefante não podem ser divididos entre múltiplos links. (c) *MTS-PolKA*: Fluxos podem ser divididos entre vários caminhos de acordo com os pesos definidos pela fonte (estrela azul) e inseridos como um rótulo no cabeçalho do pacote. Modificações de caminho requerem apenas uma mudança neste rótulo, e os *switches* precisam apenas de tabelas estáticas.



Fonte: Elaboração própria

A Figura 1 ilustra um exemplo de uma topologia de árvore multi-raiz do tipo *Clos Tier 2*, que pode oferecer multicaminhos de mesmo comprimento para um mesmo destino, mas, com diferentes larguras de banda. Isso pode acontecer devido a diferentes capacidades dos enlaces ou condições dinâmicas de concorrência de tráfego, e vai exigir uma distribui-

ção assimétrica de tráfego na rede. Na Figura 1a, é possível perceber que a estratégia de *hashing* do *ECMP* não é capaz de resolver de forma eficiente esse problema. Os pontos fracos deste esquema são bem conhecidos: quando houver colisões na tabela *hash* incorrerá em fluxos utilizando o mesmo caminho e provável desbalanceamento. O *ECMP* também é fraco em topologias de rede assimétricas (VALENTIM et al., 2019b).

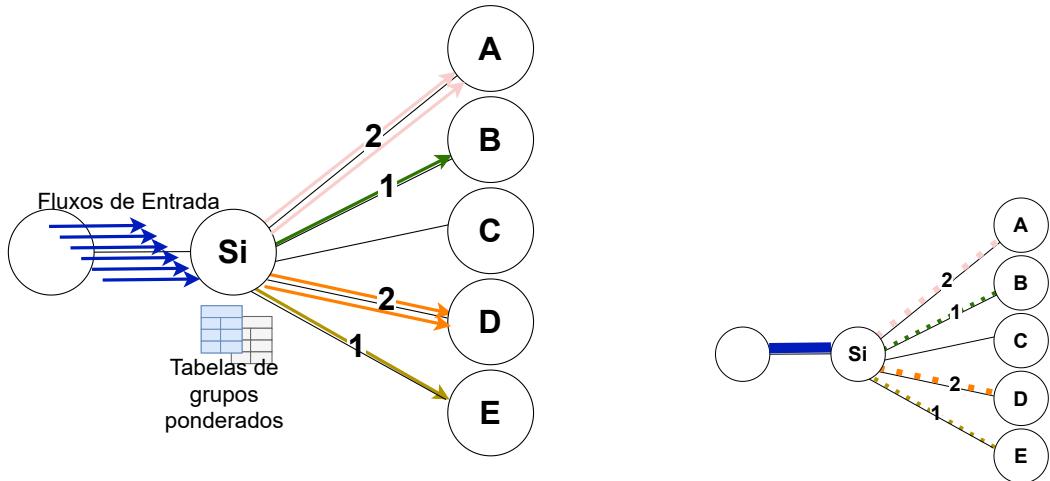
Outras abordagens existentes para resolver este problema são o *WCMP* (*Weighted-Cost MultiPath*) (CUI; HU; HOU, 2021), baseado na configuração de tabelas de divisão de tráfego, e o artigo *DASH* (*Adaptive Weighted Traffic Splitting in Programmable Data Planes*) (HSU; TAMMAMA et al., 2020), baseada em uma estrutura de dados otimizada que explora registradores em um *pipeline* com múltiplos estágios para atualizar a distribuição em tempo real. Conforme mostrado no primeiro cenário da Figura 1b, essas soluções permitem explorar múltiplos caminhos, alocar ativamente o fluxo para o caminho apropriado conforme a razão de peso desejada, e utilizar eficientemente a largura de banda dos enlaces disponíveis. Na busca por uma solução eficiente para a divisão de tráfego em redes, é importante considerar diferentes abordagens.

No entanto, modificar os pesos dos caminhos requer atualizar tabelas ou estruturas de dados em cada mudança ao longo do caminho. Por exemplo, como ilustrado na Figura 1b, isso implica ajustar todas as tabelas nos dispositivos de rede marcados com estrelas vermelhas para cada fluxo que é reorganizado. Quanto mais switches houver no caminho e maior o número de fluxos sendo gerenciados, mais tempo leva para que as tabelas dos dispositivos se convergirem, o que impacta a agilidade da operação.

A Figura 1c ilustra a arquitetura *MTS-PolKA*, onde os fluxos podem ser divididos entre vários caminhos de acordo com os pesos definidos pela fonte. Essa flexibilidade permite que as modificações de caminho sejam realizadas com apenas uma mudança no rótulo do cabeçalho do pacote, facilitando a adaptação em tempo real às condições da rede (SANTOS et al., 2024). Considerando o exemplo exposto na Figura 1c, isso significaria alterar apenas o cabeçalho dos pacotes em um único local onde o tráfego é gerado (estrela azul) para que essa alteração seja refletida em toda a topologia.

Outra característica dessas soluções é que elas distribuem n fluxos por p caminhos usando tabelas com pesos, como mostrado na Figura 2a. Dessa forma, cada fluxo é atribuído a um único caminho, evitando a distribuição de um fluxo entre múltiplos *links*. Essa limitação afeta significativamente os fluxos de elefante, que são caracterizados por grandes transferências de dados que consomem uma quantidade substancial de largura de banda ou que duram um longo período (GILLIARD et al., 2024). O segundo cenário na Figura 1b ilustra uma situação em que os fluxos de ratos (fluxos pequenos e de curta duração, com baixo consumo de recursos) e de elefantes competem pela largura de banda. Mesmo que

Figura 2 – Problema: Como configurar os *switches* para permitir a distribuição ponderada de um fluxo em multicaminhos com granularidade de pacotes?



(a) Trabalhos relacionados distribuem n fluxos por p caminhos usando tabelas com pesos.
(b) Nossso trabalho permite dividir qualquer fluxo em p caminhos usando rótulos com pesos.

Fonte: Elaboração própria

haja largura de banda sobrando em outros caminhos, o *WCMP* é limitado a alocar apenas um caminho por vez para o fluxo de elefante. Outros trabalhos utilizam estratégias de distribuição de pacotes para permitir a divisão de um fluxo entre multicaminhos (DIXIT; PRAKASH et al., 2013; HUANG; LYU et al., 2021), mas não oferecem uma forma de aplicar políticas para tráfego heterogêneo.

A gestão de fluxos de ratos é fundamental, pois esses fluxos se caracterizam por sua curta duração e elevada sensibilidade a atrasos na comunicação (XIE et al., 2024). Por outro lado, os fluxos de elefantes são particularmente comuns em tarefas de processamento de dados em larga escala, como análise de *big data* e treinamento de modelos de aprendizado de máquina. Nesse contexto, a capacidade de alocar recursos de rede de forma ágil e eficiente é crucial para o desempenho dos *datacenters* modernos (GEREMEW; DING, 2023). É interessante notar que, embora os fluxos de elefantes sejam responsáveis por cerca de 90% do volume total de dados, eles representam apenas aproximadamente 2% do total de fluxos (BENSON et al., 2010).

Em resumo, a engenharia de tráfego eficaz em redes de *datacenters* requer um mecanismo de divisão de tráfego que distribua dinamicamente e flexivelmente os fluxos em vários caminhos para lidar com tráfego heterogêneo. Esse mecanismo deve oferecer suporte a políticas por fluxo que aloquem o tráfego proporcionalmente com base na largura de banda disponível em cada caminho e, quando necessário, habilitem a divisão de um único fluxo em vários caminhos.

1.2 PROPOSTA

Este trabalho visa preencher as lacunas existentes nos trabalhos relacionados, buscando uma distribuição de tráfego eficiente em redes de *datacenter* com multicaminhos redundantes. Os métodos tradicionais de roteamento, ineficazes em topologias assimétricas e em ambientes com tráfego dinâmico, não se adaptam às variações de largura de banda e à heterogeneidade da rede, resultando em desbalanceamento e subutilização de recursos. Assim, destaca-se a necessidade de abordagens mais adaptativas para otimizar a distribuição de tráfego e melhorar a performance da rede. Propomos um esquema de engenharia de tráfego que realiza a divisão de fluxos em nível de pacote por meio de dispositivos de rede programáveis, permitindo transmissão rápida de fluxos *TCP* através de vários caminhos. Utilizamos o roteamento na fonte (*Source Routing, SR*) e diferentes perfis de divisão de tráfego, com pesos dinâmicos, para otimizar a utilização da largura de banda e reduzir o número de estados na rede em comparação com abordagens convencionais.

Para atender a essa necessidade, este artigo propõe uma solução que permite configurar como cada *switch* i , em um caminho de rede r , distribui um fluxo f entre suas portas de saída, com granularidade de pacotes, na velocidade exigida pelas redes modernas de data center. Além disso, essa divisão deve obedecer proporções de pesos pré-configuradas para cada subconjunto p de suas portas. A Figura 2b mostra um exemplo ilustrativo, de parte da proposta: na Figura, um *switch* Si, com 5 portas, deve distribuir o tráfego de um fluxo f qualquer, configurado no ingresso da rede, conforme a proporção de pesos 2:1:2:1 para as portas o $p = \{A, B, D \text{ e } E\}$.

A solução a ser apresentada neste trabalho se baseia na arquitetura *M-PolKA (Multipath Polynomial Key-based Architecture)* (GUIMARÃES et al., 2022), e propõe uma abordagem de divisão de tráfego multicaminho denominada *MTS-PolKA (Multipath Traffic Split Polynomial Key-based Architecture)*. O *MTS-PolKA* permite selecionar, na origem, um perfil de divisão de tráfego em proporção de pesos para cada fluxo de rede, por meio de um rótulo no cabeçalho do pacote. Cada *switch* no caminho mantém uma tabela estática com os perfis de divisão de tráfego disponíveis (SANTOS et al., 2024).

A inovação desse trabalho está na capacidade de alterar a distribuição do tráfego, com alta granularidade, simultaneamente em todos os *switches* do caminho, por meio de uma simples modificação de rótulo no cabeçalho dos pacotes de cada fluxo ingressante na rede. Esse método elimina a necessidade de se reconfigurar tabelas ou estruturas de dados em cada *switch*, e permite uma engenharia de tráfego mais ágil, eficiente e dinâmica em redes de *datacenters*.

As principais contribuições deste trabalho *MTS-PolKA* incluem:

- **Extensão da solução de roteamento de fonte multicaminho *M-PolKA*, para permitir a configuração de perfis de divisão de tráfego personalizadas:** O roteamento baseado na arquitetura *M-PolKA* permite que a origem selecione os múltiplos caminhos de um fluxo conforme uma distribuição que usa uma árvore *multicast*, sem reconfigurações complexas nas tabelas de comutação, apenas inserindo rótulos no ingresso dos pacotes da rede. Entretanto, o *M-PolKA* apenas clona os pacotes para todas as portas que pertencem ao caminho selecionado, sem permitir que se configure uma forma de distribuição do tráfego personalizada. Esta lacuna foi preenchida pelo *MTS-PolKA*.
- **Divisão dinâmica de tráfego:** Ao contrário das abordagens tradicionais de divisão de tráfego, que exigem modificações nas tabelas de comutação ou nas estruturas de dados, o *MTS-PolKA* permite que a fonte selecione dinamicamente os perfis de distribuição de tráfego para cada um desses fluxos usando um rótulo de rota e outro rótulo de pesos. Isso permite alterações simultâneas na distribuição de tráfego em todos os *switches* no caminho.
- **Implementação eficiente no plano de dados de *switches* programáveis:** O *MTS-PolKA* utiliza o Sistema Numérico de Resíduos (do inglês, *RNS*, *Residue Number System*) (GUIMARÃES et al., 2022) e tabelas estáticas, previamente configuradas, para definir os perfis de tráfego disponíveis na rede, eliminando a necessidade de se alterar informações de estado em cada *switch* no caminho selecionado, simplificando as operações de engenharia de tráfego e possibilitando uma implementação eficiente em *switches* programáveis.

Como prova de conceito, implementou-se o *MTS-PolKA* na linguagem P4 e avaliou-se a solução proposta usando o emulador *Mininet*. Diversos experimentos foram conduzidos para demonstrar o funcionamento do *MTS-PolKA*. Os resultados evidenciam a expressividade da nossa proposta para explorar multicaminhos de rede, mantendo a estabilidade do fluxo e possibilitando reconfigurações ágeis de perfis de divisão de tráfego na origem, com potencial de melhorar o desempenho e a eficiência em redes de *datacenters*.

Além disso, este trabalho complementa e expande os conceitos abordados no artigo *MTS-PolKA: Divisão de Tráfego Multicaminhos em Proporção de Peso com Roteamento na Fonte*, apresentado na Sessão Técnica (ST4) *Routing and Mobility* do 42º Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos (SBRC)¹, realizado em Niterói/RJ, em 21/05/2024, e publicado nos Anais do XLII Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos da Sociedade Brasileira de Computação² em 20/05/2024 (p.71-84).

¹ Disponível em: <<https://sbrc.sbc.org.br/2024/st.html>>

² Disponível em: <<https://sol.sbc.org.br/index.php/sbrc/article/view/29784/29587>>

1.3 OBJETIVOS

1.3.1 Objetivo Geral

O objetivo geral da pesquisa é propor e avaliar um mecanismo de divisão de tráfego em redes de *datacenter* com múltiplos caminhos redundantes, baseado em roteamento na fonte, com suporte a políticas por fluxo com proporção de pesos e granularidade em nível de pacote.

1.3.2 Objetivos Específicos

1. Estudar o estado da arte e analisar os requisitos para realizar divisão de tráfego em redes de *datacenter*.
2. Levantar mecanismos para utilizar roteamento na fonte em redes com múltiplos caminhos.
3. Projetar uma solução para realizar a divisão de tráfego de redes multicaminhos com roteamento na fonte de acordo com os requisitos definidos.
4. Desenvolver um protótipo de prova de conceito da solução proposta.
5. Avaliar o desempenho do protótipo de prova de conceito da solução proposta.
6. Comparar a solução proposta com trabalhos relacionados.

1.4 ORGANIZAÇÃO DO TRABALHO

O restante deste trabalho está organizado da seguinte forma. A Capítulo 2 apresenta o referencial teórico. O Capítulo 3 apresenta alguns trabalhos relacionados, evidenciando as contribuições deste trabalho. A Capítulo 4 descreve a proposta do projeto e a implementação do *MTS-PolKA*. No capítulo 5, é apresentada a prova de conceito da arquitetura proposta, bem como as tecnologias envolvidas. A Capítulo 6 apresenta e discute os resultados dos experimentos. Por fim, a Capítulo 7, são feitas as considerações finais sobre esta dissertação e os trabalhos futuros.

2 REFERENCIAL TEÓRICO

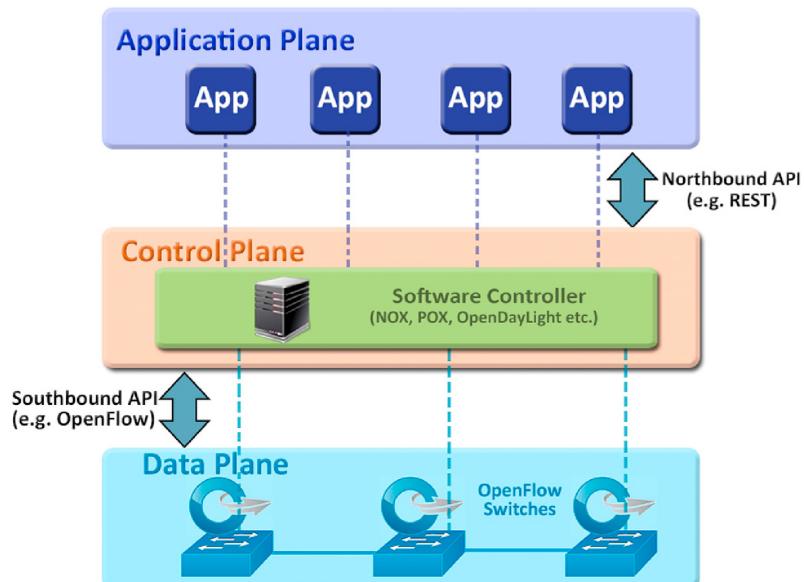
Este capítulo faz uma revisão bibliográfica sobre os principais conceitos relacionados ao problema de pesquisa.

2.1 ARQUITETURA DE REDE DEFINIDA POR SOFTWARE

A rede definida por software (*SDN*) surgiu como um novo e promissor paradigma, mudando da rede tradicional para a Internet do futuro para oferecer programabilidade e gerenciamento mais fácil (YU et al., 2016). Conforme mostrado na Figura 3 a arquitetura *SDN* está associada a três componentes principais: plano de controle que consiste em um ou vários controladores; plano de dados que é caracterizado pela presença de equipamentos de rede (e.g., roteadores, *switches*, bem como *middleboxes*) que interagem para formar uma rede de encaminhamento de dados; e a camada de aplicações *SDN* (HAMDAN et al., 2021).

Outra característica é a programação da rede através do uso de interfaces de comunicação abertas, que permitiu a criação e gerenciamento de funcionalidades customizadas, (e.g., detecção e reação a eventos de rede significativos). Tal evolução possibilitou funcionalidades como gerenciamento de topologia ou mecanismos de controle de tráfego, que aumentaram a eficiência do desempenho e a capacidade de programação da infraestrutura de rede (COSTA et al., 2020).

Figura 3 – Arquitetura SDN



Fonte: Hamdan et al. (2021).

Nas *SDN*, com a possibilidade de separar o plano de controle do plano de dados, surgiram novas funções de gerenciamento de rede. Estas funções de rede são implementadas

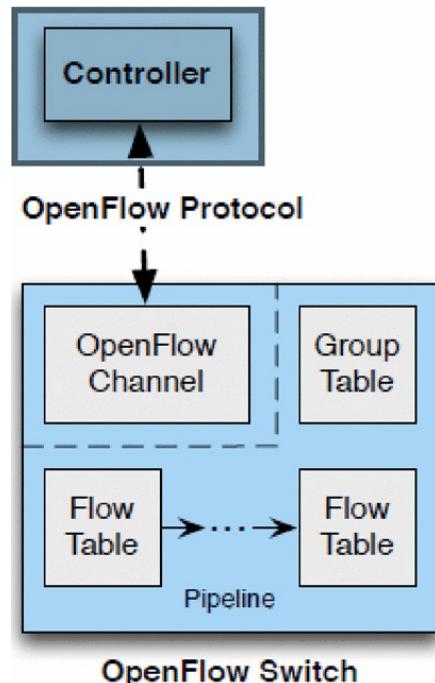
removendo decisões de controle, como roteamento, de dispositivos de *hardware* e permitindo tabelas de fluxo programáveis no hardware usando um protocolo padronizado.

As redes *SDN* podem ser divididas em duas gerações. A primeira foca na separação entre o plano de controle e o plano de dados, como no *OpenFlow*, permitindo uma gestão centralizada, mas com limitações em flexibilidade e personalização. A segunda geração, por sua vez, traz maior adaptabilidade com o uso de linguagens como P4, que permitem a programação dinâmica de protocolos e comportamentos de rede, tornando as redes mais responsivas e eficientes.

2.1.1 Primeira Geração SDN: Protocolo OpenFlow

O *OpenFlow* é um protocolo de controle amplamente reconhecido utilizado entre o dispositivo de controle e cada dispositivo de rede em uma arquitetura de rede definida por software (*SDN*). Nesse contexto, o dispositivo de controle é denominado controlador *OpenFlow*, enquanto o dispositivo de rede é referido como *switch OpenFlow*. O controlador *OpenFlow* estabelece regras de encaminhamento em tabelas de fluxo localizadas nos *switches OpenFlow* por meio do canal *OpenFlow*. Dessa forma, o *OpenFlow* encaminha pacotes conforme as regras definidas nessas tabelas de fluxo. A Figura 4 mostra os principais componentes de um *switch OpenFlow*.

Figura 4 – Principais componentes de um *switch Openflow*.



Fonte: Gopakumar, Unni e Dhipin (2015).

As implementações da primeira geração de redes definidas por software (*SDN*), como

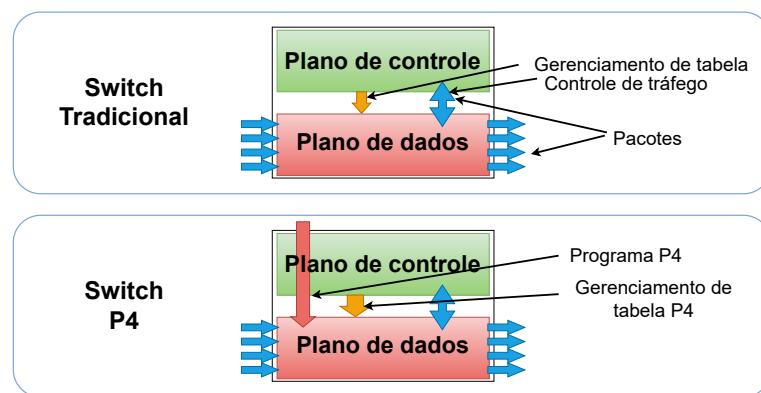
o *OpenFlow*, encontram dificuldades em lidar com procedimentos de encaminhamento baseados no estado da rede, variações de perfil e histórico de estatísticas de fluxo em nível de nó. Assim, é exigida a intervenção do controlador *SDN*, resultando em problemas de escalabilidade, latência adicional e preocupações com a segurança dos nós, afetando os recursos de TI.

O protocolo *OpenFlow* revolucionou o gerenciamento de redes ao permitir a programação direta dos *switches*, mas apresenta limitações em termos de flexibilidade e extensibilidade. Uma das principais lacunas é a capacidade de definir e customizar o comportamento do plano de dados das redes de forma dinâmica. Dessa forma, não é possível que os desenvolvedores especifiquem novos protocolos de rede, estruturas de dados e ações personalizadas, dificultando a implementação de inovações.

2.1.2 Segunda Geração SDN: Linguagem P4

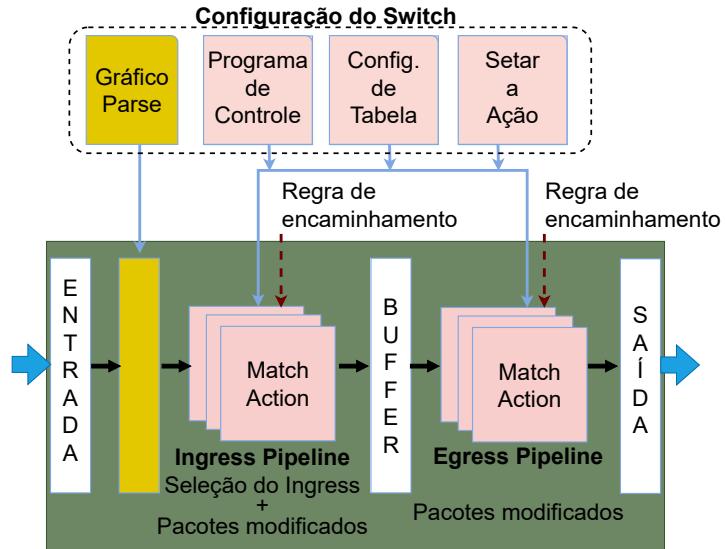
Para enfrentar os desafios da geração anterior de *SDN*, foi introduzida a linguagem de programação aberta P4 (*Protocol Independent Packet Processors*). Essa linguagem permite a programação do plano de dados de *switches SDN*, oferecendo uma solução mais flexível e eficiente para *network management* (PAOLUCCI et al., 2019). A linguagem P4 foi desenvolvida para especificar o processamento de pacotes em dispositivos de SDN, como *switches* e roteadores. Isso possibilita a programação da funcionalidade do plano de dados de forma flexível e eficiente. Como exemplo concreto, a Figura 5 ilustra a diferença entre um *switch* de função fixa tradicional e um *switch* programável P4. A principal distinção reside no fato de que, em um *switch* Programáveis P4, a funcionalidade do plano de dados é programável e dinâmica, sendo definida por um programa P4 durante a inicialização. Além disso, o compilador P4 gera uma *API* (*Application Programming Interface*), que facilita a comunicação entre os diferentes planos, permitindo uma integração mais fluida e flexível nas operações de rede (P4.org, 2021).

Figura 5 – Switches Tradicionais e Switches Programáveis (P4).



Fonte: Adaptado de (P4.org, 2021).

Figura 6 – Arquitetura PISA. Adaptado de (BOSSHART et al., 2014)

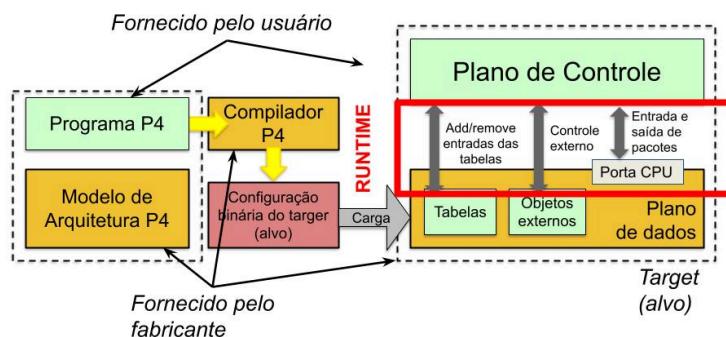


Fonte: Adaptado de (P4.org, 2021).

Com o uso da linguagem *P4*, é possível abstrair o plano de dados, simplificando a programação do comportamento dos equipamentos de rede. A linguagem é baseada na arquitetura de *switch* programável, conhecida como *PISA* (*Protocol Independent Switch Architecture*), que utiliza chips capazes de operar a velocidades de até terabits por segundo (Tb/s).

Por meio da arquitetura *PISA*, surge um novo paradigma de hardware, baseado em um pipeline de (re)configuração que utiliza o modelo *Match+Action*. Diferentemente dos *ASICs* tradicionais, essa arquitetura oferece grande flexibilidade sem comprometer o desempenho. A Figura 6 ilustra de forma genérica o funcionamento de um equipamento que adota a arquitetura *PISA* (BOSSHART et al., 2014).

Figura 7 – Fluxo de implementação do programa P4. Adaptado de (P4.org, 2021)



Fonte: Adaptado de (P4.org, 2021).

A Figura 7 apresenta um fluxo de trabalho típico para programar um dispositivo P4. Nela, os fabricantes disponibilizam modelos de implementação, sejam de hardware ou

software, juntamente com uma definição de arquitetura e um compilador P4 específico para o destino. Este compilador mapeia o código-fonte P4 para o modelo de destino, gerando uma configuração do plano de dados que implementa a lógica de encaminhamento, além de produzir uma *API* que permite ao plano de controle gerenciar o estado dos objetos do plano de dados (P4.org, 2021).

Este trabalho utilizou a linguagem de programação *P4* para descrever o comportamento de *switches*, construindo sua interface com base em um conjunto de recursos comum a diversas arquiteturas existentes. Essa abordagem garante a flexibilidade necessária para a implantação, além de proporcionar um alto nível de abstração, tornando a linguagem compatível com projetos atuais (CASTANHEIRA; PARIZOTTO; SCHAEFFER-FILHO, 2019).

Para implementação do protótipo, utilizamos a ferramenta *BMv2 (Behavioral Model version 2)*, uma arquitetura de *switch* virtual desenvolvida para a pesquisa em redes de computadores. Para a execução dos experimentos, que envolvem funcionalidades especiais como a compilação de programas P4 e a configuração dos *switches*, utilizou-se o emulador *Mininet*(DOMINICINI et al., 2020b).

2.2 ROTEAMENTO NA FONTE

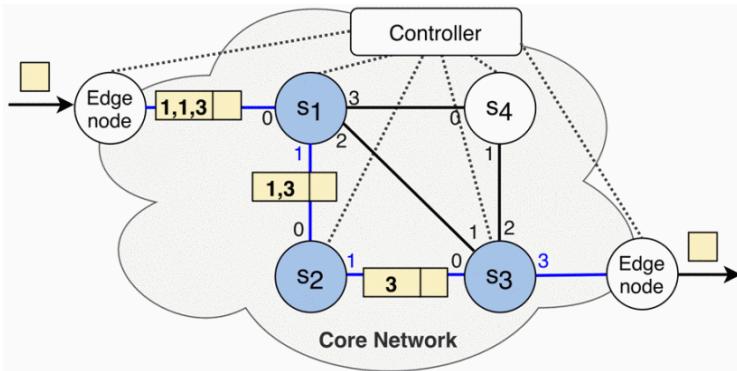
Neste contexto, os mecanismos de roteamento na fonte (do inglês *Source Routing - SR*) têm ganhado destaque, pois reduzem o tamanho das tabelas de encaminhamento e diminuem a sobrecarga do plano de controle em comparação com abordagens tradicionais. Mais especificamente, o roteamento rígido na fonte (do inglês *Strict Source Routing - SSR*) permite que, durante a classificação de um fluxo, informações sejam inseridas nos pacotes, especificando todo o caminho a ser percorrido na rede. Isso elimina a necessidade de consultas a todas as tabelas de encaminhamento nos dispositivos de comutação ao longo do caminho. As soluções que utilizam o roteamento na fonte exigem um número menor de regras instaladas, resultando em um maior grau de escalabilidade(VALENTIM et al., 2019a)

Um dos principais problemas em redes é como selecionar caminhos de roteamento no núcleo da rede e balancear a carga entre eles para se adaptar aos padrões de tráfego altamente variáveis. Uma abordagem comum para esse desafio é codificar vários caminhos em nós de núcleo na forma de entradas de tabela de encaminhamento e, em seguida, permitir que os nós da borda selecione entre os caminhos existentes. No entanto, muitas das soluções existentes requerem um grande número de entradas de tabela, são limitados pela capacidade das tabelas de encaminhamento de *switch* e ainda restringem a seleção de caminho (DOMINICINI et al., 2020a).

2.2.1 List-based Source Routing (LSR)

A forma tradicional de executar o *Source Routing* (SR) é a abordagem baseada em lista (do inglês *List-based Source Routing*, LB-SR). No *LB-SR*, a identificação da rota, ou *routeID*, é representada por uma lista de portas ou endereços. A operação de encaminhamento é realizada por meio de um comando de pop, que reescreve o rótulo da rota e atualiza sua posição na lista (DOMINICINI et al., 2021).

Figura 8 – Exemplo de roteamento na fonte sem uso de tabelas.



Fonte: (DOMINICINI et al., 2021)

Quando um pacote chega a um nó da borda, este se comunica com um controlador, que calcula um rótulo de rota. Esse rótulo representa uma lista ordenada de portas, que orienta o trajeto do pacote pela rede. Por exemplo, na Figura 8, a lista 1, 1, 0 mapeia um caminho passando pelos nós S1, S2 e S3, e define suas respectivas portas de saída. Cada nó do núcleo da rede lê o primeiro elemento dessa lista e o remove (pop), atualizando o rótulo de rota do pacote à medida que este avança pela rede (DOMINICINI et al., 2021).

No entanto, o *LSR* ainda requer a manutenção de um estado na lista de roteamento em cada pacote, o que pode ter impacto no desempenho e na escalabilidade da rede, e não é capaz de representar uma árvore para uma topologia genérica (GUIMARÃES et al., 2022).

2.2.2 KeyFlow

A proposta do *KeyFlow* (MARTINELLO et al., 2014), visa construir elementos de rede de núcleo mais simples para o encaminhamento de pacotes. Com a proposta baseada em propriedades especiais de um sistema numérico de resíduos (*RNS*). No esquema de *SR* baseado em *RNS* inteiro, o controlador calcula o rótulo da rota *routeID* como um número inteiro, que permanece o mesmo ao longo do caminho.

Por exemplo, aplicando o esquema *RNS* ao cenário descrito na Figura 8, podemos atribuir os *nodeID* 4, 3, 5, 7 aos nós S1, S2, S3 e S4, respectivamente. O *routeID* calculado com base nesse sistema seria 25. Dessa forma, as portas de saída para os nós S1, S2 e S3 seriam determinadas pelos cálculos modulares:

- $25 \bmod 4 = 1$ (porta de saída para S1);
- $25 \bmod 3 = 1$ (porta de saída para S2);
- $25 \bmod 5 = 0$ (porta de saída para S3).

Essa abordagem torna o encaminhamento mais eficiente, já que o *routeID* permanece fixo, enquanto os nós centrais utilizam operações modulares para determinar as portas de saída (MARTINELLO et al., 2014).

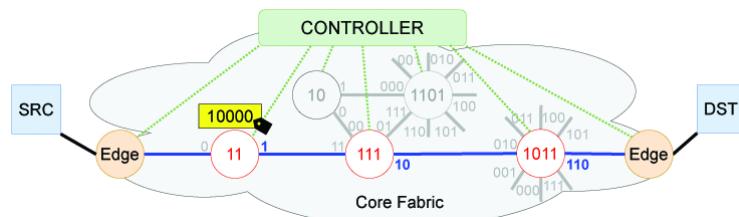
No entanto, o *KeyFlow* baseia-se na aritmética *RNS* inteira, e a operação do *mod* inteiro não pode ser implementada no *hardware* de rede de comercial atual. Portanto, usa implementações de *switches* de *software*, ou dependem da síntese de divisão de inteiros para *ASICs* ou *NetFPGAs* (DOMINICINI et al., 2021).

2.2.3 PolKA

Diferentemente do *KeyFlow*, o *PolKA* (*Polynomial Key-based Architecture*) (DOMINICINI et al., 2021) é um modelo de arquitetura que utiliza chaves polinomiais para gerir e processar dados de forma eficiente. Essa abordagem propõe uma implementação que amplia a expressividade do *Source Routing (SR)*, aproximando-se de operações polinomiais binárias elementares. Além disso, introduz uma nova interpretação sobre a aritmética clássica dos inteiros em um *RNS* (*Residue Number System*), onde o número da porta de saída é determinado pelo resultado da divisão polinomial binária do identificador de rota do pacote pelo identificador do nó.

O *PolKA* explora o sistema de números de resíduo polinomial (*RNS*) para implementar um *Source Routing (SR)* totalmente sem estado, ou seja, não é necessário transmitir atualizações de estado nos cabeçalhos dos pacotes ao longo da rota. Este esquema de SR é fundamentado no *RNS* e utiliza o teorema do resto chinês (*CRT*) aplicado a polinômios.

Figura 9 – Exemplo do roteamento *PolKA* original.



Fonte: (DOMINICINI et al., 2021)

Conforme mostrado na Figura 9, a arquitetura é composta por: (i) nós da borda, (ii) nós do núcleo e (iii) um Controlador *SDN*, responsável pela configuração dos nós. O *SR* depende de três polinômios sobre GF (2): (i) *nodeID* : um identificador fixo atribuído

aos nós centrais pelo Controlador em uma fase de configuração da rede; (ii) *portID*: um identificador atribuído às portas de saída de cada nó central; e (iii) *routeID*: um identificador de rota, calculado pelo Controlador e embutido nos pacotes pelos nós de borda (DOMINICINI et al., 2021).

A Figura 9, ilustra o funcionamento do *PolKA SR* em um caminho com três nós, cada um recebendo um identificador (*nodeID*) do Controlador (DOMINICINI et al., 2021):

$$\begin{aligned}s_1(t) &= t + 1 \quad (11), \\ s_2(t) &= t^2 + t + 1 \quad (111), \\ s_3(t) &= t^3 + t + 1 \quad (1011).\end{aligned}$$

Os identificadores de porta (*portIDs*) correspondentes são:

$$\begin{aligned}o_1(t) &= 1 \quad (01), \\ o_2(t) &= t \quad (10), \\ o_3(t) &= t^2 + t \quad (110).\end{aligned}$$

O Controlador calcula um identificador de rota (*routeID*) $R(t) = t^4 \quad (10000)$. Cada nó utiliza uma operação de módulo para calcular seu *portID*; por exemplo, no nó s_3 , o *portID* é obtido como $o_3(t) = \langle 10000 \rangle_{1011} = 110$.

A limitação do hardware de rede convencional e da linguagem P4 reside na falta de suporte para operações de módulo polinomial ou inteiro com operandos não constantes. Para superar essa restrição, foi criada uma abordagem que possibilita a execução do módulo polinomial em hardware, aproveitando operações de Códigos de Redundância Cíclica (CRC) que já estão disponíveis (DOMINICINI et al., 2021), da seguinte forma:

Cálculo do Identificador da Porta (*portID*)

1. $G = \text{nodeID}, \quad r = \text{degree}(G)$
2. $D = \text{routeID} \div 2^r \quad (\text{SHIFT RIGHT})$
3. $\text{dif} = \text{routeID} - D * 2^r \quad (\text{SHIFT LEFT, XOR})$
4. $R = \langle D * 2^r \rangle_G \quad (\text{CRC})$
5. $\text{portID} = \text{dif} + R \quad (\text{XOR})$

Portanto, o *switch* calcula a porta de saída usando duas operações **SHIFT**, uma **CRC** e duas **XOR**, o que é mais eficiente computacionalmente do que executar uma divisão (DOMINICINI et al., 2021).

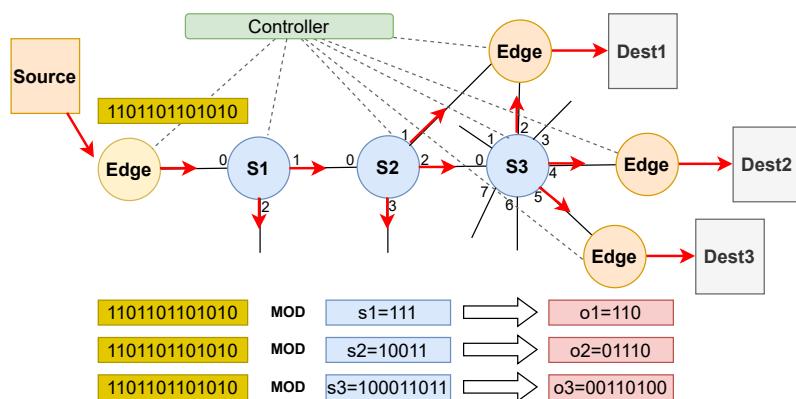
A técnica mencionada permite a implementação do *PolKA* em dispositivos que utilizam a linguagem *P4*, desde que esses dispositivos suportem a configuração de polinômios geradores. O *P4* permite operações de Códigos de Redundância Cíclica (CRC) através de bibliotecas externas, e a utilização de polinômios personalizados depende das arquiteturas dos dispositivos. As arquiteturas *PSA* (*Programming Software Architecture*) e *v1model* suportam polinômios de 16 e 32 bits. Dispositivos como o *switch* de software *bmv2* e o *switch* de hardware *Tofino* da *Barefoot* podem usar polinômios CRC personalizados, enquanto os *SmartNICs* da *Netronome* só aceitam polinômios fixos, limitando sua flexibilidade na implementação do *PolKA*. Em resumo, a implementação do *PolKA* em dispositivos *P4* depende da capacidade de lidar com polinômios CRC personalizados (DOMINICINI et al., 2021).

Posteriormente, a arquitetura *PolKA*, que propõe um esquema de *SR* para caminhos simples *unicast*, foi estendida para a aplicação de roteamento na fonte multicaminhos, com a arquitetura *M-PolKA*, descrita a seguir.

2.2.4 M-PolKA

O protocolo *M-PolKA* (Multipath Polynomial Key-based Architecture) é um roteamento de fonte multicaminho baseado em *RNS*, onde as decisões de encaminhamento no núcleo dependem de operações de módulo polinomial sobre um rótulo de rota. Nele é possível codificar um rótulo de rota para um ciclo de multicaminho ou árvore de maneira agnóstica à topologia (GUIMARÃES et al., 2022).

Figura 10 – Exemplo do roteamento M-PolKA original.



Fonte: Adaptado de (GUIMARÃES et al., 2022)

Como mostrado na Figura 10, a arquitetura do *M-PolKA* envolve *switches* do núcleo (*S_i*), da borda (*Edge*), e um controlador centralizado (*Controller*), permitindo o roteamento multicaminhos por meio de 3 (três) identificadores polinomiais com representação binária: *routeID*, *nodeID* e *portID*. Essa abordagem viabiliza o roteamento de fonte sem armazenamento, utilizando operações de módulo entre *routeID* e *nodeID* para decisões de encaminhamento e obtenção do *portID* correspondente, e possibilita o roteamento sem

armazenamento de estado, baseando-se em simples operações de módulo em rótulos de rota para definir as portas de saída (GUIMARÃES et al., 2022).

O estado de transmissão das portas de saída (*portID*) é fornecido pelo resto da divisão polinomial binária (isto é, uma operação de módulo mod) entre o identificador de rota (*routeID*) e o identificador do nó (*nodeID*). Sua implementação em *switches* programáveis conta com um mecanismo de clonagem de pacotes e a reutilização do suporte de *hardware CRC* (*Cyclic Redundancy Redundancy Check*), que permite a operação do *mod* (KOPONEN et al., 2010).

Figura 11 – Cabeçalho M-PolKA com comprimento fixo

Ethernet	routeID	IP	data
----------	---------	----	------

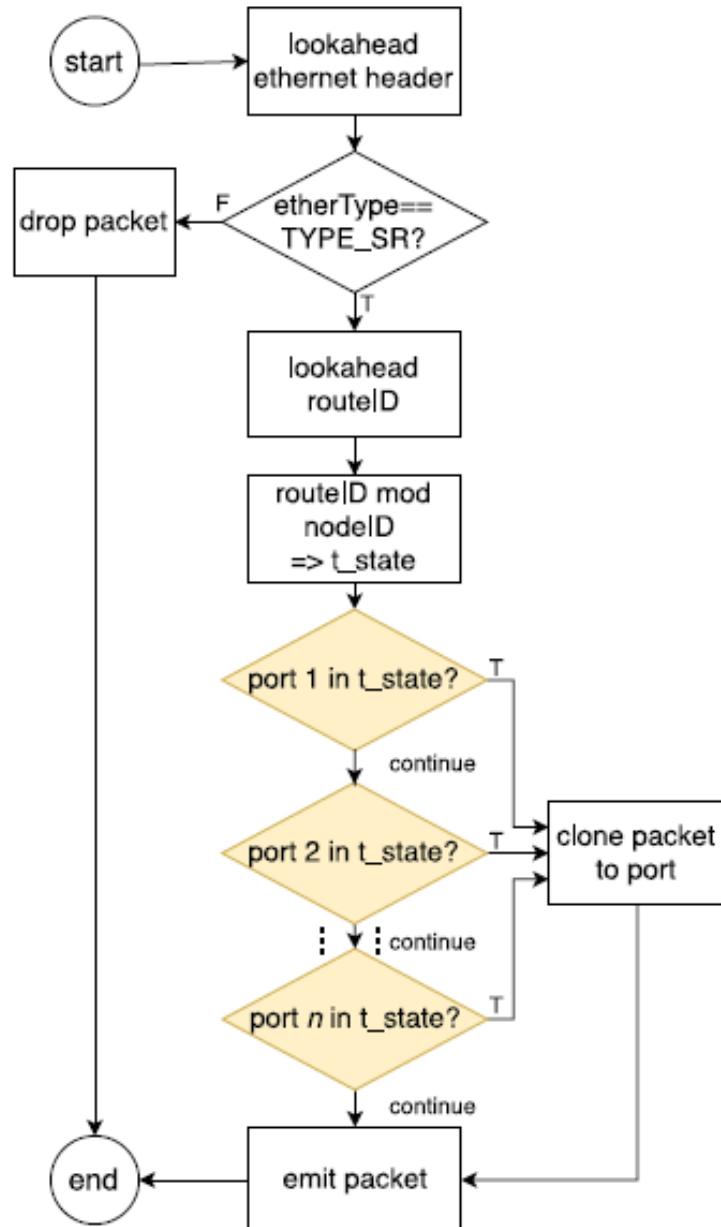
Fonte: (GUIMARÃES et al., 2022)

No exemplo da Figura 10, o controlador instala regras de fluxo no nó da borda para incorporar o identificador de rota, $routeID = 1101101101010$ nos pacotes de um fluxo. O resultado das operações de módulo nos *switches* do núcleo está representado pelas setas em vermelho. Por exemplo, o resto de $R(t) = 1101101101010$ quando dividido por $s_3(t) = 100011011$ é $o_3(t) = 00110100$. Assim, em s_3 , as portas 2, 4 e 5 encaminham esse fluxo para o próximo salto, enquanto as outras portas não o encaminham.

O cabeçalho *M-PolKA* contém um *routeID*, cujo comprimento máximo depende da topologia da rede (GUIMARÃES et al., 2022). A Figura 11 mostra o formato do cabeçalho *M-PolKA* com um campo de comprimento fixo para armazenar o *routeID*, após o cabeçalho *Ethernet*.

Na Figura 12, é explicado o funcionamento do *pipeline* do protocolo *M-PolKA* em *switches* do núcleo. Quando o cabeçalho *Ethernet* indica que o tipo de pacote é *TYPE_SR*, o *M-PolKA* precisa apenas de acesso de leitura aos cabeçalhos dos pacotes. Para isso, ele utiliza o método de *lookahead* da linguagem P4, que avalia um conjunto de bits do pacote de entrada sem avançar o ponteiro do índice do pacote. Para descobrir o estado de transmissão (*t_state*), o *switch* executa uma operação de módulo entre o *routeID* no pacote e o próprio *nodeID* do *switch*. Para iterar sobre o estado de transmissão, é utilizada uma cadeia de declarações condicionais *if* para cada porta. Por exemplo, para cada porta do *switch*, o código verifica uma condição baseada no valor de (*t_state*). Se a condição for verdadeira, o pacote é encaminhado por aquela porta. Se for falsa, o próximo "if" é avaliado para outra porta, e assim por diante, até que a porta correta seja encontrada. Essa abordagem evita *loops* complexos ou iterações extras, utilizando condições simples para verificar diretamente o estado de transmissão para cada porta, garantindo um processo de encaminhamento eficiente. Dessa forma, o uso de cadeias de "if" permite que o *switch* decida o encaminhamento de forma rápida, sem precisar de mecanismos mais pesados, como

Figura 12 – Pipeline do Núcleo P4 para M-PolKA



Fonte: (GUIMARÃES et al., 2022)

resubmit ou recirculate, que adicionariam latência ao processo. (DOMINICINI et al., 2021).

Quando o pacote chega a um *switch* da borda, o cabeçalho *SR* é removido e o *etherType* é alterado para *TYPE_IPv4*.

3 TRABALHOS RELACIONADOS

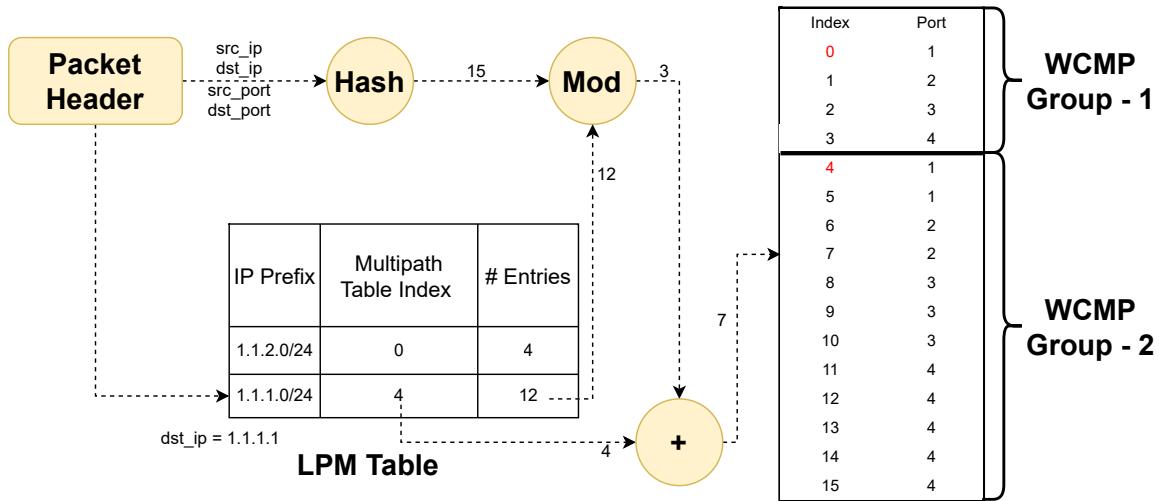
A divisão de tráfego é um recurso essencial em redes modernas para balancear o tráfego em múltiplos caminhos ou servidores de rede (ROTTENSTREICH; KANIZO et al., 2018). O problema de divisão de tráfego pode ser categorizado em quatro tipos com base na granularidade: fluxo, pacote, *flowlet* e granularidade adaptativa (LI et al., 2021). Abordagens baseadas em fluxo evitam a reordenação de pacotes, mas enfrentam atrasos de cauda longa e baixa utilização da link. Esquemas baseados em pacotes, embora melhorem a utilização do link, podem apresentar problemas como colisões de *hash* e reordenação de pacotes, o que pode degradar o desempenho da rede. Esquemas baseados em *flowlet* gerenciam picos de pacotes com intervalos de tempo para reduzir a reordenação de pacotes. No entanto, eles ainda podem sofrer com baixa utilização devido à sua granularidade de comutação relativamente grosseira. Nesta seção, comparamos vários trabalhos que estão mais relacionados ao *MTS-PolKA*.

O método mais popular para平衡ar os fluxos em diferentes caminhos é o ECMP, que usa *hashing* para distribuir fluxos igualmente entre caminhos de igual comprimento, apesar de ter pouca aplicação em topologias assimétricas (VALENTIM et al., 2019b). No entanto, este esquema apresenta pontos fracos bem conhecidos: colisões na tabela *hash* podem resultar em fluxos utilizando o mesmo caminho, causando desbalanceamento. Além disso, o ECMP é ineficaz em topologias de rede assimétricas (VALENTIM et al., 2019b).

O tamanho do *hash* e a chance de colisão dependem da função de *hash* utilizada e do número de fluxos. Em geral, funções de *hash* bem projetadas têm um espaço de saída suficientemente grande para minimizar a probabilidade de colisões. No entanto, em redes de data center com um grande número de fluxos, a chance de colisões pode aumentar, especialmente se o número de caminhos disponíveis for limitado (WEI et al., 2015).

O WCMP, por sua vez, generaliza o ECMP para permitir distribuições de carga não uniformes, conforme exemplificado na Figura 13. O WCMP utiliza uma estrutura de dados baseada em *hashing* que replica as entradas da tabela de caminhos, permitindo a definição de uma proporção de pesos. No entanto, atualizar essa estrutura requer a modificação dos IDs de caminho em múltiplas tabelas, o que atualmente é inviável devido ao número limitado de acessos à memória por pacote em cada estágio do pipeline de um dispositivo programável (ROTTENSTREICH; KANIZO et al., 2018). Dessa forma, enquanto o ECMP utiliza um caminho para fluxos, incorrendo em desbalanceamento em caso de colisões na tabela *hash*, o WCMP realiza a divisão proporcional de pesos diferentes por caminhos, mas ainda enfrenta desafios devido à limitação de acessos à memória.

Figura 13 – Encaminhamento multicaminho - WCMP



Fonte: Adaptado de Zhou et al. (2014).

O artigo DASH (*Adaptive Weighted Traffic Splitting in Programmable Data Planes*) propõe resolver os problemas do WCMP com um esquema de estruturas de dados para divisão de tráfego com reconhecimento de carga no plano de dados. Essas novas estruturas de dados simultaneamente, espalham novos fluxos por vários caminhos em proporção à carga atual nesses caminhos e se ajustam dinamicamente às mudanças de carga nas velocidades do plano de dados (HSU; TAMMANA et al., 2020). No entanto, o DASH é limitado pelo número de caminhos suportados (ROBIN; KHAN, 2022) e depende de mudanças de estado em dispositivos de rede.

Conforme descrito em Hsu, Tammana et al. (2020), abordagens como CONGA, HULA e CONTRA se concentram no balanceamento de carga diretamente dentro do plano de dados, atualizando dinamicamente o estado do plano de dados sem a necessidade de intervenção do plano de controle. Apesar dos avanços, esses métodos têm limitações, pois normalmente consideram apenas um melhor caminho por vez, o que pode levar à subutilização de outros caminhos potenciais. Por outro lado, o DASH melhora isso ao distribuir fluxos entre vários caminhos de forma proporcional à carga atual, adaptando-se dinamicamente a mudanças (HSU; TAMMANA et al., 2020).

Uma solução proposta, chamada *MP-TCP*, propõe a divisão do tráfego em subfluxos. O *MPTCP*, por sua vez, divide um fluxo *TCP* em vários subfluxos nos *hosts* finais, roteando-os por diferentes caminhos na rede usando *ECMP*. No receptor final, os subfluxos *TCP* são agregados e os pacotes reordenados (PANG; XU; FU, 2017). O desafio do *MP-TCP* é a necessidade de alterações no *host* final, podendo não ser viável em todos os ambientes, e a alta sobrecarga para fluxos curtos nos data centers (DIXIT; PRAKASH et al., 2013). Uma alternativa discutida na literatura é a integração do *MP-TCP* (*MultiPath TCP*) com o *Source Routing (SR)* (PANG; XU; FU, 2017), que visa otimizar o gerenciamento de tráfego,

Tabela 1 – Trabalhos Relacionados de Divisão de Tráfego em Redes de Datacenter

Método	Granularidade			Método de Roteamento	Divisão dos Fluxos		Desvantagens
	Núcleo sem estado	Fluxo	Subfluxo		Pacote	Cam. Rede	Cam. Transp.
ECMP	X	✓		Tabela Hash	✓		Colisões na tabela hash.
WCMP	X	✓		Tabela Hash	✓		Modificações em tabelas.
HULA	X		✓	Tabela Hash	✓		Único melhor caminho.
DASH	X		✓	Hash c/ registradores	✓		No. de caminhos suportados
MP-TCP + SR	✓		✓	SR		✓	Alterações no host final.
RPS	X			Random	✓		Pacotes fora de ordem.
QDAPS	X			Tabela Hash	✓		Assimetria de topologia.
MTS-Polka	✓	✓		SR+Hash Table	✓		Reordenação de Pacotes e Sobre carga de Cabeçalho.

Fonte: Elaboração própria

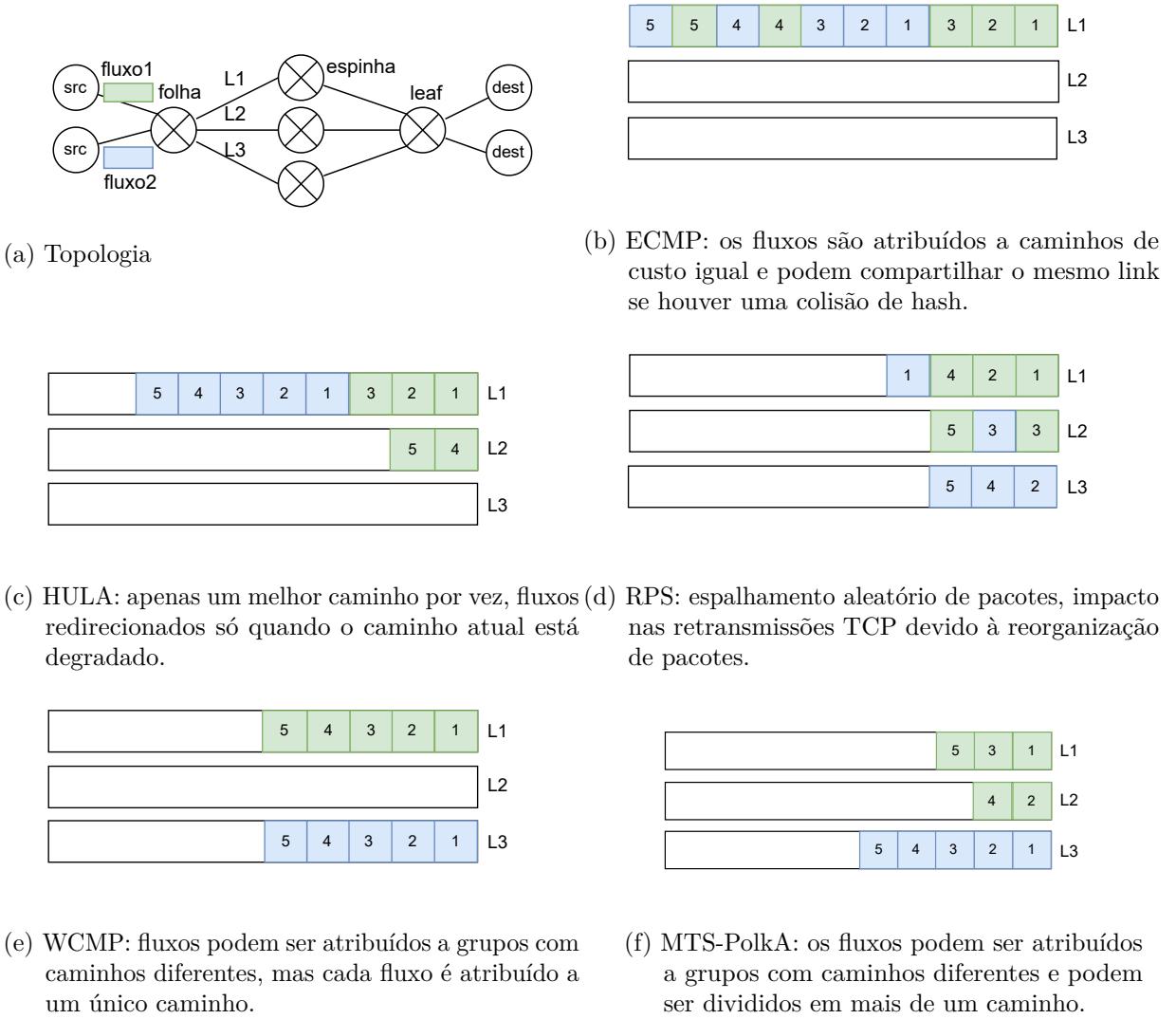
reduzindo a demanda por recursos de memória no plano de dados e o tamanho das tabelas de fluxo nos dispositivos de rede. Essa colaboração melhora o desempenho da rede, mantendo uma alta vazão do *MPTCP*, reduzindo o tempo de conclusão do fluxo e otimizando o uso de recursos de link. Entretanto, a implementação do *MP-TCP + SR* exige configurações específicas nos *hosts* finais (PANG; XU; FU, 2017), podendo ter sua aplicabilidade limitada em ambientes que não permitem grandes mudanças na pilha de protocolos dos *hosts*, como nas plataformas de nuvem pública. Ademais, a complexidade da sinalização e estabelecimento de conexão pode ser um desafio, especialmente para os fluxos curtos, comumente encontrados em ambientes de data center (DIXIT; PRAKASH et al., 2013).

Outras metodologias abordam a questão da divisão de tráfego utilizando estratégias de pulverização de pacotes, mas enfrentam o problema da reordenação de pacotes que pertencem a um fluxo - um problema conhecido por interagir negativamente com o controle de congestionamento *TCP*. Por exemplo, o *RPS* (*Random Packet Spraying*) (DIXIT; PRAKASH et al., 2013) fornece um esquema intuitivo e simples, no qual os pacotes de cada fluxo são atribuídos aleatoriamente a um dos caminhos mais curtos disponíveis para o destino. Já o *QDAPS* (*Queuing Delay Aware Packet Spraying*) (HUANG; LYU et al., 2021) seleciona caminhos para pacotes de acordo com o atraso de enfileiramento e permite que o pacote que chega mais cedo seja encaminhado antes dos pacotes posteriores para evitar a reordenação de pacotes. Embora o *QDAPS* consiga mitigar efetivamente a reordenação de pacotes, a técnica continua sensível à assimetria da topologia.

A Tabela 1 sumariza a comparação das diferentes estratégias de divisão de tráfego multi-caminhos em redes de *datacenter*, considerando como parâmetros: granularidade, presença de um núcleo sem estado, método de roteamento, localização da distribuição do fluxo e desvantagens associadas a cada proposta. A presença de um núcleo sem estado indica alta agilidade de (re)configuração de caminhos e pesos, na velocidade demandada pelas aplicações suportadas pelos planos de dados atuais, sendo essa a principal característica distintiva do *MTS-PolKA*.

As abordagens para gerenciar o fluxo de dados apresentam diferentes *trade-offs*, equilibrando desempenho e eficiência. Esquemas de nível de fluxo tratam do tráfego de forma geral, enquanto o nível de *flowlets* foca em subdivisões menores para operações específicas, mas

Figura 14 – Trabalhos relacionados: exemplo ilustrando as filas de links para um cenário com fluxos de tamanhos comparáveis e links simétricos (Adaptado de (HUANG; LYU et al., 2021)).



(e) WCMP: fluxos podem ser atribuídos a grupos com caminhos diferentes, mas cada fluxo é atribuído a um único caminho.

(d) RPS: espalhamento aleatório de pacotes, impacto nas retransmissões TCP devido à reorganização de pacotes.

Fonte: Elaboração própria

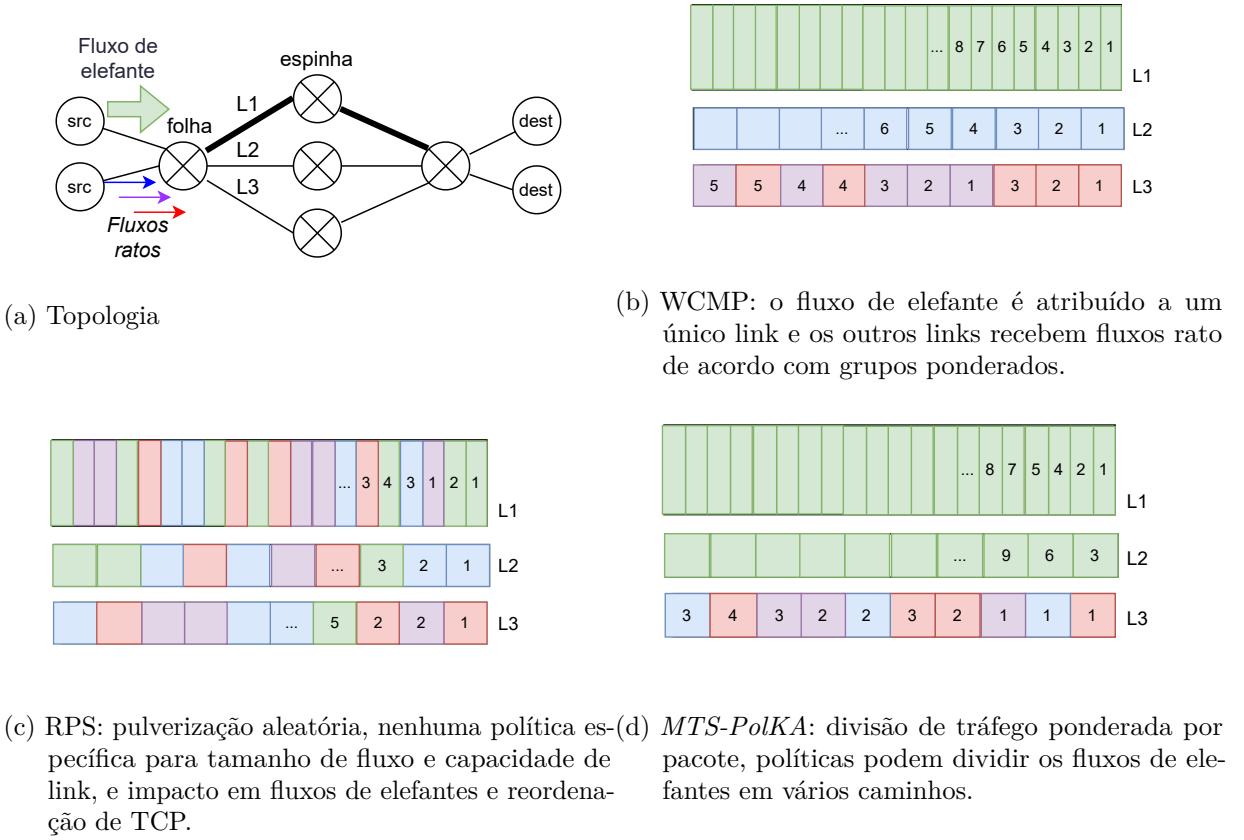
ambos sofrem com baixa utilização de recursos. Já os esquemas de nível de pacotes lidam com reordenação, resultando em problemas de desempenho. Assim, é crucial ponderar esses aspectos ao selecionar a abordagem mais adequada para o gerenciamento do fluxo de dados.

Para resolver essas questões e combinar as vantagens das abordagens de nível de fluxo e de nível de pacotes, foi introduzido o *MTS-PolKA*, que oferece granularidade tanto em nível de fluxo quanto em nível de pacote. Isso é feito por meio de um mecanismo de divisão de tráfego ponderado na rede de núcleo, que distribui cada pacote com base em um rótulo de peso (*weightID*) e um rótulo de rota *multipath* (*routeID*). Além disso, os nós das bordas classificam os fluxos de entrada e atribuem os rótulos adequados de acordo com as políticas específicas de cada fluxo. As Figuras 14 e 15 fornecem uma ilustração mais detalhada dessa discussão.

A Figura 14 ilustra um cenário envolvendo fluxos de tamanhos comparáveis e links

simétricos. No *ECMP*, cada fluxo é atribuído a um único caminho de custo igual e pode acabar compartilhando links com outros fluxos devido a colisões de *hash*, mesmo que caminhos ociosos estejam disponíveis. O *HULA* (*Hierarchical User-level Labeled Architecture*), por outro lado, redireciona fluxos quando um caminho fica congestionado, mas só fornece um melhor caminho por vez para cada fluxo, o que pode resultar na subutilização dos links. O *RPS* (*Random Packet Spraying*) distribui pacotes aleatoriamente pelos links, o que pode levar à reordenação de pacotes e ao aumento de retransmissões *TCP*. Em contrapartida, o *WCMP* atribui fluxos a diferentes grupos e caminhos, mas cada fluxo é limitado a um único caminho por vez, o que pode causar alocação insuficiente de largura de banda. Por fim, o *MTS-PolKA* atribui fluxos a múltiplos grupos de caminhos e permite a distribuição por mais de um caminho por fluxo. Embora essa abordagem ofereça uma oferta mais controlada em comparação ao *RPS*, ainda enfrenta desafios com a reordenação de pacotes.

Figura 15 – Trabalhos relacionados: exemplo ilustrando as filas de links para um cenário com fluxos de elefantes e ratos e links assimétricos.



Fonte: Elaboração própria

A Figura 15 ilustra um cenário com fluxos de elefante e ratos sobre *links* assimétricos. Neste exemplo, o *WCMP* atribui o fluxo de elefante a um único link com maior capacidade, enquanto distribui os fluxos de ratos por outros links com base nas definições de grupos ponderados. Embora essa alocação melhore o desempenho em comparação ao *ECMP*, ainda pode não ser suficiente se o fluxo elefante puder se beneficiar de links adicionais. Em

contrapartida, o *RPS* não aplica políticas específicas para tamanho de fluxo ou capacidade do link, resultando em uma distribuição independente de todos os fluxos. Essa abordagem pode impactar significativamente os fluxos de elefante e agravar a reorganização de pacotes do *TCP* devido à assimetria tanto no tamanho do fluxo quanto na capacidade do link. Por outro lado, *MTS-PolKA* facilita a divisão ponderada de tráfego por pacote e pode conter fluxos de elefante em múltiplos caminhos com maior utilização da rede. No entanto, são necessários mecanismos de enfileiramento mais complexos, como proposto em Huang, Lyu et al. (2021), para lidar eficazmente com a reorganização de pacotes ao dividir pacotes sobre caminhos assimétricos.

4 PROPOSTA

Este capítulo aborda o detalhamento do projeto e implementação da proposta do *MTS-PolKA*. Neste trabalho, apresentamos uma abordagem inovadora com o objetivo de otimizar a divisão de tráfego em um caminho composto por n switches. Cada switch deve ser configurado para distribuir o tráfego entre um subconjunto de suas p portas, considerando os pesos atribuídos a essas portas. Nossa abordagem tem como objetivo superar as limitações das abordagens existentes, como o uso de tabelas estáticas *WCMP* ou estruturas de dados complexas do artigo *DASH (Adaptive Weighted Traffic Splitting in Programmable Data Planes)*, que frequentemente exigem reconfigurações nos switches para ajustar os pesos dos caminhos.

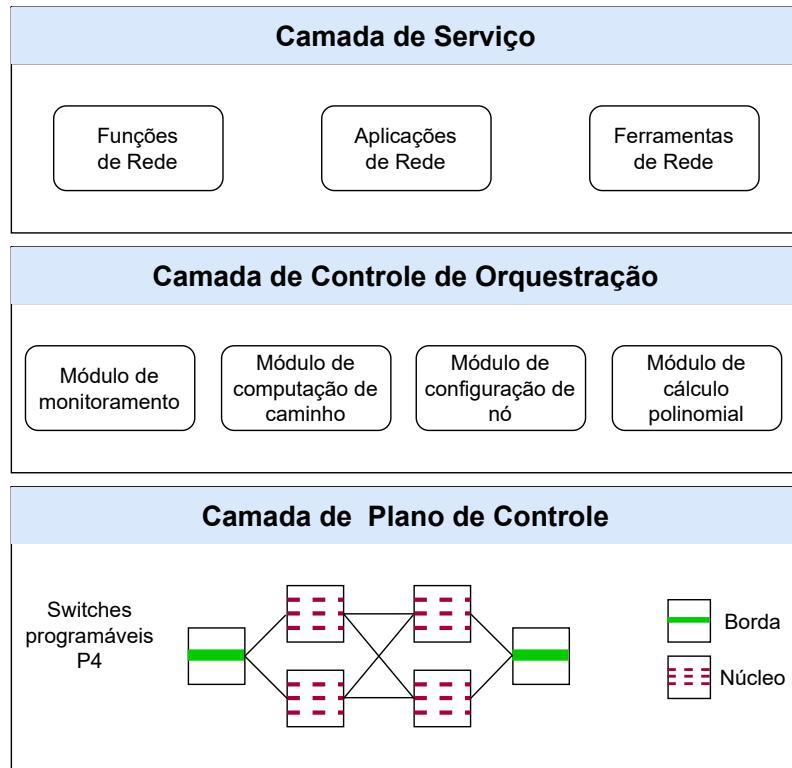
O *MTS-PolKA*, conforme proposto por (SANTOS et al., 2024), deve permitir que a origem selecione a divisão de tráfego desejada, nos multicaminhos, usando rótulos no cabeçalho dos pacotes dos fluxos, e cada switch nos caminhos selecionados deve manter uma tabela estática com um conjunto de perfis de divisão de tráfego disponíveis para seleção. Essa abordagem proporciona flexibilidade ao controlador para alocação de caminhos e distribuição de fluxos, ajustando pesos em tempo real e otimizando recursos em *datacenters*. Isso proporciona alta granularidade e flexibilidade para ajustar a divisão de tráfego em tempo real, sem a necessidade de reconfigurações complexas no núcleo da rede, resultando em melhor desempenho e eficiência na utilização dos multicaminhos redundantes nas redes de *datacenter*.

4.1 ROTEAMENTO M-POLKA

O *MTS-PolKA* utiliza o protocolo *M-PolKA*, conforme descrito na seção 4.1, para conseguir um roteamento de fonte sem estados baseado em *RNS*, onde as decisões de encaminhamento no núcleo dependem de operações de módulo polinomial sobre um rótulo de rota. O *M-PolKA* foi escolhido por ser o único método de roteamento de fonte encontrado na literatura capaz de codificar um rótulo de rota para um ciclo de multicaminho ou árvore de maneira agnóstica à topologia.

O *M-PolKA* é uma maneira promissora de executar um roteamento de fonte totalmente sem armazenamento do estado, no qual as decisões de encaminhamento nos nós centrais dependem de uma operação de módulo simples sobre um rótulo de rota. Entretanto, o *M-PolKA* original clona os pacotes para todas as portas ativas sem fazer nenhuma divisão de tráfego. De forma diferente, este trabalho propõe a aplicação do *M-PolKA* para dividir o tráfego com proporção de peso entre as portas de saída. Isso permite explorar a diversidade de caminhos para fornecer balanceamento de carga.

Figura 16 – Arquitetura *MTS-PolKA*.



Fonte: Adaptado de Guimarães et al. (2022).

4.2 ARQUITETURA DO *MTS-POLKA*

O controlador *MTS-PolKA* calcula três identificadores polinomiais: i) *nodeID*, um identificador atribuído aos *switches* do núcleo na configuração da rede; ii) *routeID*, um identificador de rota multicaminhos; e iii) *weightID*, um identificador de peso que define o perfil de divisão de tráfego em cada *switch* no caminho do tráfego. Tanto o *routeID* quanto o *weightID* são calculados por meio do Teorema Chinês dos Restos (*CRT, Chinese Remainder Theorem*) (DOMINICINI et al., 2020a) e incorporados aos pacotes pelos *switches* de borda, que possuem regras de fluxo que podem ser instaladas de forma proativa pelo controlador, sem a necessidade de se consultar o plano de controle para cada pacote que ingressa na rede.

No plano de dados, a implementação do *mod* polinomial explora a operação de *CRC (Cyclic Redundancy Check)* suportada pelo equipamento. Como detalhado em (GUIMARÃES et al., 2022), o tamanho do *routeID* é dado pela multiplicação do número máximo de saltos da topologia por 2 bytes (para uso do algoritmo *CRC16* em topologias com até 4.080 nós). A mesma lógica pode ser adotada para o cálculo do *weightID*. A implementação deste trabalho adotou um número máximo de 10 saltos com um cabeçalho de 20 bytes para cada campo de *routeID* e *weightID*. Assim, existe um custo em termos de bits no cabeçalho para explorar o sistema *RNS* no roteamento, mas que pode ser reduzido aplicando algumas técnicas de codificação conhecidas (DOMINICINI et al., 2020a).

Além disso, dois identificadores polinomiais são calculados, no plano de dados, para cada pacote: i) *portID*, um identificador que representa as portas de saída ativas de cada *switch* de núcleo, ou seja, quais portas transmitirão uma proporção do tráfego a ser encaminhado; e ii) *profileID*, um identificador obtido pela operação de módulo entre o *weightID* do pacote e o *nodeID*, que representa o perfil de proporção de divisão de tráfego nas portas de saída ativas do *switch*. Em resumo, a arquitetura *MTS-PolKA* é composta por nós no núcleo da rede (em vermelho), nós de borda (em azul) e um controlador logicamente centralizado, conforme Figura 16. As camadas da arquitetura são descritas a seguir:

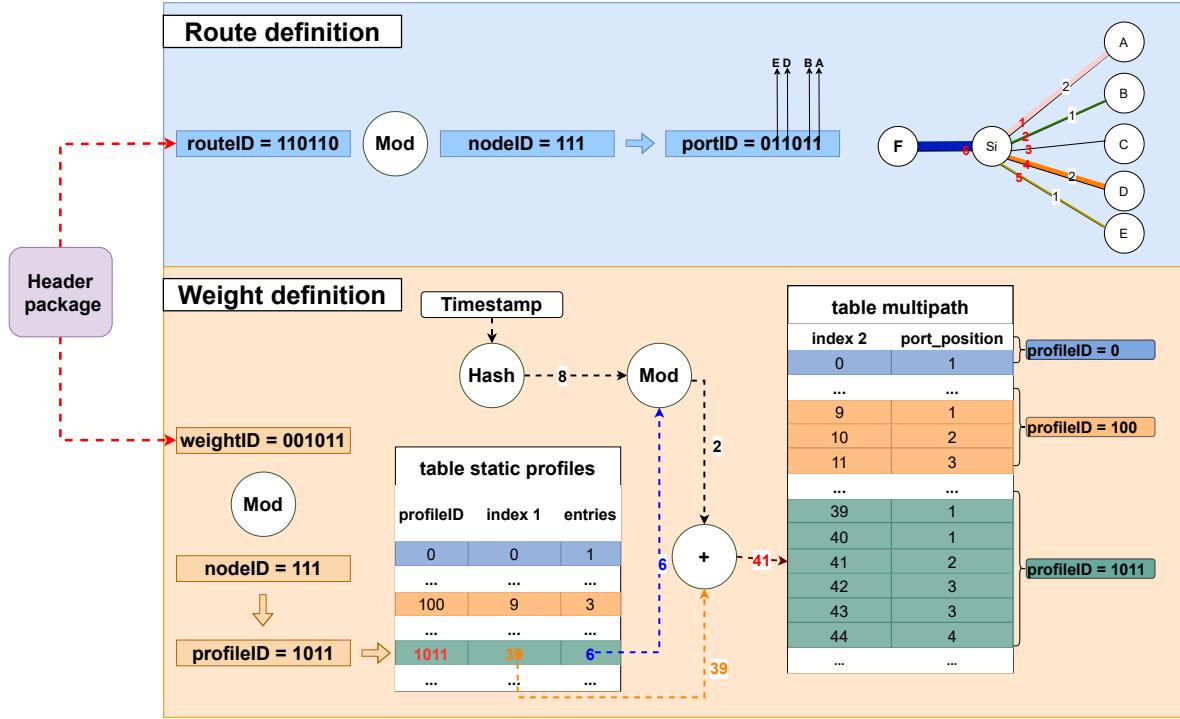
- **Camada do Plano de Dados:** Assume uma topologia física composta por *switches* programáveis habilitados para P4, atuando como nós de borda ou núcleo para a divisão de tráfego multicaminho em proporção de peso e roteamento na fonte.
- **Camada de Controle e Orquestração:** Composta por quatro módulos:
 1. **Monitoramento:** descobre a topologia e coleta dados sobre o estado da rede, tais como falha de enlace e congestionamento, a fim de fornecer um ciclo de *feedback* para decisões de orquestração;
 2. **Computação de caminho:** calcula múltiplos caminhos para um fluxo, considerando opções de nós e enlaces, disjuntos ou não disjuntos;
 3. **Configuração de nó:** configura *nodeIDs* e tabelas de perfis de tráfego nos nós de núcleo, além de configurar as entradas de fluxo em nós de borda;
 4. **Cálculo polinomial:** calcula *weightIDs*, *routeIDs* e *nodeIDs* para os caminhos e distribuições de tráfego definidos.
- **Camada de Serviço:** Responsável pela implementação de funções de rede, aplicações do usuário e demais ferramentas utilizadas nas etapas de implantação e validação da arquitetura.

4.2.1 Plano de Controle

Como ilustrado na Figura 17, o controlador *MTS-PolKA* tem duas atribuições principais: i) definir a rota, baseando-se na definição de uma árvore *multicast* que identifica, para cada um dos *switches* no caminho, quais são as portas ativas que devem encaminhar os pacotes, conforme representação desta árvore *multicast*; e ii) definir os pesos para divisão de tráfego em cada porta ativa de cada um dos nós da árvore *multicast*.

No núcleo da rede, o controlador é responsável por configurar os *switches* com os *nodeIDs*, e as tabelas estáticas de cada *switch* com os perfis de divisão de tráfego disponíveis. Na

Figura 17 – Exemplo de funcionamento do *MTS-PolKA* com perfil de divisão de tráfego selecionado pela origem com um rótulo no cabeçalho do pacote. Ambas as tabelas são configuradas previamente pelo controlador.



Fonte: Elaboração própria

borda, por sua vez, o controlador disponibiliza um conjunto de multicaminhos e perfis de divisão de tráfego por meio dos *routeIDs* e *weightIDs*, que podem ser escolhidos pelos fluxos ingressantes, conforme o estado da rede, direcionando e realizando o balanceamento do tráfego. Essa solução explora a diversidade de caminhos disponíveis na rede, resultando em uma utilização eficiente da largura de banda disponível na rede.

4.2.2 Plano de Dados

No *MTS-PolKA*, o plano de dados utiliza rótulos específicos para controlar o roteamento e a divisão de tráfego de maneira otimizada, proporcionando balanceamento de carga conforme a necessidade de cada fluxo. Para isso, o controlador pré-configura tabelas tanto nos *switches* de borda quanto nos *switches* de núcleo, definindo caminhos e políticas de balanceamento de tráfego para cada perfil de rede.

Cada *switch* de borda possui uma **tabela de fluxos**, previamente preenchida pelo controlador, que associa informações dos fluxos (por exemplo, IP de destino e portas) a rótulos específicos: o *routeID* (usado para definir a rota de saída) e o *weightID* (que define o perfil de balanceamento de tráfego). A partir desses rótulos, o plano de dados se encarrega de aplicar as regras de roteamento e balanceamento nos *switches* de núcleo, conforme o caminho e o perfil selecionados.

Nos *switches* de núcleo, são utilizadas duas tabelas principais, que não precisam ser alteradas durante a operação para seleção de diferentes perfis de tráfego (diferentemente dos trabalhos relacionados):

- **Tabela de Perfis Estáticos (table static profiles):** Esta tabela, configurada pelo controlador, contém os perfis de divisão de tráfego. Cada perfil é identificado por um *profileID* e define a proporção de tráfego a ser distribuída entre as portas ativas. O perfil é selecionado com base no *weightID* do pacote. A quantidade de perfis e a proporção de tráfego para cada perfil são definidos no plano de controle, permitindo ajustes dinâmicos e uma gestão eficiente de recursos.
- **Tabela de Múltiplos Caminhos (table multipath):** Esta tabela define a distribuição exata do tráfego entre as portas de saída, considerando as proporções dos perfis na **table static profiles**. Ela contém várias entradas para cada perfil de tráfego, onde cada entrada representa um “peso” associado a uma porta específica. Dessa forma, o tráfego pode ser dividido conforme as proporções esperadas,平衡ando o uso das portas de saída de acordo com o perfil aplicado.

O processo completo, desde o ingresso até a saída do pacote, funciona da seguinte maneira:

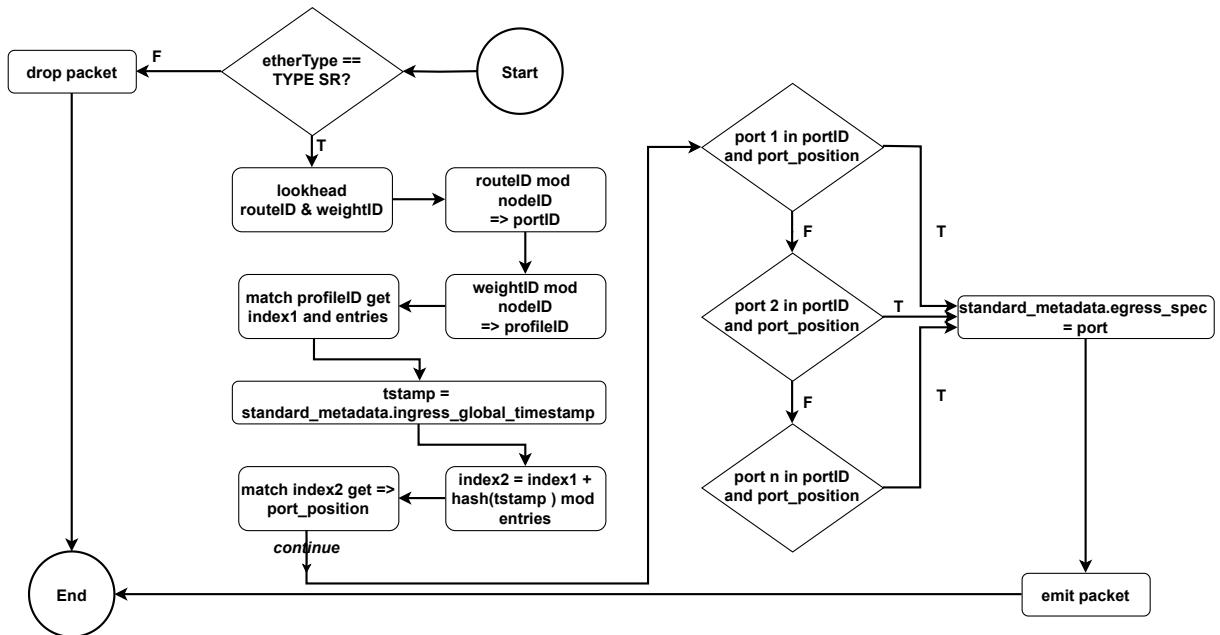
- 1. Ingresso do Pacote:** No ingresso de um pacote em um *switch* de borda, o controlador já configurou a **tabela de fluxos** para inserir automaticamente os rótulos *routeID* e *weightID* de acordo com as características do fluxo, como IP de destino e portas de origem e destino. Esses rótulos são adicionados ao cabeçalho do pacote e servem como referência para os *switches* de núcleo no processo de roteamento e balanceamento.
- 2. Definição de Rota:** Ao receber um pacote rotulado, o *switch* de núcleo executa uma operação de módulo (MOD) entre o *routeID* e o *nodeID* do *switch*, o que gera um identificador *portID* que indica as portas de saída ativas para aquele pacote. As portas ativas correspondem aos *bits* com valor “1” em *portID*, analisados da direita para a esquerda. Essas portas são utilizadas para garantir que o tráfego siga o caminho configurado pelo controlador.
- 3. Seleção de Perfil de Balanceamento:** Com o *weightID* do pacote, o *switch* determina o *profileID* consultando a **table static profiles**. Esse *profileID* é usado para acessar a **table multipath**, onde a proporção de tráfego para cada porta ativa é definida em função do perfil selecionado.
- 4. Distribuição de Tráfego:** Uma função de *hashing* baseada no *timestamp* de chegada do pacote é aplicada para determinar a entrada exata na **table multipath**. O valor de *hashing* é combinado com o índice de início do perfil para selecionar a porta de saída final,

garantindo que a distribuição do tráfego respeite a proporção configurada. Essa função assegura que o tráfego seja balanceado conforme o peso definido para cada porta ativa.

A Figura 17 mostrou um exemplo de funcionamento é apresentado: o *switch* de núcleo *Si* possui 6 portas com pesos de distribuição 2:1:2:1 para as saídas A, B, D e E, respectivamente. A operação de MOD entre *routeID* = 110110 e *nodeID* = 111 define *portID* como 011011, indicando as portas de saída ativas. Em seguida, o *weightID* gera *profileID*, que acessa a **table multipath** e determina a proporção de tráfego nas portas de saída, resultando em um encaminhamento balanceado. Este exemplo será explicado com mais detalhes na próxima seção.

Esse método, utilizando o controlador e tabelas pré-configuradas, possibilita que o *MTS-PolKA* ofereça uma divisão de tráfego eficiente e adaptável sem que os *switches* de núcleo necessitem de reconfigurações adicionais, permitindo um平衡amento de carga dinâmico e escalável para fluxos com diferentes necessidades de largura de banda.

Figura 18 – *Pipeline* na linguagem P4 dos *switches* de núcleo do *MTS-PolKA*.



Fonte: Elaboração própria

A Figura 18 apresenta a mesma lógica do exemplo da Figura 17, mas representando a implementação usando a linguagem P4 do *pipeline* do plano de dados dos *switches* de núcleo no *MTS-PolKA*. Na etapa de *parsing*, se o campo *etherType* for *TYPE_SR*, a próxima etapa é a leitura dos rótulos *routeID* e o *weightID*, via *lookahead*.

Com base nos valores lidos, o bloco de *ingress* realiza operações de módulo polinomial entre o *routeID* e o *nodeID* e entre o *weightID* e o *nodeID*, para calcular os estados das portas de transmissão (*portID*) e o perfil de divisão de tráfego (*profileID*), respectivamente.

Para o *routeID*, o *switch* utiliza duas operações de *SHIFT*, uma de *CRC* e duas de *XOR* (DOMINICINI et al., 2021). Da mesma forma, para o *weightID*, o *switch* aplica duas operações de *SHIFT*, uma de *CRC* e duas de *XOR*. Essas operações permitem determinar tanto a porta de saída (*portID*) quanto o perfil de divisão de tráfego (*profileID*) para encaminhar o pacote de maneira eficiente.

Depois, o *profileID* é utilizado para busca na tabela `static profiles` para recuperar os valores das variáveis *entries* e *index1*. Esses valores são usados em uma operação de módulo inteiro com o *hashing* do *timestamp* do pacote para obter a variável $index2 = index1 + hash(tstamp) \bmod entries$. Então, este último índice é usado para acessar a tabela `table multipath` e identificar qual porta ativa deve transmitir aquele pacote específico (*port_position*). Por fim, é feita uma busca por qual é a porta ativa no vetor de estados que corresponde ao valor de *port_position*, e, uma vez encontrada, o pacote é transmitido por essa porta.

Com essa técnica, o *MTS-PolKA* pode ser implantado em *switches P4* que permitem a configuração de polinômios geradores. Como o *P4* suporta operações de *CRC* através do uso de bibliotecas externas, o suporte para polinômios personalizados depende dos modelos de arquitetura e de como os *switches* específicos os implementam. As arquiteturas *PSA*³ e *v1model*⁴ suportam polinômios personalizados de 16 e 32 bits. Em termos de *targets*, o *switch* de *software bmv2*⁵ e o *switch* de *hardware Tofino® da Barefoot* suportam polinômios de *CRC* personalizados, enquanto os *SmartNICs da Netronome* suportam apenas polinômios de *CRC* fixos (GUIMARÃES et al., 2022).

4.3 EXEMPLO DETALHADO SOBRE O FUNCIONAMENTO DO *MTS-POLKA*

Esta seção detalha o exemplo ilustrado na Figura 17 sobre o funcionamento do *MTS-PolKA*.

4.3.1 Introdução ao Funcionamento

O *MTS-PolKA* consiste em quatro etapas principais: **Ingresso do Pacote, Definição de Rota, Seleção de Perfil, e Distribuição de Tráfego**. A seguir, cada uma dessas etapas será detalhada:

4.3.2 Ingresso do Pacote

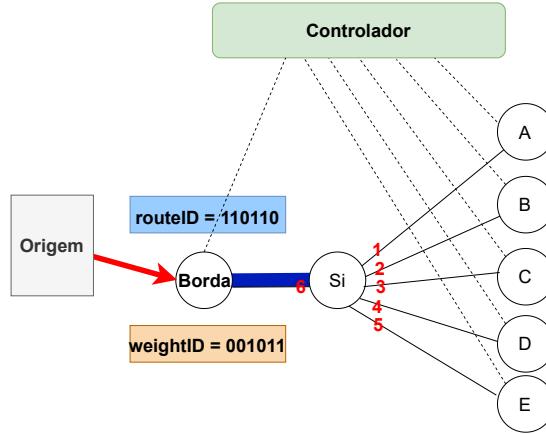
Cada pacote que entra na rede recebe dois identificadores essenciais: o *routeID* e o *weightID*. O *routeID* define a rota que o pacote seguirá, enquanto o *weightID* define o peso que será utilizado na distribuição de tráfego, conforme a Figura 19

³ <<https://p4.org/p4-spec/docs/PSA.html>>

⁴ <<https://github.com/p4lang/p4c/blob/master/p4include/v1model.p4>>

⁵ <<https://github.com/p4lang/behavioral-model>>

Figura 19 – Ingresso de um pacote na rede com $routeID = 110110$ e $weightID = 001011$.



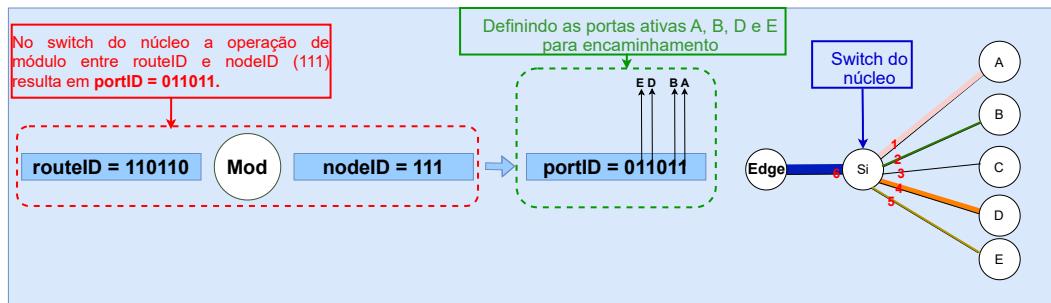
Fonte: Elaboração própria

4.3.3 Definição de Rota

A definição de rota é feita no *switch* do núcleo, onde o *routeID* do pacote é usado para calcular o *portID* por meio de uma operação de módulo entre o *routeID* e o *nodeID* do *switch*. O resultado dessa operação determina as portas ativas que serão usadas para encaminhar o pacote.

Conforme a Figura 20, se o *routeID* = 110110 e o *nodeID* = 111, então o *portID* resultante será 011011. Isso significa que as portas **A, B, D e E** serão as portas ativas. O *portID* é interpretado da direita para a esquerda, onde cada bit "1" indica que a respectiva porta está ativa.

Figura 20 – Mostrando a operação de módulo e as portas ativas resultantes.



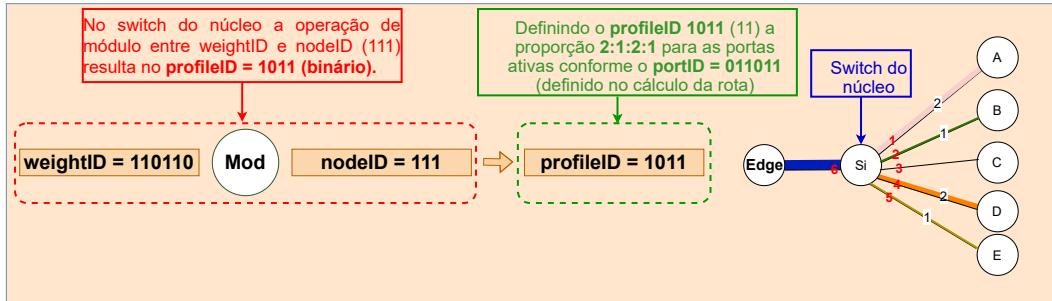
Fonte: Elaboração própria

4.3.4 Seleção de Perfil

Na etapa de seleção de perfil, o sistema calcula o *profileID* usando o *weightID* do pacote e o *nodeID* do *switch*. O *profileID* define a proporção de tráfego para cada porta ativa. Neste exemplo, o perfil 1011 indica uma distribuição de tráfego na proporção **2:1:2:1** entre as portas ativas **A, B, D e E**, previamente calculadas no *portID*.

Conforme a Figura 21, se $weightID = 001011$ e $nodeID = 111$, o $profileID$ resultante é 1011.

Figura 21 – Cálculo do $profileID$ e a distribuição do tráfego entre as portas ativas.

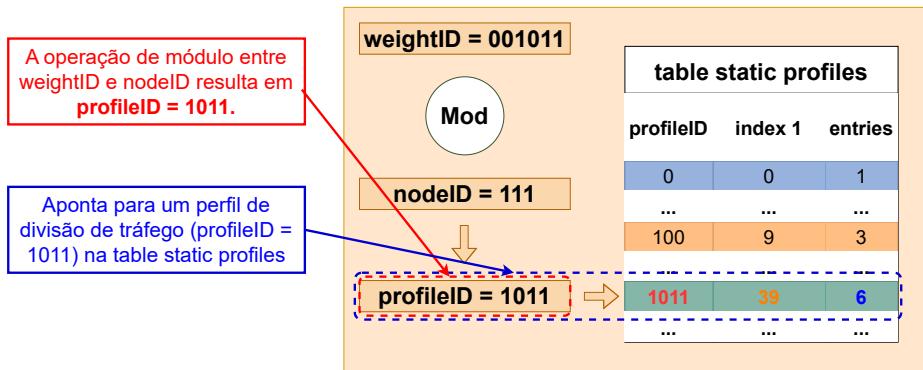


Fonte: Elaboração própria

No *MTS-PolKA* as alterações nos pesos não requerem modificações nas tabelas de cada *switch* ao longo do caminho, pois são definidas exclusivamente na origem e inseridas no cabeçalho do pacote.

Os *switches* do núcleo da rede mantêm tabelas estáticas com perfis de distribuição de tráfego, eliminando a necessidade de reconfigurações frequentes. O sistema seleciona o perfil de distribuição de tráfego correspondente ($profileID = 1011$) na **table static profiles** (tabela de perfis estáticos), conforme a Figura 22.

Figura 22 – Seleção do perfil de peso ($profileID$) na **table static profiles**



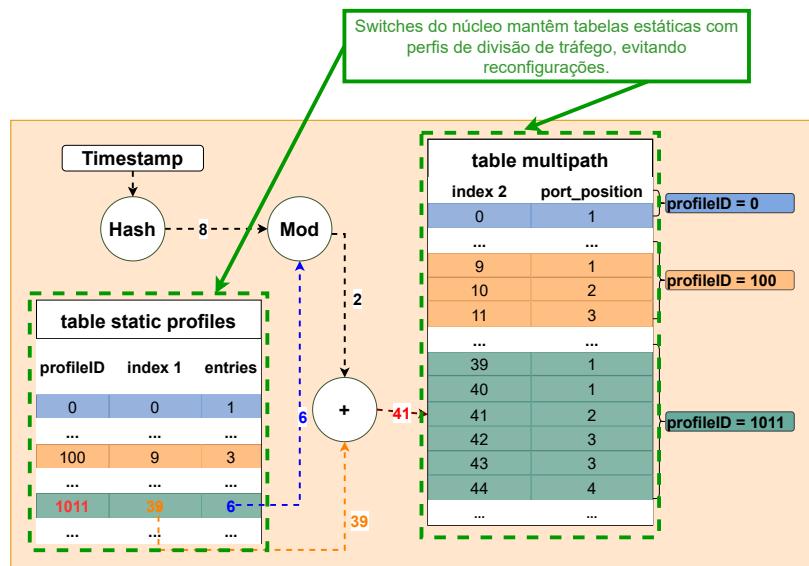
Fonte: Elaboração própria

4.3.5 Distribuição de Tráfego

Os *switches* do núcleo da rede mantêm duas tabelas estáticas **table static profiles** e **table multipath**, conforme a Figura 23. Estas tabelas são componentes essenciais na gestão da divisão de tráfego em *switches*, especialmente na arquitetura *MTS-PolKA*. A **table static profiles** armazena perfis de divisão de tráfego pré-definidos, que especificam como o tráfego deve ser distribuído entre as portas ativas de um *switch*, permitindo uma seleção rápida e eficiente de perfis com base no rótulo $weightID$ dos pacotes. Por sua vez, a **table multipath** define a distribuição exata do tráfego para cada porta ativa,

acessando entradas específicas que correspondem ao perfil selecionado, o que possibilita uma adaptação ágil às variações na demanda de tráfego. Juntas, essas tabelas eliminam a necessidade de reconfigurações constantes, promovendo uma engenharia de tráfego mais dinâmica e eficiente em ambientes de *datacenters*.

Figura 23 – Tabelas Estáticas



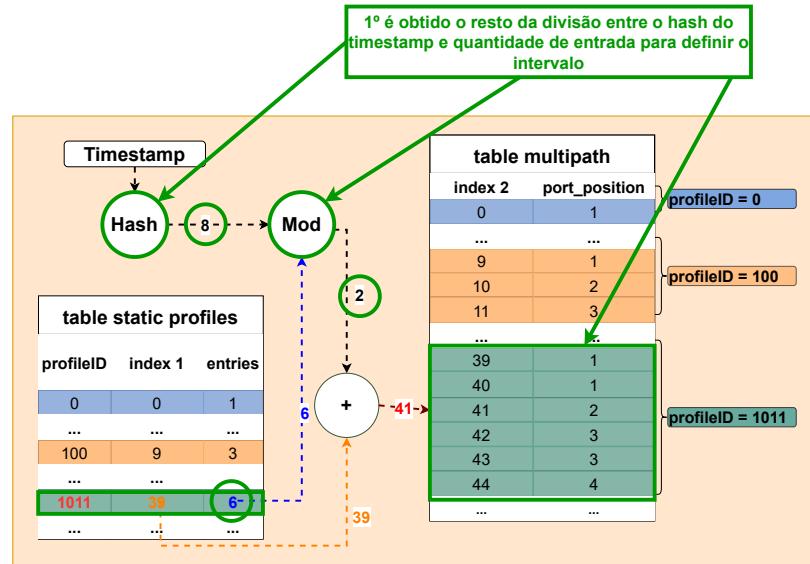
Fonte: Elaboração própria

A distribuição de tráfego utiliza um algoritmo de seleção de portas baseado no *hash* do *timestamp* do pacote. A operação de módulo entre o *hash* do *timestamp* e o índice de entrada define o intervalo de tempo em que o pacote será roteado para uma porta ativa específica.

Etapas:

1. O *hash* do *timestamp* do pacote é calculado e usado para selecionar um intervalo de tempo, conforme a Figura 23.

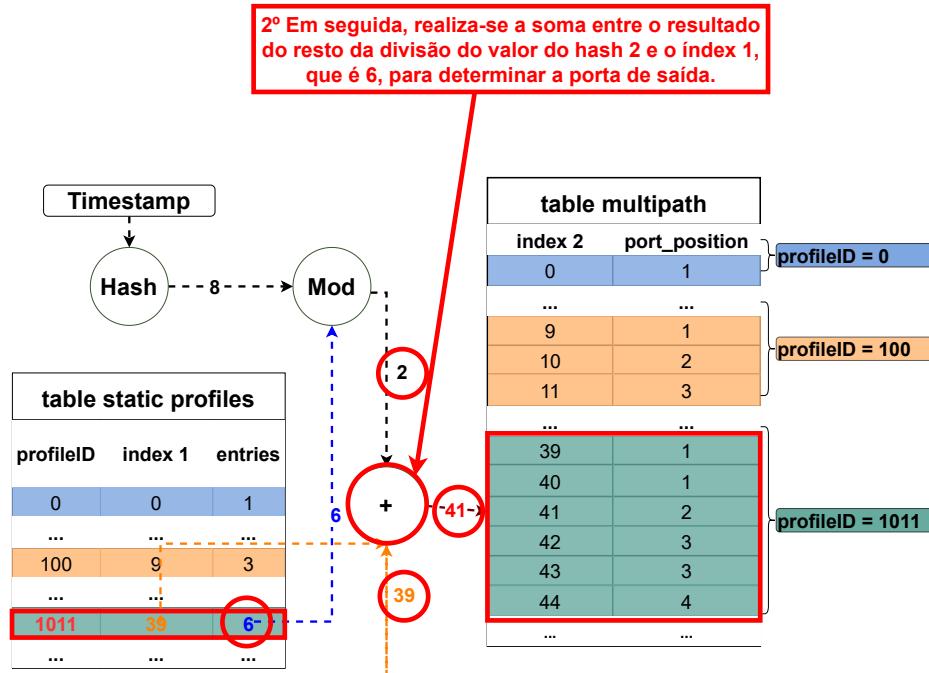
Figura 24 – Tabelas estáticas `table static profiles` e `table multipath`



Fonte: Elaboração própria

2. A soma do *hash* com o índice de entrada define a porta de saída, conforme a Figura 25.

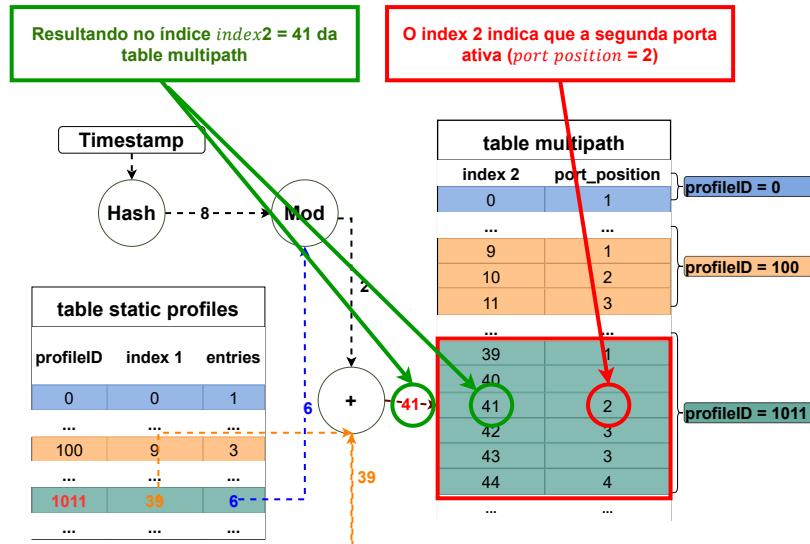
Figura 25 – Soma do *hash* do *timestamp* com o índice de entrada da tabela `table static profiles`



Fonte: Elaboração própria

3. Isso resulta no índice $index2 = 41$ da tabela `table multipath`. O índice 2 indica que a segunda porta ativa ($port_position = 2$), conforme a Figura 26.

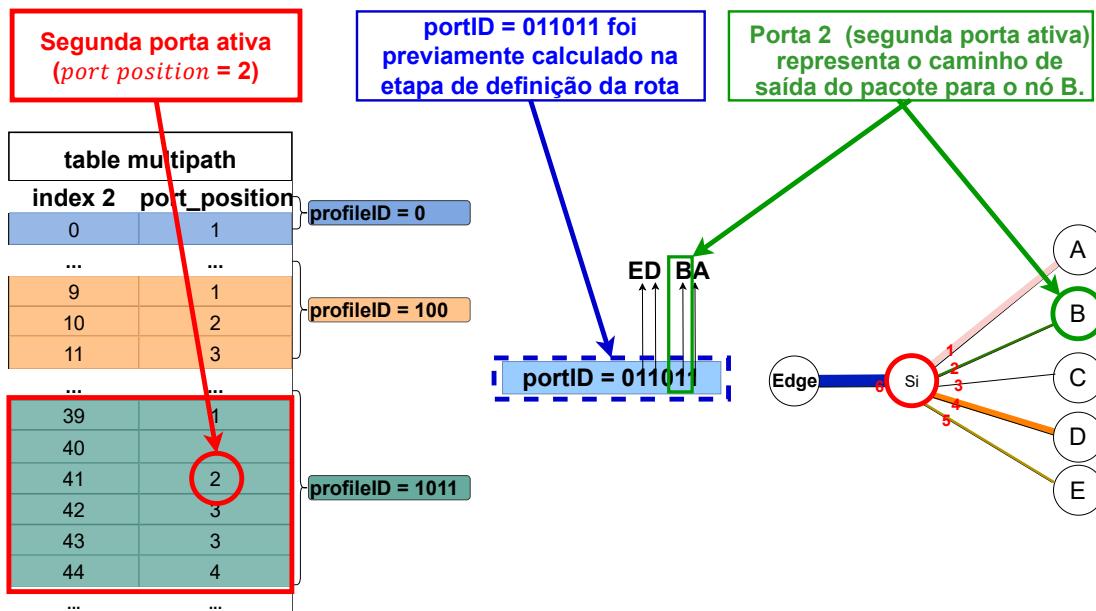
Figura 26 – Definição da Porta de saída



Fonte: Elaboração própria

4. A segunda porta ativa, identificada como $port_position = 2$ e com um $portID$ de 011011, foi previamente determinada durante a etapa de definição da rota. Essa porta representa o caminho de saída do pacote para o nó B, permitindo que o tráfego seja direcionado de forma eficiente através da rede. Assim, a configuração da porta assegura que os pacotes sejam encaminhados corretamente, seguindo a lógica de divisão de tráfego estabelecida, conforme a Figura 27.

Figura 27 – Definição da Porta de saída

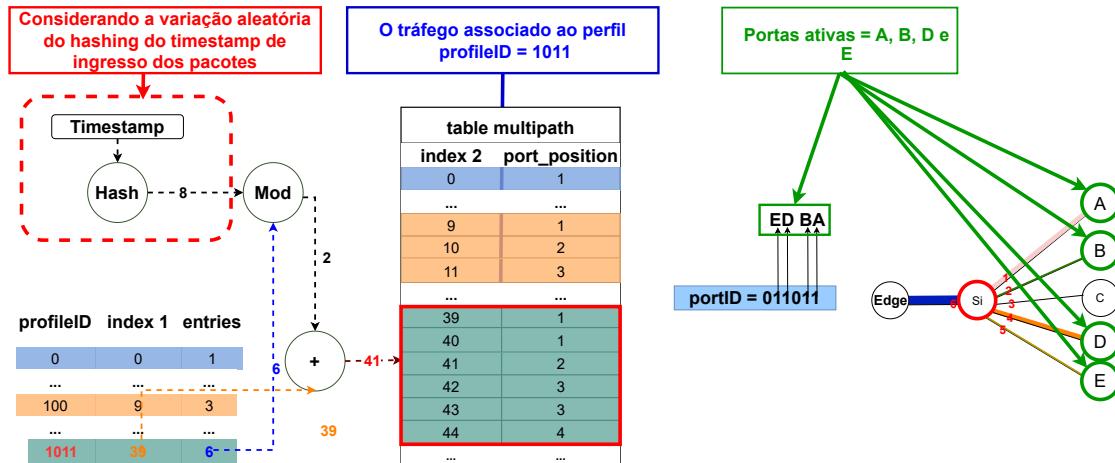


Fonte: Elaboração própria

5. Aplicando o algoritmo de seleção de portas descrito até aqui, e considerando a variação

aleatória do *hashing* do *timestamp* de ingresso dos pacotes, o tráfego associado ao perfil *profileID* = 1011 será espalhado entre as portas ativas com uma proporção de pesos igual a **2:1:2:1**, conforme a Figura 28.

Figura 28 – Aplicação do algoritmo de seleção das portas



Fonte: Elaboração própria

5 PROVA DE CONCEITO

Para avaliar a proposta apresentada no Capítulo 4 foi desenvolvido um protótipo em linguagem P4 com o emulador *Mininet*. Nos testes emulados, demonstramos o *MTS-PolKA*⁶, uma solução de divisão de tráfego com roteamento de fonte multicaminho que explora a diversidade de caminhos e aplica diferentes perfis de peso para distribuir eficientemente o tráfego em redes de *datacenter*.

5.1 PROTÓTIPO

A prova de conceito foi realizada utilizando um servidor *PowerEdge R650* com 32 GB de RAM e processador *Intel Xeon Silver 4314* de 2.4 GHz, empregando o emulador *Mininet* para a emulação da rede, compilação de programas P4-16 e configuração do *switch* de software `bmv2 simple_switch` com a arquitetura `v1model`.

Para realizar os cálculos polinomiais, como o cálculo *RNS* de *nodeIDs* e *routeIDs*, utilizamos uma biblioteca *Python*, com o pacote *galoistools* da biblioteca *sympy*. Além disso, foi implementado um aplicativo de plano de controle que interage com os *switches* através da CLI do *bmv2 simple_switch* e se conecta via porta *Thrift RPC*. Esse aplicativo implementa a camada de Controle e Orquestração, conforme capítulo 1.2, com exceção do módulo de monitoramento que está fora do escopo deste trabalho.

Para implementação das operações de resto da divisão, foi adotada a reutilização da operação de CRC, conforme proposto no trabalho do M-PolKA (GUIMARÃES et al., 2022). Embora nossos testes pudessem usar CRC8 para topologias de até 30 nós, já que o *simple_switch* suporta apenas a definição de polinômios CRC de 16 e 32 bits, a implementação P4 adotou o CRC16.

5.2 EXPERIMENTOS

Quatro experimentos foram realizados: i) um teste funcional, para demonstrar a aplicação de diferentes perfis de divisão de tráfego; ii) um teste de reconfiguração ágil de perfis de divisão de tráfego na fonte; iii) um teste com múltiplos fluxos concorrentes; e iv) uma comparação do tempo de conclusão do fluxo com trabalhos relacionados. Os testes envolveram medições usando a ferramenta *bwm-ng* e foi gerado pelo *iperf3*, com um *host* como servidor e outro como cliente. Nos testes 1 e 2, os pacotes UDP foram enviados com uma taxa de transferência de 10 Mbps. No teste 3, os fluxos *UDP* foram enviados com uma taxa de transferência de 4 Mbps. No teste 4, foi usado um atraso de 1 ms foi introduzido, com *links* do núcleo configurados para 1 Mbps e links de borda para 10 Mbps. Foram enviados 2 fluxos de 100 MB e múltiplos fluxos de 10 KB (4 por segundo), todos

⁶ <<https://github.com/giancarloliver/MTS-PolKA>>

utilizando o protocolo *TCP*. O tráfego foi capturado nas máquinas de destino, e dados de latência foram coletados com a ferramenta *ping* para análise.

5.2.1 Teste 1: Validação Funcional

- **Objetivos dos Testes**

- **Teste com Perfil 0**

- * **Objetivo:** Verificar a eficácia do roteamento de tráfego utilizando uma configuração de peso uniforme (1) que prioriza uma única porta de saída (*S1_0-eth4*) no *switch S1_0*.
- * **Expectativa:** Através da árvore de transmissão indicada, o tráfego deveria ser totalmente direcionado para a porta ativa, permitindo uma análise de como a distribuição de tráfego se comporta em uma configuração com perfil estático.

- **Teste com Perfil 100**

- * **Objetivo:** Avaliar a performance do *switch S1_0* quando o tráfego é dividido igualmente entre três portas ativas (*S1_0-eth3*, *S1_0-eth4* e *S1_0-eth6*) utilizando uma proporção de pesos (1:1:1).
- * **Expectativa:** Observar se a divisão igual do tráfego resulta em uma distribuição balanceada de pacotes UDP entre os *switches* ao longo do caminho.

- **Teste com Perfil 1011**

- * **Objetivo:** Analisar a distribuição de tráfego em um cenário mais complexo, onde o tráfego é dividido entre quatro portas ativas no *switch S1_0* com uma proporção de pesos variada (2:1:2:1).
- * **Expectativa:** A distribuição assimétrica do tráfego deve refletir na quantidade de pacotes UDP recebidos nos *switches*, demonstrando a eficácia do uso de pesos variáveis na gestão do tráfego pelo *switch*.

- **Métricas Avaliadas**

- **Número de Pacotes**

- * Contagem total de pacotes UDP recebidos por cada *switch* em cada teste. Essa métrica é crucial para entender a carga em cada porta e a eficácia da estratégia.

de roteamento.

– Proporção de Tráfego

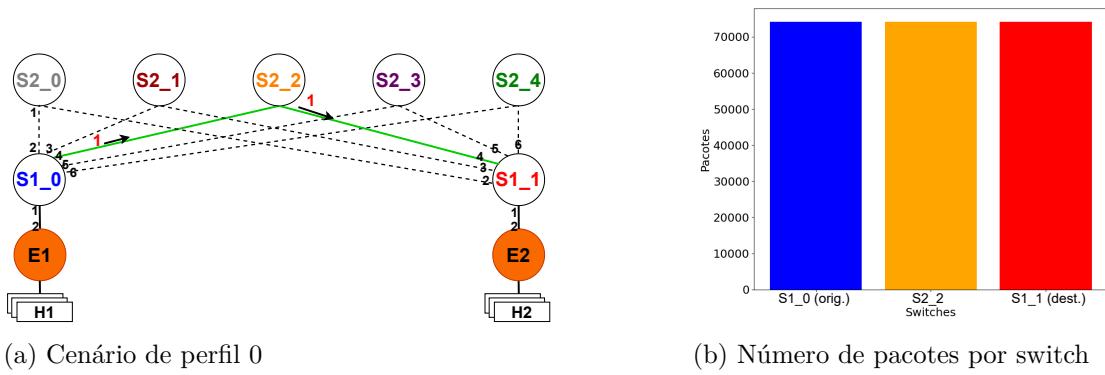
- * Avaliação da proporção de tráfego direcionado a cada porta ativa, em relação ao total de pacotes. Isso permite verificar se os pesos aplicados nos perfis de tráfego estão sendo respeitados durante os testes.

Com esses objetivos e métricas em mente, é possível entender melhor o propósito de cada experimento e avaliar a eficácia do gerenciamento de tráfego nos *switches* da rede.

Considere o seguinte cenário: três perfis de tráfego diferentes foram selecionados na origem para um determinado núcleo *switch* (*S1_0*), que representa, por exemplo, um *switch* de acesso em uma rede *datacenter* com topologia hierárquica. Os três perfis estão representados nas Figuras 29a, 30a e 31a que serão detalhadas posteriormente nesta seção. Os *switches* restantes mantiveram uma distribuição de tráfego constante usando o Perfil 0, que representa, neste exemplo, o direcionamento de todo o tráfego pela primeira porta ativa. Como já explicado, o perfil de divisão de tráfego foi determinado pela origem, e inserido no fluxo pelo *switch* de borda, através do uso de dois rótulos no cabeçalho do pacote: i) o *routeID*, para selecionar as portas ativas em cada *switch* do caminho e montar a árvore de transmissão; e ii) *weightID*, para selecionar a proporção de pesos.

Nesses testes, os perfis 0, 100 e 1011 foram configurados no *switch* *S1_0*, enquanto os demais *switches* mantiveram uma distribuição de tráfego constante usando o perfil 0, direcionando todo o tráfego pela primeira porta ativa.

Figura 29 – Teste 1: perfil 0



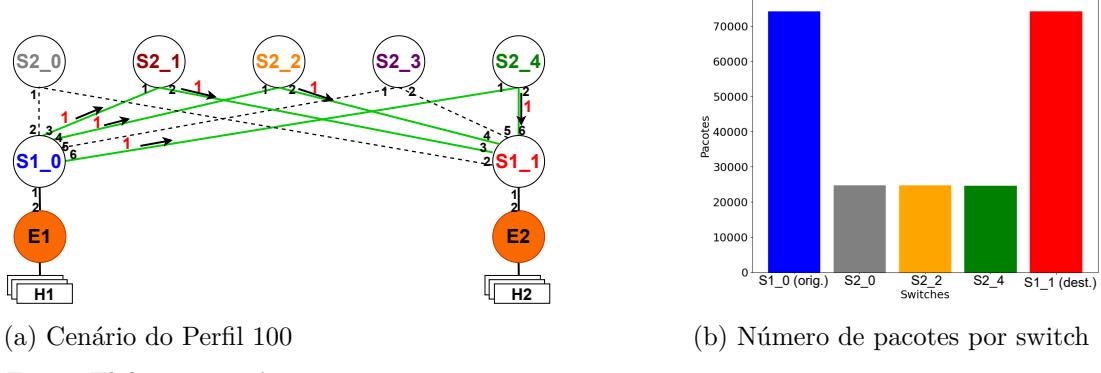
(a) Cenário de perfil 0

(b) Número de pacotes por switch

Fonte: Elaboração própria

Teste com Perfil 0 (Figura 29): Usando a árvore de transmissão indicada (Figuras 29a) e uma taxa de peso de 1 na primeira porta ativa, todo o tráfego de saída foi roteado para a porta *S1_0-eth4* no *switch* *S1_0*. Os resultados (Figura 29b) mostraram a divisão balanceada do tráfego UDP de acordo com o perfil 0 nos *switches* *S1_0*, *S2_2* e *S1_1*.

Figura 30 – Teste 1: perfil 100

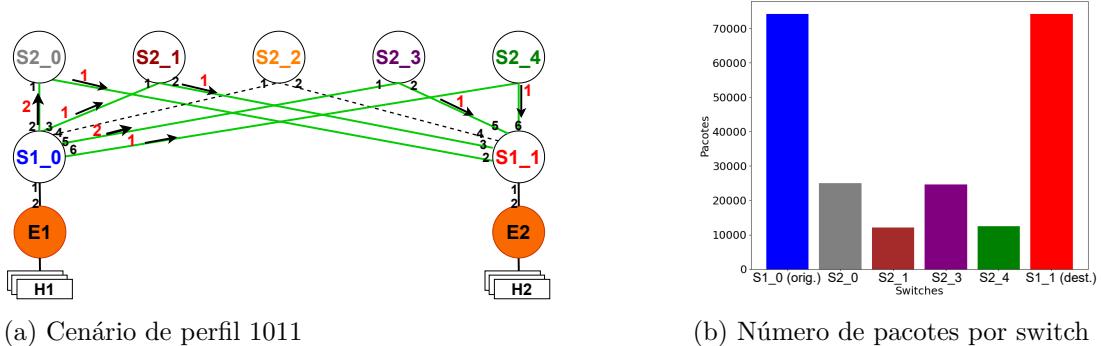


(a) Cenário do Perfil 100

Fonte: Elaboração própria

Teste com Perfil 100 (Figura 30): Usando a árvore de transmissão indicada (Figura 30a) e uma proporção de peso de 1 :1:1, o *switch* S1_0 dividiu o tráfego de saída igualmente entre suas três portas ativas (S1_0-eth3, S1_0-eth4 e S1_0-eth6). A Figura 30b apresenta os resultados, ilustrando a distribuição de pacotes UDP seguindo o perfil 4 (1:1:1) em *switches* S1_0, S2_0, S2_2, S2_4 e S1_1.

Figura 31 – Teste 1: perfil 1011



(a) Cenário de perfil 1011

Fonte: Elaboração própria

Teste com Perfil 1011 (Figura 31): Usando a árvore de transmissão indicada (Figura 31a) e uma proporção de peso de 2: 1:2:1, *switch* S1_0 distribuiu o tráfego de forma variada em suas quatro portas ativas. Os resultados, representados na Figura 31b, demonstraram a distribuição dos pacotes UDP alinhados com o perfil 11 (2:1:2:1) nos *switches* S1_0, S2_0, S2_2, S2_4 e S1_1.

5.2.2 Teste 2: Reconfiguração Ágil

- **Objetivo do Teste :**

- Avaliar a reconfiguração ágil da divisão de tráfego entre diferentes perfis (Perfil 0, Perfil 100 e Perfil 1011) em um ambiente de rede, com foco em manter a estabilidade do fluxo durante as transições entre os perfis e demonstrar a eficácia do sistema *MTS-PolKA* na gestão da divisão de tráfego.

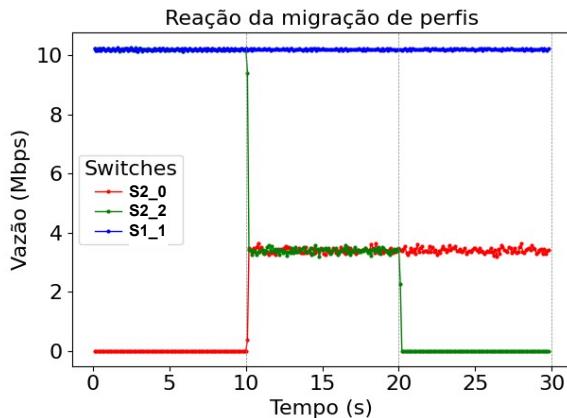
- **Métricas Avaliadas**

- Média de vazão registrada durante cada fase de teste.
- Estabilidade do fluxo durante as transições entre os perfis, sem flutuações ou interrupções.

Com esses objetivos e métricas, o Teste 2: Reconfiguração Ágil busca verificar a eficácia da abordagem de gerenciamento dinâmico de tráfego em relação às técnicas tradicionais, enfatizando a importância da adaptabilidade e da eficiência nas redes de *datacenters*.

O experimento consistiu na reconfiguração ágil entre os perfis de divisão de tráfego 0, 100 e 1011, já apresentados nas Figuras 29, 30 e 31. A divisão de tráfego foi determinada pela origem e, novamente, o *switch E1* é responsável por inserir os rótulos *routeID* e *weightID* no cabeçalho dos pacotes. Foram capturados os tráfegos nas portas ativas dos switches *S2_0*, *S2_2* e *S1_1*. Importante ressaltar que, durante o experimento, os valores de *routeID* e *weightID* variaram conforme o perfil selecionado. O teste realizado consiste em três fases: uso do Perfil 0 por 10 s, em seguida, migração para o Perfil 100 por mais 10 s, e nova migração para o Perfil 1011 por mais 10 s. Foram conduzidos 30 testes nos quais as médias de vazão foram registradas.

Figura 32 – Teste 2: Resultado



Fonte: Elaboração própria

Durante a realização dos 30 testes conduzidos, foram registradas as médias de vazão dos resultados. Essas médias são apresentadas na Figura 4. Ficou evidente que o fluxo permaneceu estável durante as transições entre os perfis, conforme ilustrado na Figura 32. A vazão obtida nas portas ativas dos *switches* durante o experimento estão representadas na Figura 32, evidenciando um fluxo estável e constante durante as transições entre os perfis, sem flutuações ou interrupções.

Nossa abordagem explora rótulos nos cabeçalhos dos pacotes (*weightID* e *routeID*) e o uso de tabelas estáticas nos *switches*, permitindo selecionar os perfis de divisão de tráfego com base na origem, com flexibilidade e granularidade na distribuição de tráfego. Como resultado, houve uma notável melhoria no desempenho e eficiência das redes de data.

Dessa forma, a partir dos resultados da Figura 32, pode-se afirmar que, com o *MTS-PolKA*, a divisão de tráfego pode ser reconfigurada rapidamente apenas definindo novos rótulos na borda, sem necessidade de atualizar as tabelas dos nós de núcleo.

Em abordagens baseadas em tabelas, há uma latência grande para configuração e convergência, já que o controlador precisa aplicar as alterações a todos os nós afetados ao longo do caminho (JYOTHI; DONG; GODFREY, 2015).

Comparando com o *WCMP*, que replica entradas de tabela com *IDs* de caminho baseados em pesos de caminho, nossa abordagem é vantajosa, pois o *WCMP* enfrenta atualizações lentas e sobrecarga adicional. Trabalhos como *CONGA*, *HULA* e *Contra* atualizam o estado do plano de dados para direcionar o tráfego com base nas condições de tráfego, mas estão limitados a considerar um único caminho melhor de cada vez. O método *DASH* cria e mantém um *pipeline* sensível à carga para a divisão de tráfego, lidando com situações complexas de tráfego, mas cada etapa desse *pipeline* exige cálculos e operações específicas, como cálculos de fronteiras de *hash* e armazenamento nas memórias locais, resultando em desenvolvimento e gerenciamento detalhados.

Como resultado, demonstramos elevada granularidade e flexibilidade para ajustar a divisão de tráfego em tempo real, sem a necessidade de reconfigurações complexas nos *switches*, resultando em melhor desempenho e eficiência nas redes de *datacenters* com múltiplos caminhos redundantes.

5.2.3 Teste 3: Múltiplos fluxos e concorrência

- **Objetivo do Teste :**

- Demonstrar a eficácia de múltiplos fluxos e a migração ágil de perfis de tráfego para otimizar a utilização de recursos da rede em cenários de concorrência. O teste visou observar como a reconfiguração dinâmica da divisão de tráfego pode melhorar a eficiência da largura de banda e minimizar o congestionamento, ao mesmo tempo em que se avalia o impacto do uso de múltiplos caminhos nas vazões dos fluxos de dados.

- **Métricas Avaliadas**

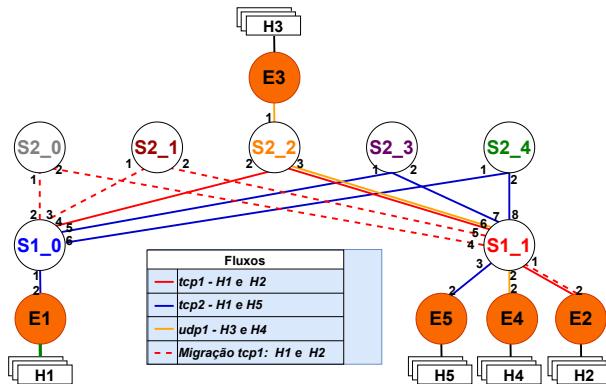
- **Vazão do Fluxo TCP1:** Mediu a capacidade de transmissão do fluxo *TCP1* antes

e após a migração de caminho e perfil de tráfego.

- **Vazão do Fluxo TCP2:** Avaliou a performance do fluxo *TCP2* durante o experimento.
- **Vazão do Fluxo UDP1:** Acompanhar a performance do fluxo *UDP1* que competia pelo mesmo recurso no enlace.
- **Vazão Agregada no Host H1:** Analisou o aumento da largura de banda total disponível ao *host* gerador de tráfego (*H1*) após a migração de fluxo, refletindo a eficiência da estratégia de divisão de tráfego.

Com esses objetivos e métricas, o Teste 3: Múltiplos Fluxos e Concorrência busca demonstrar a eficácia da migração ágil de perfis de tráfego e da utilização de múltiplos caminhos para otimizar o desempenho das redes em cenários de concorrência. Este teste ressalta a importância da flexibilidade na alocação de recursos e a adaptação dinâmica às condições de tráfego, evidenciando como essas estratégias podem melhorar significativamente a eficiência e a largura de banda nas redes.

Figura 33 – Teste 3: Cenário



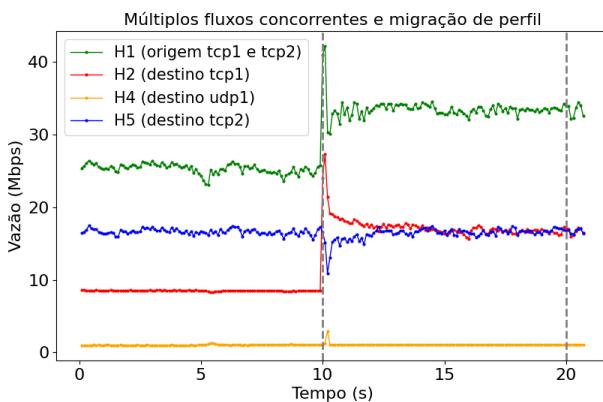
Fonte: Elaboração própria

Neste experimento, mostramos o funcionamento de múltiplos fluxos e a migração ágil de perfil de tráfego para otimizar o uso de recursos com concorrência. Foram enviados: um fluxo *TCP* entre *H1* e *H2* (*tcp1*, em vermelho), um fluxo *TCP* entre *H1* e *H5* (*tcp2*, em azul) e um fluxo *UDP* com vazão de 4Mbps entre *H3* e *H4* (*udp1*, em amarelo), conforme Figura 33. A divisão de tráfego foi determinada pelos *switches* de borda *E1* e *E3*, que são responsáveis por inserir os rótulos *routeID* e *weightID* no cabeçalho dos pacotes. São usados dois perfis de divisão de tráfego: perfil 0 (envia todo tráfego para a primeira porta ativa) e perfil 1 (pesos 1:1, envia metade do tráfego para a primeira porta ativa e metade do tráfego para a segunda porta ativa). A escolha de uma janela *tcp* de 1 M em um experimento com um núcleo de 10 M pode ser estratégica para equilibrar a eficiência da largura de banda, evitando congestionamentos e perda de pacotes. Essa configuração busca

otimizar o desempenho, adaptando-se à largura de banda efetiva da rede e minimizando a chance de problemas associados a janelas muito grandes.

Foram capturados os tráfegos nas portas ativas dos *switches* de origem (*port_position = 1*) (*s1_0es2_2*) e destino (*S1_1*). O teste consiste em duas fases: No período de **0 s a 10 s**, o fluxo *tcp1* seguiu um único caminho (*S1_0 – S2_2 – S1_1*) com perfil de tráfego 0, o fluxo *tcp2* usou o perfil de tráfego 1, dividindo o tráfego igualmente entre dois caminhos (*S1_0 – S2_3 – S1_1* e *S1_0 – S2_4 – S1_1*), e o fluxo *udp1* seguiu um único caminho (*S2_2 – S1_1*) com perfil de tráfego 0. Como o fluxo *tcp1* inicia o experimento concorrendo com o fluxo *udp1* no enlace *S2_2 – S1_1* e existem caminhos com enlaces ociosos, o experimento executou uma migração de caminho e perfil de tráfego para o fluxo *tcp1*. No período de **10 s a 20 s**, o fluxo *tcp1* foi migrado, apenas alterando a regra de fluxo que insere os rótulos dos pacotes referentes ao fluxo *tcp1* em *E1*, e passou a usar o perfil de tráfego 1 com a nova árvore de transmissão, dividindo o tráfego igualmente entre os dois caminhos que estavam ociosos (*S1_0 – S2_0 – S1_1* e *S1_0 – S2_1 – S1_1*).

Figura 34 – Teste 3: Resultado.



Fonte: Elaboração própria

As vazões obtidas durante o experimento estão representadas na Figura 34 e mostram o aumento da vazão atingida pelo fluxo *tcp1* (linha vermelha) após a migração eliminar a concorrência deste fluxo com o fluxo *udp1* e passar a explorar dois caminhos ao invés de um único caminho. Além disso, é possível perceber o aumento da vazão agregada no *host H1* (linha verde), que gera o tráfego, devido ao aumento da banda total disponível com a exploração de multicaminhos. Dessa forma, pode-se afirmar que, com o *MTS-PolKA*, a divisão de tráfego pode ser facilmente reconfigurada definindo novos rótulos na borda, com o funcionamento de múltiplos fluxos. Ainda, é possível explorar essa funcionalidade de migração ágil de perfil de tráfego para agregação de múltiplos fluxos *TCP* com o objetivo de otimizar o uso dos múltiplos caminhos com concorrência.

5.2.4 Teste 4: Comparação com Trabalhos Relacionados sobre o Tempo de Conclusão de Fluxo para Fluxos Elefante e Análise de Latência para Fluxos Rato

- **Objetivo do Teste :**

- Comparar o desempenho da solução proposta com abordagens existentes, focando em:
 - * Tempo de Conclusão de Fluxo (FCT) e Retransmissões TCP para fluxos elefante.
 - * Análise de latência para fluxos rato.

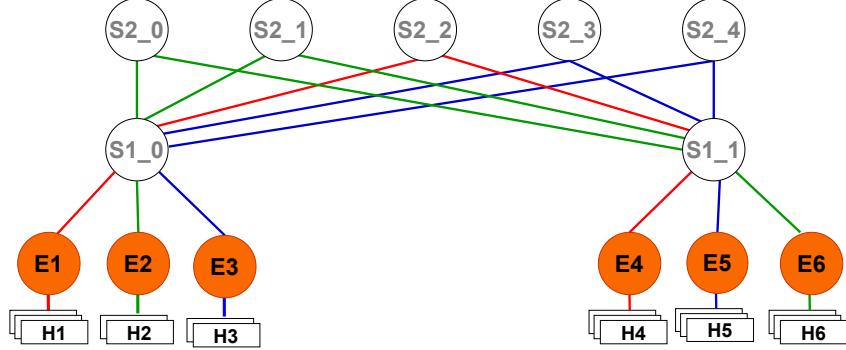
- **Métricas Avaliadas**

- **Tempo de Conclusão de Fluxo (FCT):** Essa métrica foi utilizada para avaliar como diferentes métodos de distribuição de tráfego afetam o tempo total que um fluxo de dados leva para ser concluído na rede.
- **Latência:** A latência média dos caminhos usados para transmitir pequenos fluxos de dados (fluxos de rato) foi medida para entender como a distribuição de tráfego impacta o tempo de resposta.
- **Retransmissões TCP:** Contagem de retransmissões durante a transmissão dos fluxos, que indicam a eficiência do método de encaminhamento utilizado.

Neste experimento, implementamos *ECMP*, *WCMP* e *RPS* usando a linguagem P4 dentro do emulador *Mininet* para realizar um estudo comparativo. Para os fluxos elefante, focamos na métrica de Tempo de Conclusão de Fluxo (FCT), que mede a duração necessária para que um fluxo de dados percorra a rede do origem ao destino até ser totalmente recebido. Essa métrica é crucial para avaliar o desempenho de rede em *datacenters* modernos. Para os fluxos rato, analisamos a latência durante a transmissão, já que esses fluxos são caracterizados por sua curta duração e sensibilidade a atrasos.

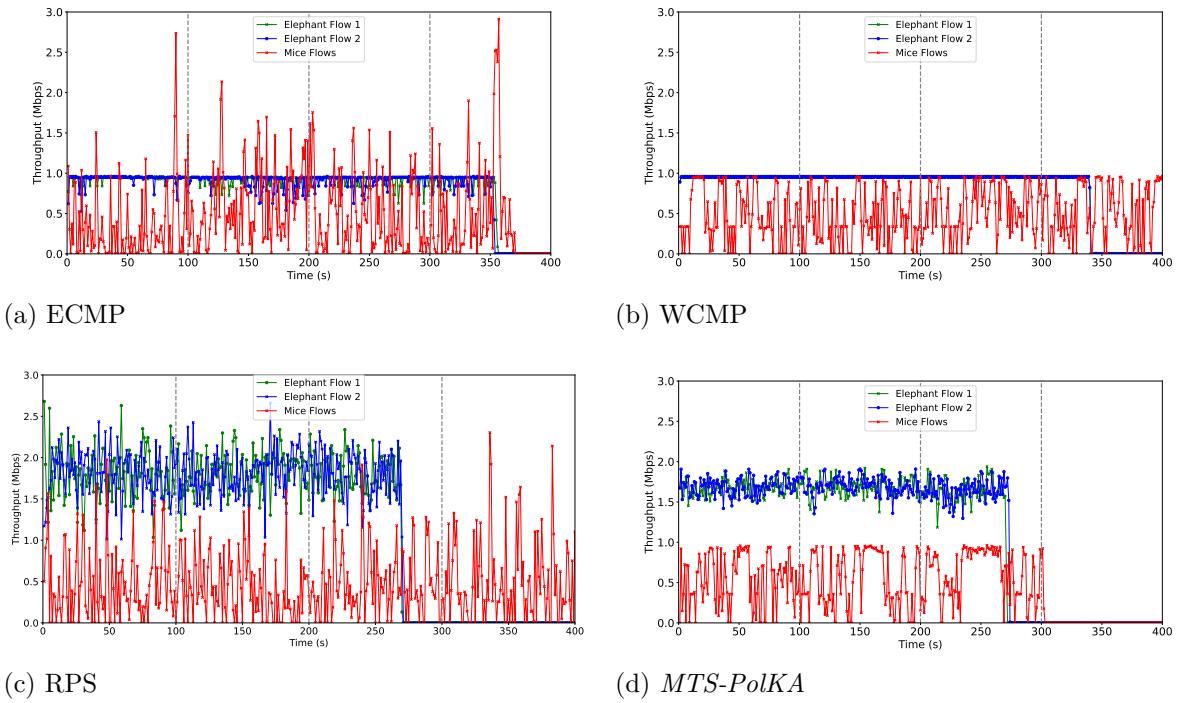
A Figura 35 mostra o cenário do teste. Um atraso de 1 ms foi introduzido na rede. Os links do núcleo da rede foram configurados para 1 Mbps, enquanto os links de borda foram configurados para 10 Mbps. Os seguintes fluxos de *TCP* foram enviados usando a ferramenta *iperf*: um fluxo de 100MB entre H2 e H6 (em verde), um fluxo de 100MB entre H3 e H5 (em azul), e vários fluxos de 10KB (4 fluxos por segundo seguindo uma distribuição de Poisson) entre H1 e H4 (em vermelho). O tráfego foi capturado nas máquinas de destino H4, H5 e H6, que estão conectadas ao switch *S1_1*. Além disso, dados de latência foram coletados usando a ferramenta *ping*.

Figura 35 – Teste 4: Cenário (*MTS-PolKA* configuração de divisão de tráfego).



Fonte: Elaboração própria

Figura 36 – Teste 4: Comparação do Tempo de Conclusão do Fluxo com trabalhos relacionados.

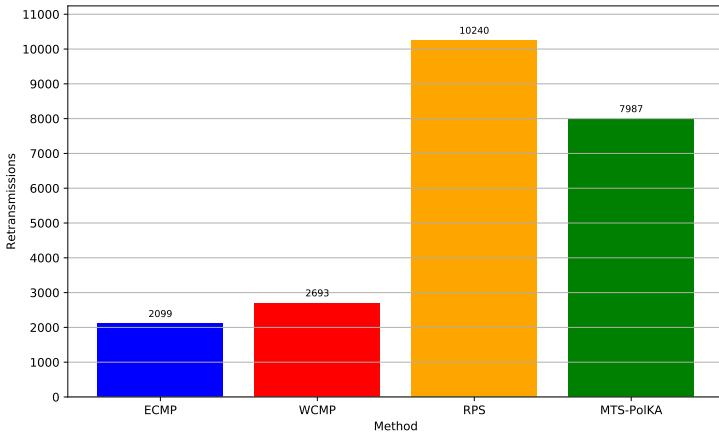


Fonte: Elaboração própria

A Figura 35 ilustra especificamente a configuração de divisão de tráfego selecionada por *MTS-PolKA*. Dois perfis de divisão de tráfego foram utilizados: o fluxo elefante 1 (em verde) e o fluxo elefante 2 (em azul) seguiram o perfil 1, que divide o tráfego igualmente entre as duas portas ativas, enquanto os fluxos de rato (em vermelho) seguiram o perfil 0, que envia todo o tráfego para uma única porta ativa. Para cada fluxo, os *switches* de borda E1, E2 e E3 são responsáveis por inserir os rótulos *routeID* e *weightID* nos cabeçalhos dos pacotes, de acordo com as portas ativas selecionadas e os perfis de divisão de tráfego.

Além disso, *ECMP*, *WCMP* e *RPS* foram implementados para distribuir os fluxos de elefante e de rato por múltiplos caminhos. Com o *ECMP*, cada fluxo é atribuído a um único caminho de custo igual, o que pode resultar em colisões de *hash* e uma distribuição uniforme que ignora as características específicas dos fluxos. O *RPS* distribui pacotes aleatoriamente

Figura 37 – Teste 4: Comparação com trabalhos relacionados à retransmissão de TCP para o fluxo elefante 1.



Fonte: Elaboração própria

entre os links disponíveis, sem considerar os tamanhos dos fluxos ou as capacidades dos links. Com o *WCMP*, cada fluxo é atribuído a um único caminho, mas a distribuição do tráfego entre esses caminhos pode ser ajustada usando tabelas de grupo ponderadas em cada *switch*. Consequentemente, a configuração ideal com o *WCMP* envolveu reservar um link para cada fluxo de elefante, enquanto se alocava um link exclusivo para os fluxos de rato.

Os resultados demonstraram que o *MTS-PolKA* foi o único que conseguiu beneficiar simultaneamente tanto os fluxos dos ratos quanto os dos elefantes, otimizando efetivamente o desempenho para ambos os tipos. A comparação de *FCT* na Figura 36 mostrou que tanto o *MTS-PolKA* quanto o *RPS* conseguiram reduzir significativamente o *FCT* dos fluxos de elefantes (aproximadamente 270s) em comparação com o *ECMP* (aproximadamente 355s) e o *WCMP* (aproximadamente 340s), pois eles permitem que os fluxos de elefantes explorem mais de um link.

Neste experimento, a distribuição de dois fluxos de elefante em dois links reservados para cada fluxo pelo *MTS-PolKA* resultou no mesmo tempo de conclusão de fluxo (*FCT*) que quando todos os fluxos foram espalhados por cinco links usando *RPS*. No entanto, a Figura 37 destaca uma diferença significativa nas retransmissões TCP para o fluxo de elefante 1, com o *RPS* mostrando uma contagem muito maior de 10.240 em comparação com os 7.987 do *MTS-PolKA*. Além disso, as técnicas de espalhamento de pacotes empregadas tanto pelo *MTS-PolKA* quanto pelo *RPS* levaram a um aumento considerável nas retransmissões TCP em comparação com o *ECMP* e *WCMP*. Para resolver esse problema, planejamos explorar, em trabalhos futuros, estratégias de gerenciamento de filas dentro dos algoritmos de escalonamento de fluxo para reduzir a reordenação de pacotes causada pelo espalhamento de pacotes através de vários caminhos. Especificamente, nosso objetivo é selecionar caminhos com base no atraso na fila do *buffer* de saída, priorizando o encaminhamento

de pacotes que chegam primeiro em relação aos que chegam depois. Essa abordagem busca minimizar a reordenação de pacotes, semelhante ao método utilizado pelo *QDAPS* (HUANG et al., 2018).

Ao mesmo tempo que melhora a utilização de largura de banda para fluxos de elefantes, o *MTS-PolKA* consegue otimizar simultaneamente a latência para fluxos de ratos. A Figura 38 mostra os resultados de latência dos caminhos para cada método. No ECMP, os fluxos pequenos podem ser distribuídos entre qualquer um dos cinco caminhos com base na saída da função de *hash*, o que pode fazer com que eles compartilhem um enlace de alta latência com um fluxo grande (caminhos 4 e 5 na Figura 38a, com latência média superior a 7000ms). A Figura 38b mostra que tanto *MTS-PolKA* quanto o WCMP são capazes de alocar fluxos pequenos em caminhos com latência mais baixa (latência média inferior a 150ms), pois eles separam os fluxos pequenos, que são sensíveis à latência, dos fluxos grandes. De acordo com (HU et al., 2018a), essa separação é vital para proteger os fluxos curtos de serem bloqueados pelos fluxos longos, já que os caminhos que transportam fluxos longos normalmente têm *RTTs* muito maiores do que aqueles com apenas fluxos curtos, o que impacta negativamente o *FCT* dos fluxos pequenos. Em contrapartida, enquanto o *WCMP* consegue essa separação e benefícios para os fluxos pequenos, ele leva a *FCTs* mais longos para os fluxos grandes porque restringe a divisão dos fluxos entre vários *links*.

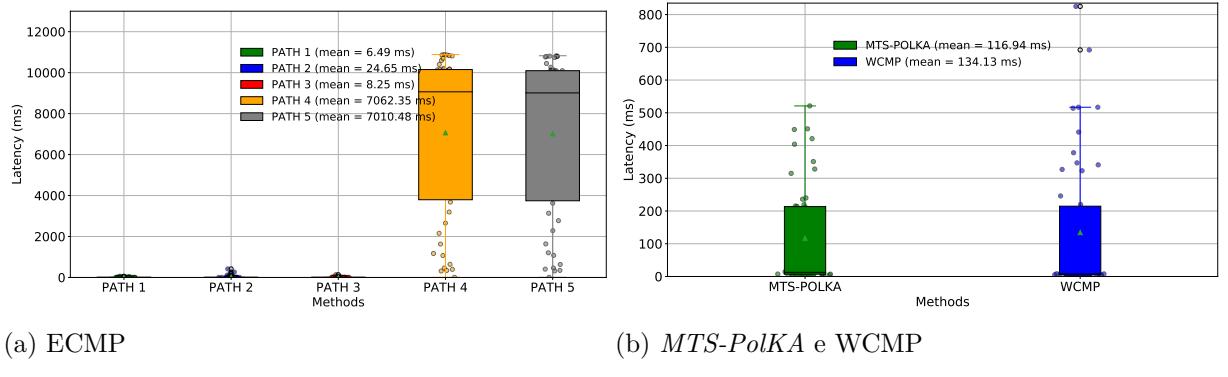
MTS-PolKA adota uma abordagem híbrida que combina granularidade de fluxo e de pacotes, junto com roteamento baseado na origem, permitindo otimizar dinamicamente o uso da largura de banda e minimizar a congestão da rede. Essa distribuição inteligente de tráfego alivia a congestão em rotas específicas, permitindo que grandes fluxos (fluxos de elefante) utilizem caminhos menos sobrecarregados, o que reduz o tempo de conclusão das transferências. Ao mesmo tempo, essa estratégia garante que fluxos menores (fluxos de rato) sejam transmitidos rapidamente, sem competir diretamente com fluxos maiores pelos recursos limitados da rede. Ao aproveitar múltiplos links, *MTS-PolKA* ajuda a alcançar um uso de largura de banda mais equilibrado, melhorando o desempenho da rede para todos os tipos de tráfego. Além disso, esse cenário de teste pode ser extrapolado para cargas de tráfego mais complexas, onde políticas por fluxo escolhidas pela origem podem ser selecionadas individualmente para otimizar o desempenho geral.

5.2.5 Limitações do protótipo

A solução apresentada neste trabalho provou ser viável e funcional em comparação aos métodos tradicionais. No entanto, existem algumas limitações que precisam ser destacadas, as quais serão o foco da pesquisa e desenvolvimento em trabalhos futuros:

- O *switch* de software `bmv2 simple_switch` com a arquitetura `v1model` foi selecionado como alvo para este protótipo, porque ele suporta todas as funcionalidades requeridas

Figura 38 – Teste 4: Comparação de latências com trabalhos relacionados.



(a) ECMP

(b) *MTS-PolKA* e *WCMP*

Fonte: Elaboração própria

por *MTS-PolKA*, como a configuração de polinômios CRC. É importante destacar que este *switch* de *software* é uma implementação de espaço do usuário com foco em testes de recursos. Existem outras implementações de alto desempenho de *switches* de *software* e compiladores P4 (por exemplo, PISCES⁷, P4ELTE⁸ e MACSAD⁹), mas eles ainda não cobrem todos os recursos requeridos por *MTS-PolKA*. À medida que essas implementações evoluem, será possível testar nosso protótipo com cargas maiores. Por enquanto, a solução foi limitar o número de fluxos e as taxas de link para 10 Mbps em nosso protótipo emulado para evitar atingir os limites de capacidade de processamento do `bmv2 simple_switch`.

- Devido às limitações descritas do *switch* de *software*, este trabalho não conseguiu realizar uma comparação mais profunda de desempenho e escalabilidade com trabalhos relacionados. Planejamos abordar isso em pesquisas futuras implementando nossa proposta e soluções relacionadas em dispositivos físicos, como o *switch Tofino*, e no simulador ns-3. Este último será particularmente importante para obter mais informações sobre problemas de reordenação de *TCP*.
- Este trabalho se concentrou em projetar o mecanismo de divisão de tráfego dentro do plano de dados. Outras funcionalidades importantes, como monitoramento, escalonamento de fluxo, enfileiramento e alocação de caminho, foram tratadas como entradas de plano de controle predefinidas nos experimentos e não estavam dentro do escopo deste trabalho. Esses aspectos serão abordados em pesquisas futuras.

⁷ <<http://pisces.cs.princeton.edu/>>

⁸ <<http://p4.elte.hu/>>

⁹ <<https://github.com/intrig-unicamp/macasad/>>

6 RESULTADOS E DISCUSSÕES

Foram conduzidos quatro experimentos. No primeiro, foram configurados três perfis de tráfego (0, 4 e 11) em um *switch* de núcleo (S1_0). O Perfil 0 roteou todo o tráfego pela primeira porta ativa, o Perfil 4 dividiu o tráfego igualmente entre três portas ativas, e o Perfil 11 distribuiu o tráfego com proporções 2:1:2:1 em quatro portas ativas. Os resultados mostraram a distribuição de pacotes conforme os perfis selecionados de forma consistente e eficaz.

No segundo experimento, foi testada a capacidade de reconfiguração ágil dos perfis de tráfego, mudando rapidamente de 0 para 4 e depois para 11. Os perfis foram aplicados na origem, inserindo rótulos *routeID* e *weightID* nos pacotes. A vazão permaneceu estável durante as transições, demonstrando eficiência na reconfiguração ágil sem a necessidade de atualizações complexas nos *switches* de núcleo.

No terceiro experimento, múltiplos fluxos (*TCP* e *UDP*) foram enviados através de diversos caminhos em uma rede emulada, e a migração de perfis de tráfego foi testada para otimizar o uso de recursos. A migração aumentou a vazão dos fluxos *TCP*, demonstrando a capacidade de reconfiguração dinâmica para melhorar a eficiência de tráfego e o desempenho geral da rede.

No quarto experimento, compararam-se os métodos *ECMP*, *WCMP* e *RPS* na distribuição de fluxos de elefante e de rato. O *ECMP* atribui cada fluxo a um único caminho de custo igual, o que pode gerar colisões de *hash*; o *RPS* distribui pacotes aleatoriamente; e o *WCMP* ajusta a distribuição de tráfego utilizando tabelas ponderadas, reservando *links* específicos para diferentes tipos de fluxo. Os resultados mostraram que *MTS-PolKA* foi a única solução capaz de otimizar o desempenho para ambos os tipos de fluxo, reduzindo o Tempo de Conclusão de Fluxo (FCT) dos fluxos de elefante para cerca de 270 segundos, em comparação com 355 segundos do *ECMP* e 340 segundos do *WCMP*. Assim, *MTS-PolKA* se destacou na otimização da largura de banda e na mitigação de congestionamentos.

Em resumo, os experimentos realizados demonstraram a eficácia do sistema de roteamento multi-caminho *MTS-PolKA* na distribuição de tráfego em redes de *datacenter*. A capacidade de reconfiguração ágil e a otimização dinâmica do uso de recursos não apenas melhoraram a vazão dos fluxos *TCP*, mas também garantiram um desempenho superior em comparação com métodos tradicionais como *ECMP* e *WCMP*. Os resultados ressaltam o potencial do *MTS-PolKA* para atender às crescentes demandas por eficiência em ambientes de rede complexos, fornecendo soluções inovadoras que asseguram uma gestão de tráfego mais equilibrada e eficaz. A continuidade das pesquisas nesta área poderá contribuir para o desenvolvimento de redes ainda mais resilientes e adaptáveis.

7 CONCLUSOES E TRABALHOS FUTUROS

A abordagem *MTS-PoLK*A para otimização de tráfego em redes de *datacenter* representa uma contribuição significativa ao campo. Utilizando um sistema de resíduos numéricos para roteamento de fonte sem necessidade de estado, o *MTS-PoLK*A supera as abordagens tradicionais ao distribuir o tráfego de forma eficiente e flexível em todos os *switches* do caminho, sem configurações complexas.

Os testes realizados com o emulador *Mininet* demonstraram que a *MTS-PoLK*A explora eficazmente múltiplos caminhos na rede, garantindo a estabilidade dos fluxos e otimizando o uso de caminhos redundantes. Um dos principais destaques dessa solução é a facilidade de alterar os perfis de divisão de tráfego sem impactar a configuração dos *switches*, o que confere uma vantagem prática sobre métodos tradicionais como *ECMP* e *WCMP*. Além disso, o *MTS-PoLK*A reduz significativamente o número de estados da rede, aumentando sua eficiência e simplicidade na divisão de tráfego multicaminho.

Abordagens como *CONGA*, *HULA*, *DASH* e métodos baseados em *SDN*, apesar de eficazes, apresentam limitações quanto ao uso de múltiplos caminhos ou à escalabilidade, devido ao aumento do número de tabelas de fluxo. Alternativas como *MP-TCP + Source Routing (SR)* requerem configurações específicas nos hosts finais, enquanto métodos de pulverização de pacotes, como *RPS* e *QDAPS*, enfrentam problemas de reordenação de pacotes e não permitem políticas específicas para as diferentes classes de fluxo. O *MTS-PoLK*A se destaca por superar essas limitações ao combinar granularidade por fluxo e pacote com roteamento na fonte, permitindo ajustes ágeis por meio de pequenas alterações no cabeçalho dos pacotes.

Experimentos com *switches* programáveis P4 evidenciaram que a *MTS-PoLK*A melhora a taxa de transferência agregada, mantendo a estabilidade dos fluxos durante mudanças de perfil. Isso demonstra seu potencial como uma solução eficaz e escalável para a engenharia de tráfego em redes de *datacenter*.

O próximo passo envolve estender a avaliação do *MTS-PoLK*A, comparando seu desempenho com outras soluções, especialmente em termos de tempo de conclusão de fluxo, escalabilidade, sobrecarga de memória e uso de largura de banda. Experimentos adicionais serão realizados em *switches* programáveis *Tofino®* para validar a eficácia da solução em ambientes físicos reais.

Portanto, o *MTS-PoLK*A surge como uma solução promissora para a otimização de tráfego em redes de *datacenter*, oferecendo uma alternativa eficaz às abordagens tradicionais, com maior adaptabilidade, simplicidade e eficiência na utilização de recursos multicaminho.

REFERÊNCIAS

- BENSON, Theophilus et al. Understanding data center traffic characteristics. *SIGCOMM Comput. Commun. Rev.*, Association for Computing Machinery, New York, NY, USA, v. 40, n. 1, p. 92–99, jan 2010. ISSN 0146-4833. Disponível em: <<https://doi.org/10.1145/1672308.1672325>>.
- BOSSHART, Pat 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. Disponível em: <<https://doi.org/10.1145/2656877.2656890>>.
- CASTANHEIRA, Lucas; PARIZOTTO, Ricardo; SCHAEFFER-FILHO, Alberto E. Flowstalker: Comprehensive traffic flow monitoring on the data plane using p4. In: *ICC 2019 - 2019 IEEE International Conference on Communications (ICC)*. [S.l.: s.n.], 2019. p. 1–6.
- COSTA, Kevin Barros et al. Enhancing orchestration and infrastructure programmability in sdn with notoriety. *IEEE Access*, v. 8, p. 195487–195502, 2020.
- CUI, Zixi; HU, Yuxiang; HOU, Saifeng. Adaptive weighted cost multipath routing on pisa. In: *2021 IEEE International Conference on Artificial Intelligence and Industrial Design (AIID)*. [S.l.: s.n.], 2021. p. 541–544.
- DIXIT, Advait; PRAKASH et al. On the impact of packet spraying in data center networks. In: *2013 Proceedings IEEE INFOCOM*. [S.l.: s.n.], 2013. p. 2130–2138.
- DOMINICINI, Cristina et al. Deploying polka source routing in p4 switches : (invited paper). In: *2021 International Conference on Optical Network Design and Modeling (ONDM)*. [S.l.: s.n.], 2021. p. 1–3.
- DOMINICINI, Cristina et al. Polka: Polynomial key-based architecture for source routing in network fabrics. In: *2020 6th IEEE Conference on Network Softwarization (NetSoft)*. [S.l.: s.n.], 2020. p. 326–334.
- DOMINICINI, Cristina Klippel et al. Programmable, expressive, scalable, and agile service function chaining for edge data centers. In: SBC. *Anais Estendidos do XXXVIII Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos*. [S.I.], 2020. p. 177–184.
- GEREMEW, Getahun Wassie; DING, Jianguo. Elephant flows detection using deep neural network, convolutional neural network, long short-term memory, and autoencoder. *Journal of Computer Networks and Communications*, v. 2023, n. 1, p. 1495642, 2023. Disponível em: <<https://onlinelibrary.wiley.com/doi/abs/10.1155/2023/1495642>>.
- GILLIARD, Ezekia et al. Intelligent load balancing in data center software-defined networks. *Transactions on Emerging Telecommunications Technologies*, v. 35, n. 4, p. e4967, 2024. Disponível em: <<https://onlinelibrary.wiley.com/doi/abs/10.1002/ett.4967>>.
- GOPAKUMAR, R; UNNI, A M; DHIPIN, V P. An adaptive algorithm for searching in flow tables of openflow switches. In: *2015 39th National Systems Conference (NSC)*. [S.l.: s.n.], 2015. p. 1–5.

GUIMARÃES, Rafael S. et al. M-polka: Multipath polynomial key-based source routing for reliable communications. *IEEE Transactions on Network and Service Management*, v. 19, n. 3, p. 2639–2651, 2022.

HAMDAN, Mosab et al. A comprehensive survey of load balancing techniques in software-defined network. *Journal of Network and Computer Applications*, v. 174, p. 102856, 2021. ISSN 1084-8045. Disponível em: <<https://www.sciencedirect.com/science/article/pii/S1084804520303222>>.

HSU, Kuo-Feng; TAMMANA et al. Adaptive weighted traffic splitting in programmable data planes. In: *Proceedings of the Symposium on SDN Research*. New York, NY, USA: Association for Computing Machinery, 2020. (SOSR '20), p. 103–109. ISBN 9781450371018. Disponível em: <<https://doi.org/10.1145/3373360.3380841>>.

HU, Jinbin et al. Caps: Coding-based adaptive packet spraying to reduce flow completion time in data center. In: *IEEE INFOCOM 2018 - IEEE Conference on Computer Communications*. [S.l.: s.n.], 2018. p. 2294–2302.

HU, Tao et al. Multi-controller based software-defined networking: A survey. *IEEE Access*, v. 6, p. 15980–15996, 2018.

HUANG, Jiawei et al. Qdaps: Queueing delay aware packet spraying for load balancing in data center. In: *2018 IEEE 26th International Conference on Network Protocols (ICNP)*. [S.l.: s.n.], 2018. p. 66–76.

HUANG, Jiawei; LYU et al. Mitigating packet reordering for random packet spraying in data center networks. *IEEE/ACM Transactions on Networking*, v. 29, n. 3, p. 1183–1196, 2021.

JYOTHI, Sangeetha Abdu; DONG, Mo; GODFREY, P Brighten. Towards a flexible data center fabric with source routing. In: *Proceedings of the 1st ACM SIGCOMM Symposium on Software Defined Networking Research*. [S.l.: s.n.], 2015. p. 1–8.

KOPONEN, Teemu et al. Onix: A distributed control platform for large-scale production networks. In: *9th USENIX Symposium on Operating Systems Design and Implementation (OSDI 10)*. [S.l.: s.n.], 2010.

LI, Weihe et al. Survey on traffic management in data center network: from link layer to application layer. *IEEE Access*, IEEE, v. 9, p. 38427–38456, 2021.

MARTINELLO, Magnos et al. Keyflow: a prototype for evolving sdn toward core network fabrics. *IEEE Network*, v. 28, n. 2, p. 12–19, 2014.

NKOSI, M. C.; LYSKO, A. A.; DLAMINI, S. Multi-path load balancing for sdn data plane. In: *2018 International Conference on Intelligent and Innovative Computing Applications (ICONIC)*. [S.l.: s.n.], 2018. p. 1–6.

P4.org. *P4 Language Specification*. 2021. Disponível em: <<https://p4.org/p4-spec/docs/P4-16-v1.2.2.html>>. Acesso em: 30 ago. 2021.

PANG, Junjie; XU, Gaochao; FU, Xiaodong. Sdn-based data center networking with collaboration of multipath tcp and segment routing. *IEEE Access*, v. 5, p. 9764–9773, 2017.

- PAOLUCCI, F. et al. P4 edge node enabling stateful traffic engineering and cyber security. *Journal of Optical Communications and Networking*, v. 11, n. 1, p. A84–A95, 2019.
- ROBIN, Debobroto Das; KHAN, Javed I. Clb: Coarse-grained precision traffic-aware weighted cost multipath load balancing on pisa. *IEEE Transactions on Network and Service Management*, v. 19, n. 2, p. 784–803, 2022.
- ROTTENSTREICH, Ori; KANIZO et al. Accurate traffic splitting on commodity switches. In: *Proceedings of the 30th on Symposium on Parallelism in Algorithms and Architectures*. New York, NY, USA: Association for Computing Machinery, 2018. (SPAA '18), p. 311–320. ISBN 9781450357999. Disponível em: <<https://doi.org/10.1145/3210377.3210412>>.
- SANTOS, Giancarlo et al. Mts-polka: Divisão de tráfego multicaminhos em proporção de peso com roteamento na fonte. In: *Anais do XLII Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos*. Porto Alegre, RS, Brasil: SBC, 2024. p. 71–84. ISSN 2177-9384. Disponível em: <<https://sol.sbc.org.br/index.php/sbrc/article/view/29784>>.
- VALENTIM, Rodolfo et al. Rdna balance: Balanceamento de carga por isolamento de fluxos elefante em data centers com roteamento na origem. In: SBC. *Anais do XXXVII Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos*. [S.l.], 2019. p. 1000–1013.
- VALENTIM, Rodolfo et al. Rdna balance: Balanceamento de carga por isolamento de fluxos elefante em data centers com roteamento na origem. In: *Anais do XXXVII Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos*. Porto Alegre, RS, Brasil: SBC, 2019. p. 1000–1013. ISSN 2177-9384. Disponível em: <<https://sol.sbc.org.br/index.php/sbrc/article/view/7418>>.
- WEI, Jie et al. Minimizing data transmission latency by bipartite graph in mapreduce. In: *2015 IEEE International Conference on Cluster Computing*. [S.l.: s.n.], 2015. p. 521–522.
- XIE, Shengxu et al. Online elephant flow prediction for load balancing in programmable switch-based dcn. *IEEE Transactions on Network and Service Management*, v. 21, n. 1, p. 745–758, 2024.
- YU, Jinke et al. A load balancing mechanism for multiple sdn controllers based on load informing strategy. In: *2016 18th Asia-Pacific Network Operations and Management Symposium (APNOMS)*. [S.l.: s.n.], 2016. p. 1–4.
- ZHOU, Junlan; TEWARI et al. Wcmp: Weighted cost multipathing for improved fairness in data centers. In: *Proceedings of the Ninth European Conference on Computer Systems*. New York, NY, USA: Association for Computing Machinery, 2014. (EuroSys '14). ISBN 9781450327046. Disponível em: <<https://doi.org/10.1145/2592798.2592803>>.
- ZHOU, Junlan et al. Wcmp: Weighted cost multipathing for improved fairness in data centers. In: *Proceedings of the Ninth European Conference on Computer Systems*. [S.l.: s.n.], 2014. p. 1–14.