Universidade Federal de São Carlos– UFSCar
Centro de Ciências Exatas e de Tecnologia– CCET
Departamento de Computação– DC
Programa de Pós-Graduação em Ciência da Computação– PPGCC

Thiago Henrique Silva Rodrigues

Detectando Heavy Hitters globalmente em dispositivos programáveis multi-pipes

São Carlos
2024

Thiago Henrique Silva Rodrigues

Detectando Heavy Hitters globalmente em dispositivos programáveis multi-pipes

Dissertação apresentada ao Programa de Pós-Graduação em Ciência da Computação do Centro de Ciências Exatas e de Tecnologia da Universidade Federal de São Carlos, como parte dos requisitos para a obtenção do título de Mestre em Ciência da Computação.

Área de concentração: Sistemas Distribuídos, Arquiteturas e Redes de Computadores

Orientador: Fábio Luciano Verdi

São Carlos
2024

Este trabalho é dedicado a Deus.
Que sempre me ajudou nas horas difíceis, sem ele nada seria possível.

Agradecimentos

Agradeço, primeiramente, a Deus por ter me concedido o privilégio de chegar até aqui, dando-me forças e discernimento para superar todos obstáculos, e capacitando-me com essa experiência única.

Expresso meu agradecimento especial aos meus pais, Edimilson e Elsa, ao meu irmão André e à minha namorada Marcella. Seu apoio incondicional e compreensão em cada momento desta jornada foram fundamentais para que este sonho se tornasse realidade. Com eles, compartilhei as alegrias e enfrentei as dificuldades deste percurso. Também estendo esse agradecimento a todos meu amigos, que me apoiaram e fizeram parte dessa trajetória.

Expresso minha gratidão ao meu orientador, professor Dr. Fábio Luciano Verdi, pela orientação, pelos valiosos conhecimentos compartilhados, principalmente por acreditar em mim e ter compreendido minhas dificuldades, auxiliando-me na realização deste trabalho.

Agradeço a toda equipe do LERIS (Laboratório de Estudos em Redes, Inovação e Software), pelo apoio incansável e contribuição significativa para a realização deste trabalho.

Agradeço aos professores do Programa de Pós Graduação em Ciência da Computação da Universidade Federal de São Carlos (UFSCar), cujas contribuições foram fundamentais para minha formação.

Sou grato à Coordenação de Aperfeiçoamento de Pessoal de Nível Superior (CAPES) pela concessão de bolsa de estudos.

Por fim, agradeço a todos que, de forma direta ou indireta, contribuíram para que eu obtivesse o título de mestre em Ciência da Computação.

"Somos essencialmente profissionais do sentido. Educamos, quando ensinamos com sentido. Educar é impregnar de sentido a vida. A profissão docente está centrada na vida, no bem querer."
(Prof. Gilberto Teixeira)

Resumo

Mecanismos de monitoramento de alta precisão desempenham um papel fundamental na gestão de diversas tarefas de rede, tais como controle de congestionamento, detecção de anomalias, balanceamento de carga, entre outros. Uma forma de contribuir para a gestão de redes é aplicar uma detecção de fluxos que exercem maior influência no tráfego, caracterizados como *Heavy Hitter* (HH). Fluxos HH correspondem a fluxos que conduzem a maior parcela de *bytes* transmitidos pela rede, consequentemente consumindo mais recursos. Através da utilização de *hardware* programável (*Switches, SmartNICs e DPU's,*) em conjunto com linguagens de programação como P4, é possível detectar esses fluxos *in-line rate*. Isto é, uma detecção diretamente no plano de dados da rede. A literatura revela que a detecção de fluxos HH representa um tópico amplamente explorado. No entanto, estudos que propõem soluções para a detecção de HH erroneamente assumem que o *switch* possui apenas um único *pipe*. Contudo, *switches* programáveis possuem múltiplos *pipes*, variando de 2 a 16. Diante desse panorama, este estudo apresenta uma abordagem para identificar fluxos HH em *switches* programáveis com múltiplos *pipes*. O desenvolvimento contempla duas abordagens para detecção. Na primeira abordagem, um acumulador localizado no *switch* centraliza dados provenientes de todos os *pipes* e se comunica com o plano de controle. Na segunda abordagem, as comunicações com o plano de controle são independentes para cada *pipe*. Nossas abordagens foram validadas por meio de um emulador, o que nos permitiu adquirir resultados preliminares, indicando a eficácia e melhoria na detecção de HH em *switches multi-pipes*, em comparação com a detecção em *switches* operando com um único *pipe*.

Palavras-chave: Monitoramento de rede, Detecção de *Heavy Hitter*, *Network-wide*, *Switch Multi-Pipe*.

Abstract

High-precision monitoring mechanisms play a fundamental role in managing various network tasks, such as congestion control, anomaly detection, load balancing, among others. One way to contribute to network management is by applying a detection of flows that have a greater influence on traffic, characterized as HH. HH flows correspond to flows that account for the largest share of transmitted bytes across the network, consequently consuming more resources. By utilizing programmable hardware (*Switches, SmartNICs, and DPUs*) in conjunction with programming languages like P4, it is possible to detect these flows at in-line rate, meaning detection directly in the network's data plane. The literature reveals that HH flow detection is a widely explored topic. However, studies proposing solutions for HH detection erroneously assume that the *switch* has only a single *pipe*. In reality, programmable *switches* have multiple *pipes*, ranging from 2 to 16. In light of this, this study presents an approach to identifying HH flows in programmable *switches* with multiple *pipes*. The development encompasses two detection approaches. In the first approach, an accumulator located in the *switch* centralizes data from all *pipes* and communicates with the control plane. In the second approach, communications with the control plane are independent for each *pipe*. Our approaches were validated through an emulator, which allowed us to obtain preliminary results indicating the effectiveness and improvement in HH detection in *multi-pipe switches*, compared to detection in *switches* operating with a single *pipe*.

Keywords: Network Monitoring, Heavy Hitter Detection, Network-wide, Multi-pipe Switch.

Lista de ilustrações

Lista de tabelas

Lista de siglas

API *Application Programming Interface*

ASIC *Application Specific Integrated Circuits*

CDM *Continuous Distributed Monitoring*

DDoS *Distributed Denial of Service*

EWMA *Exponentially Weighted Moving Average*

HH *Heavy Hitter*

ISP *Internet Service Provider*

OF *OpenFlow*

P4 *Programming Protocol-Independent Packet Processor*

SDN *Software Defined Network*

WAP *Wireless Access Point*

Sumário

Capítulo 1

Introdução

O processo de monitoramento de rede é uma ação para operações e identificações de comportamentos e anomalias, que visam identificar eventos capazes de comprometer o funcionamento e o desempenho, como por exemplo: a engenharia de tráfego, a detecção de Ataques de Negação de Serviço Distribuídos (*Distributed Denial of Service* (DDoS)), falhas de configuração em políticas de roteamento, etc. (DING et al., 2020; CRUZ et al., 2014). Operadores de rede frequentemente monitoram o tráfego em busca de otimizar o desempenho, identificar fluxos maliciosos, sobrecarga, ataques, fluxos pesados, entre outros (HARRISON et al., 2020; HOFSTEDE et al., 2014).

Neste contexto, uma prática comum é a identificação dos *Heavy Hitters* (HHs). São considerados *Heavy Hitters* os fluxos que ultrapassam uma quantidade estabelecida de *bytes* ou pacotes. Esses fluxos representam menos de 10% dos fluxos que passam em uma rede de *datacenter*, porém, consomem a maior parte dos recursos. O monitoramento desse tipo de fluxo pode aprimorar, facilitar e contribuir para uma gestão diária eficiente da rede. Além disso, a identificação oferece benefícios significativos aos administradores de rede, contribuindo para uma gestão eficiente e otimizada (LIN; HUANG; TSAI, 2019; VILELA, 2006).

Estudos indicam que mais de 80% dos fluxos têm uma duração média menor do que 11 segundos, conseguindo transportar apenas alguns pacotes (menos que 10 KB de dados); enquanto que cerca de 0,1% tem a duração maior do que 200 segundos (TURKOVIC; OOSTENBRINK; KUIPERS, 2019). Embora os HHs sejam uma minoria, as características do tráfego associado aos fluxos HHs têm o potencial de esgotar rapidamente os recursos dos dispositivos de rede (SILVA, 2019). Portanto, torna-se essencial distinguir e agir prontamente a esse fluxo, de preferência em um curto período de tempo, para que medidas de gerência possam ser aplicadas em tempo hábil, observando as políticas de

tráfego da rede. Caso contrário, a não detecção poderá comprometer de forma negativa o tráfego de fluxos leves que compartilham a mesma rede (SILVA, 2019; CHIESA; VERDI, 2023).

Um exemplo prático dessa necessidade pode ser observado na mitigação de DDoS em redes corporativas ou de *Internet Service Provider* (ISP). Esses ataques são enviados por duas ou mais fontes em uma tentativa de tornar um recurso de máquina ou rede indisponível para seus usuários designados. O tráfego massivo gerado por essas múltiplas fontes sobrecarrega o alvo, resultando em interrupções significativas nos serviços. Ao utilizar algoritmos de detecção de HH, é possível identificar rapidamente fluxos que excedem os limites predefinidos, sinalizando um possível ataque DDoS (CALLEGARI et al., 2011; YAN et al., 2015).

A identificação antecipada dos HHs possibilita aos administradores de rede implementar estratégias específicas, como a otimização do roteamento do tráfego, a aplicação de filtros para restringir o tráfego de IPs suspeitos ou a priorização desses fluxos, garantindo uma alocação adequada de largura de banda para manter o desempenho desejado. Além disso, podem ser realizadas medidas proativas para evitar congestionamentos, seja redirecionando ou otimizando o roteamento. Os administradores podem utilizar a detecção como parte de estratégias de segurança, monitorando de perto esses fluxos para detectar possíveis ameaças (MACHADO et al., 2019).

Uma forma para detecção envolve a utilização de Redes Definidas por *Software* (SDN). Diferentemente da rede de dados tradicional, a SDN propõe o desacoplamento entre o plano de controle e o plano de dados, facilitando o gerenciamento da rede e permitindo a configuração por meio de programação. Além dessa separação, as redes SDN incluem um elemento central conhecido como controlador, encarregado de configurar a tabela de encaminhamento dos fluxos nos dispositivos de rede. Através da centralização do coordenador, o administrador da rede pode monitorar o estado global e implementar novas políticas sem a necessidade de realizar a reconfiguração manual em vários equipamentos, tudo a partir de um único ponto.

Detectar os HHs no plano de dados reduz o *overhead* de comunicação e a transferência de dados entre os planos de dados e controle, além de possibilitar a análise de informações de todos os *switches* de uma rede (MACHADO et al., 2019; DING et al., 2020). No plano de dados, os *switches* possuem funções e processos que lidam com encaminhamento de pacotes. Os *switches* se comunicam com o coordenador presente no plano de controle. Este, por sua vez, possui funções e processos que determinam a gerência da rede, além de ser responsável pelas instruções dadas aos comutadores no plano de dados (LIN; HUANG; TSAI, 2019; SONG et al., 2020).

Atualmente, por meio de *switches* programáveis, é viável alcançar um controle mais eficaz do plano de dados, enriquecendo as operações nesse âmbito. Com a origem e avanço dessa tecnologia, surgiu a capacidade de programar diretamente no plano de dados,

incorporando funcionalidades avançadas e permitindo a implementação de soluções de monitoramento mais sofisticadas, no próprio *hardware* do *switch*. A partir dos *switches* programáveis, surgiu a possibilidade de realizar um monitoramento mais detalhado na rede, possibilitando a análise de informações provenientes de todos os *switches* em uma infraestrutura, bem como a rápida implementação de algoritmos de rede (BASAT et al., 2020; BEN-BASAT et al., 2018).

Os *switches* programáveis possibilitam aos usuários criar módulos personalizados no plano de dados, capacitando a análise de cabeçalhos de pacotes específicos, a execução de ações definidas pelo usuário e o acesso à memória interna do *switch* para realizar operações com estado. Uma forma é utilizar a implementação de estruturas probabilísticas nesses *hardwares*, com eficiência de memória, como os *sketches*, reduzindo informações de monitoramento redundantes (DING et al., 2020; IVKIN et al., 2019).

Switches programáveis de alta velocidade, como os modelos Tofino e Broadcom, têm a capacidade de processar pacotes de forma paralelizada utilizando o conceito de *multi-pipes*. Essa arquitetura é projetada para maximizar a capacidade de processamento do *switch*. Os *pipes* conectam uma parte das portas do *switch* a um componente *crossbar* e realizam o processamento exclusivo dos pacotes transmitidos e recebidos para uma porta específica no *switch* (CHIESA; VERDI, 2023).

Esses *pipes* nos *switches* operam de forma independente, processando simultaneamente vários pacotes desde a entrada até a saída, o que resulta uma melhora substancial na eficiência do processamento de dados. Dessa forma, possuem sua própria memória isolada, sem acesso às memórias de outros *pipes*. A quantidade de *pipes* utilizados pode ser configurada de acordo com a capacidade máxima suportada pelo *switch*. Os *hardwares* atuais possuem de 2 até 16 *pipes*. O *switch* Broadcom Tomahawk, por exemplo, possui 16 *pipes*, já outros como o Tofino 2, possuem no máximo 4 *pipes* (BROADCOM, 2019; WHEELER, 2019; AGRAWAL; KIM, 2020; INTEL, 2022).

Na literatura, trabalhos como os de Harrison et al. (2018), Ding et al. (2020), Turkovic, Oostenbrink e Kuipers (2019), Sivaraman et al. (2017) e Basat et al. (2018b), assim como outros, não detectam fluxos HHs na rede com *hardwares multi-pipes*. Essa detecção representa um desafio para contagem precisa do fluxos. Isso ocorre porque, durante a detecção em *switches multi-pipes*, os fluxos podem entrar em qualquer *pipe*, e alternar entre eles ao longo de sua duração. A independência dos *pipes* pode levar a uma contagem imprecisa da quantidade de pacotes por fluxo processados pelo *switch*, resultando potencialmente em uma detecção equivocada do fluxo HH na rede.

Esse problema é agravado quando a detecção ocorre em *Network-Wide*[1], na qual a contagem de um fluxo HH é realizada em vários *switches* na rede. O Monitoramento Distribuído Contínuo (*Continuous Distributed Monitoring* (CDM)), também conhecido como

[1] Optamos por manter no texto o termo em inglês, pois é mais conhecido na literatura e representa melhor o significado que queremos.

Monitoramento *Network-Wide*, é um monitoramento global distribuído em uma rede, que precisa recolher dados de diferentes dispositivos de rede, agrupa-los e então tomar uma decisão (BASAT et al., 2018a; BASAT; EINZIGER; TAYH, 2020; CORMODE, 2011).

As soluções atuais de monitoramento apresentam três principais requisitos: precisão, memória e reatividade. A precisão, nesse contexto, diz respeito a capacidade de identificar corretamente os eventos de rede. A memória refere-se a uma necessidade de baixa sobrecarga de memória, visando minimizar os recursos alocados em um *switch*. Por último, a reatividade aborda o tempo necessário para detectar um evento específico (CHIESA; VERDI, 2023). Nos experimentos realizados, comentamos acerca desses requisitos.

Dessa forma, estendemos um dos trabalhos presentes na literatura que detecta fluxos HHs (HARRISON et al., 2018), apontado como um dos principais no que se refere a detecção de HHs por meio de *Network-Wide*. Nesse contexto, nosso trabalho consiste em detectar fluxos HHs com monitoramento *Network-Wide* em *switches* de borda programáveis *multi-pipe* na rede, além de empregar uma técnica de limites adaptativos para a detecção. *Switches* de borda são dispositivos que recebem fluxos de dados do mundo externo, interagindo diretamente com dispositivos dos usuários finais.

Nesse sentido, pretendemos realizar a detecção de HHs minimizando a comunicação entre o plano de dados e o plano de controle, diminuindo o *overhead* de comunicação, uma característica priorizada por Harrison et al. (2018). Ademais, utilizamos a métrica *F1-Score* para identificar o equilíbrio entre precisão e revocação, visando manter um elevado valor da métrica, assim como soluções que utilizam *switches* com um único *pipe*.

Em vista disso, desenvolvemos um emulador para detecção de fluxos HHs em *switches* de borda *multi-pipe*, e elaboramos duas abordagens para a detecção. A primeira, denominada "*Accumulator*", inclui um contador local em cada *pipe* e um acumulador local do *switch* que calcula a somatória das contagens individuais de cada *pipe*, e comunica com o plano de controle quando esse limite for atingido. Já a outra, chamada de "*Local-pipe*", também possui um contador local em cada *pipe*, porém não utiliza um acumulador no *switch*. Desse modo, cada *pipe* se comunica individualmente com o plano de controle, após terem seus respectivos limites atingidos.

1.1 Objetivo e Contribuições

Este trabalho apresenta uma contribuição significativa para a detecção de fluxos HHs em *switches* de borda programáveis com *multi-pipe*. A solução proposta aborda os problemas decorrentes da adaptação de aplicações de detecção de fluxos projetadas para operar com um único *pipe* em *switches* de borda com *multi-pipes*, onde a distribuição dos fluxos entre os *pipes* pode comprometer a precisão da contagem. Nossa estratégia não apenas identifica esses desafios, mas também propõe uma solução que melhora a precisão da detecção de HHs.

Além disso, a adaptabilidade da nossa estratégia para ser aplicada em uma escala de rede maior, é uma contribuição importante, pois proporciona uma solução versátil e abrangente para problemas de HHs em ambientes de rede complexos, mantendo sua eficácia. A flexibilidade na detecção de HHs, introduzida através de limites adaptáveis, também representa um avanço significativo, reduzindo o *overhead* de comunicação sem comprometer a precisão.

A combinação de alta precisão, eficiência e adaptabilidade faz com que nossa solução seja eficaz em cenários de redes complexas. Os experimentos realizados indicam que nossa abordagem compete de forma sólida com as soluções atualmente utilizadas, que operam com um único *pipe*, e demonstra vantagens em contextos onde *multi-pipes* são empregados.

Foram conduzidos experimentos comparativos que analisaram o desempenho de *switches multi-pipe* em contraste com *switches* que operam com apenas um *pipe*, além de examinar a eficácia de um limite adaptativo em comparação com um limite fixo.

1.2 Estrutura da Dissertação

O restante do trabalho está organizado da seguinte maneira:

- O Capítulo 2 apresenta a motivação que impulsionou a realização deste estudo. Neste capítulo, serão explorados os desafios específicos que serviram como catalisadores para a condução do trabalho.

- O Capítulo 3 aborda os principais conceitos teóricos utilizados neste trabalho, os quais compreendem algumas características de estruturas probabilísticas, características de redes SDN relevantes para o contexto, *OpenFlow*, fluxos HHs e linguagem P4.

- O Capítulo 4 apresenta os estudos e trabalhos relacionados à detecção de fluxos pesados na rede, assim como técnicas para monitoramento desses fluxos, que contribuem para o contexto deste trabalho.

- O Capítulo 5 apresenta o funcionamento da arquitetura de *hardwares multi-pipes*, assim como o algoritmo proposto, as métricas utilizadas para detecção dos fluxos e nossas estratégias para obtenção dos resultados.

- O Capítulo 6 apresenta os resultados obtidos a partir da avaliação realizada, incluindo uma comparação com um *switch* que utiliza um único *pipe* em suas aplicações.

- Por fim, a Conclusão sintetiza as principais considerações e contribuições deste trabalho, também as perspectivas de trabalhos futuros.

Capítulo 2

Motivação

Uma arquitetura de *hardwares multi-pipes* oferece vantagens significativas na manipulação de um grande volume de pacotes, além de reduzir as incidências de conflitos durante o acesso aos recursos de *hardware*. Nosso trabalho se destaca como o primeiro estudo prático, no contexto *Network-Wide*, a aplicar a detecção de HH utilizando *multi-pipes* em *switches* programáveis. Nossas abordagens buscam analisar a maneira mais eficiente de realizar essa detecção, com o objetivo de minimizar o *overhead* de comunicação entre os planos de dados e o plano de controle. Além disso, quantificamos seu impacto em comparação com *switches* programáveis que empregam apenas um *pipe* em suas operações.

Conforme discutido na literatura, muitos estudos assumem que *switches multi-pipes* possuem apenas um único *pipe*, o que pode levar a interpretações imprecisas. Em contraste, nossa abordagem reflete a realidade ao considerar e explorar os múltiplos *pipes* presentes na arquitetura desses *hardwares*.

No entanto, a implementação em *switches multi-pipes* enfrenta desafios que resultam em menor precisão e reatividade em comparação com aplicações projetadas para *switches* operando com apenas um *pipe*. Estes desafios decorrem, em grande parte, devido à complexidade adicional associada à gestão de *multi-pipes*:

- **Ineficiência de memória:** Nos *multi-pipes*, as estruturas de dados de monitoramento são copiadas em todos os *pipes*. Dessa forma, quando pacotes pertencentes à mesma categoria de tráfego (como a monitoração de IPs de origem) são recebidos e transmitidos através de vários *pipes* do *switch*, eles se dispersam por todos esses *pipes*. Isso resulta em um aumento do uso de memória para armazenar dados sobre cada categoria de tráfego.

- **Detecção lenta de ocorrências na rede:** Quando pacotes da mesma categoria

de tráfego se espalham pelos *pipes*, isso dificulta a detecção de ocorrências na rede em tempo hábil. Em situações de detecção de fluxos HH, por exemplo, esses atrasos impactam a capacidade de reduzir a prioridade a fluxos grandes, comprometendo de forma negativa o tráfego de fluxos leves e sensíveis à latência, que compartilham a mesma rede.

- **Detecção imprecisa na rede:** Esse obstáculo surge, pois diversas aplicações são tradicionalmente projetadas e implementadas com base em uma arquitetura de *hardware* que utiliza um único *pipe*, sem levar em consideração os *multi-pipes*. Essa abordagem utilizando apenas um *pipe* pode levar a uma falta de adaptação adequada a *multi-pipes*, resultando em uma série de erros e inadequações ao realizar a migração direta dessas aplicações.

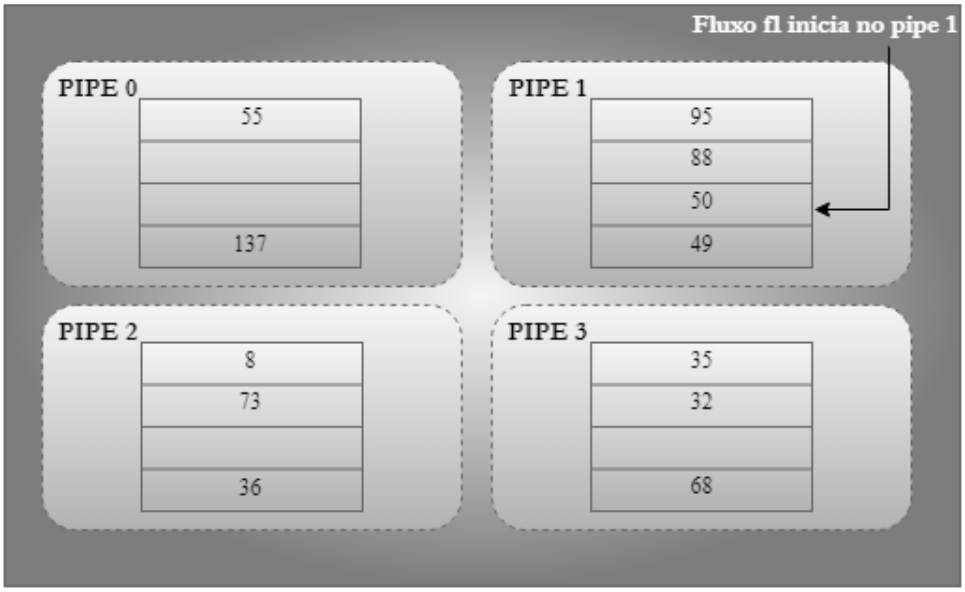


Figura 1 – Espalhamento de um tráfego de rede HH entre diversos *pipes*.

Fonte: Autoria própria.

Retratamos na Figura 1 um exemplo da complexidade envolvida nos *switches multi-pipes*. A figura representa um *switch multi-pipe* equipado com quatro *pipes* ativos em sua arquitetura, executando uma aplicação que detecta HHs na rede. Cada *pipe*, consequentemente possui uma instância independente deste programa de detecção de HHs, devido à sua natureza isolada. Ou seja, se tratando de *hardwares* programáveis, cada *pipe* possuiria o mesmo código para detecção dos fluxos pesados, além dos recursos utilizados pelo *switch* como memória, ALUs, etc. Sendo assim, podemos observar como é a interação entre os diversos *pipes* e como o funcionamento dos *multi-pipes* pode interferir e impactar as aplicações que rodam nos *switches*, neste tipo de ambiente.

Na Figura 1, cada *pipe* possui uma tabela com quatro espaços disponíveis. Cada espaço está destinado a contagem dos pacotes de um fluxo distinto, representando os

diversos fluxos da rede que passam pelo *switch*. Por exemplo, o primeiro espaço da tabela representa o fluxo "X", o segundo representa um fluxo "Y", o terceiro o fluxo "$f1$", e assim por diante.

Em *switches multi-pipe*, cada *pipe* opera a mesma instância da aplicação executada no *switch*, a detecção de padrões HHs na rede. Essa abordagem é adotada devido à natureza independente de cada *pipe*, garantindo uma execução autônoma do programa de detecção de padrões em cada instância. Sendo assim, cada *pipe* contém recursos de memória dedicados, não possuindo conhecimento da memória de outro *pipe*.

Neste exemplo, um fluxo identificado como $f1$ tem início no "*pipe* 1", e sua contagem é realizada na terceira posição da tabela. O valor representado na figura, na posição do fluxo $f1$, é referente a quantidade de pacotes passados por ele, ou seja, 50 pacotes. Após um tempo, devido a fatores como balanceamento de carga de fluxo, oscilações na rede ou falhas, ocorreu uma reconfiguração e o fluxo $f1$ passou a ser encaminhado pelo "*pipe* 3", como representado na Figura 2. Independente da mudança de *pipe*, como ainda é o mesmo fluxo, a contagem continua sendo realizada na terceira posição da tabela. No entanto, uma nova contagem é iniciada nessa posição, pois se trata de outro *pipe*.

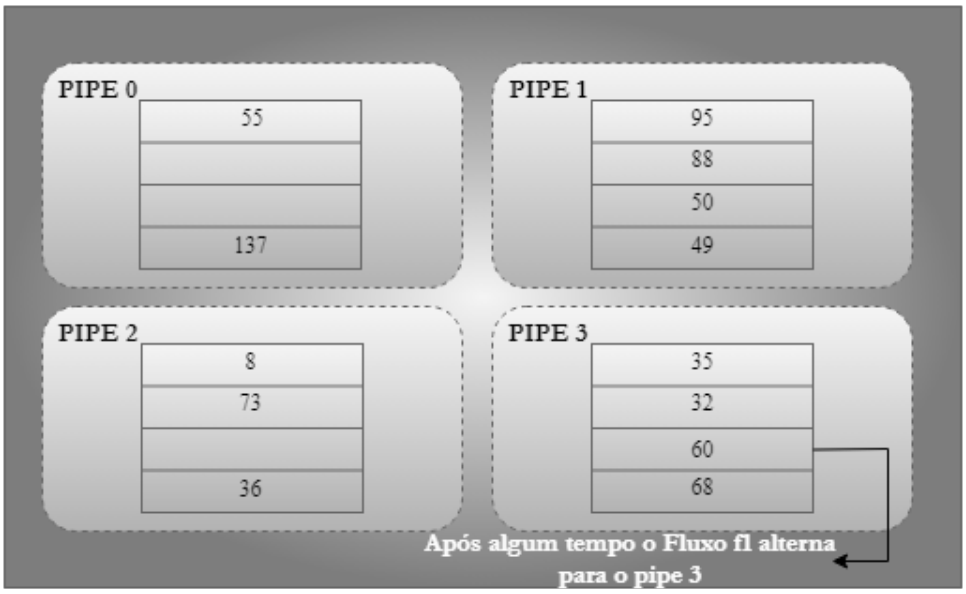


Figura 2 – Espalhamento de um tráfego de rede HH entre diversos *pipes* após alternância do fluxo nos *pipes*

Fonte: Autoria própria.

Certamente, é possível notar a passagem de outros fluxos pelo *switch*, visto que diversas entradas na tabela apresentam contagens de pacotes. No "*pipe* 1" a contagem permanece estagnada em 50 pacotes, enquanto no "*pipe* 3" a contagem totalizou 60 pacotes encaminhados através deste *pipe*. Devido à reconfiguração do fluxo, o contador mantido no "*pipe* 1" acumulou uma parcela dos pacotes contados, enquanto os pacotes remanescentes foram registrados no contador do "*pipe* 3", totalizando 110 pacotes.

Considere que neste exemplo o limite para um fluxo HH seja de 110 pacotes, e o fluxo $f1$ possua um total de 220 pacotes. Neste caso, o $f1$ é considerado um HH, mas não seria detectado, pois parte dos pacotes foi contada no "*pipe* 1" e outra parte no "*pipe* 3", uma vez que um *pipe* não tem acesso às informações armazenadas em outro *pipe*. Além disso, pode ocorrer contagem duplicada. Por exemplo, se 110 pacotes forem contabilizados no "*pipe* 1" e mais 110 no "*pipe* 3", o HH será detectado duas vezes.

Ao ocorrer esta situação, tanto o *switch* quanto o administrador da rede não estarão cientes da presença de um fluxo HH. É importante destacar que qualquer alteração nos campos do cabeçalho, classificaria o fluxo como diferente e, consequentemente, realocaria esse fluxo para uma nova posição na tabela.

Embora apresente desafios, a utilização de *multi-pipes* em *switches* programáveis oferece uma série de vantagens notáveis, tais como:

- **Aumento da Capacidade de Processamento:** A presença de *multi-pipes* possibilita aos *switches* programáveis processar um volume maior de tráfego simultaneamente, elevando significativamente a capacidade total do dispositivo;
- **Redução de Latência:** O processamento paralelo de pacotes reduz os períodos de espera em filas, resultando em menor latência e em um aumento do desempenho para aplicações sensíveis ao tempo;
- **Escalabilidade:** A arquitetura com *multi-pipes* permite aos *switches* programáveis a capacidade de escalabilidade expandida, possibilitando uma adaptação mais eficaz às crescentes demandas de tráfego em redes modernas.

No Capítulo 3, será apresentada uma visão dos principais conceitos utilizados neste trabalho, com o objetivo de proporcionar um entendimento básico abrangente sobre os temas essenciais para entendimento do estudo.

Capítulo 3

Conceitos Básicos

Neste capítulo, serão brevemente discutidos os conceitos fundamentais relacionados a este trabalho. Para uma melhor compreensão do tema, abordaremos aspectos cruciais, incluindo estruturas probabilísticas, fluxos HHs, Redes SDN, Protocolo *OpenFlow* e a Linguagem P4.

3.1 Estruturas probabilísticas

A manipulação de grandes volumes de dados torna técnicas determinísticas inviáveis, exigindo a aplicação de abordagens probabilísticas. Estruturas de dados probabilísticas destacam-se pela característica de fornecerem resultados aproximados, em vez de precisos (SENE, 2020). Conforme destacado por Silveira (2017), algoritmos de processamento de cadeia de dados que fundamentam-se em estruturas probabilísticas possibilitam o monitoramento de tráfego com baixo custo computacional. A manipulação eficiente dessas estruturas é viabilizada pela economia de espaço de memória e pela rapidez de execução.

Para o funcionamento eficiente desse tipo de estrutura, torna-se viável a manipulação de grandes conjuntos de dados utilizando funções *hash*. Tais funções têm como propósito simular aleatoriedade para um determinado conjunto de dados. A precisão da função *hash* é diretamente proporcional à diminuição da probabilidade de erro, mas inversamente proporcional ao consumo de memória. A adoção de um valor ampliado para a criação de uma função *hash* pode gerar um alto valor de precisão ao armazenar informações referentes a uma categoria específica de tráfego em um dado local. Entretanto, a seleção de um valor de *hash* menor acarreta o risco da tabela ser preenchida rapidamente, gerando colisões por chaves semelhantes e perdendo informações de monitoramento, diminuindo a precisão (LOPES; PINTO; OLIVEIRA, 2016).

De acordo com a literatura, quatro tipos de estruturas probabilísticas se destacam: *Bloom filter*, *Count-Min sketch*, *MinHash* e *HyperLogLog*. Cada uma dessas estruturas possui características e funcionalidades específicas para diferentes cenários; no entanto, todas elas compartilham o uso de funções *hash* (SENE, 2020). Dentre outras vantagens, as estruturas probabilísticas podem:

- ❏ Evitar operações de E/S residindo na memória principal;
- ❏ Servir como substitutas dos dados originais, com a vantagem de apresentar baixo custo;
- ❏ Armazenar informações de forma eficiente;
- ❏ Consumir menos recursos do que os dados originais;
- ❏ Permitir atualizações incrementais.

Normalmente, estruturas desse tipo desconsideram informações que seriam necessárias para representar fielmente os dados originais. Assim, as estruturas de dados probabilísticas, como os *sketches*, empregam a quantidade de recursos apresentados ao processamento para aproximar suas respostas, considerando uma margem de erro relativa (LOPES et al., 2017). Isso indica que essas estruturas equilibram a precisão e eficiência, utilizando algoritmos capazes de gerar estimativas rápidas das métricas desejadas, mesmo em cenários onde há restrições de memória ou tempo de processamento. Além disso, soluções baseadas em *sketch* têm se tornado amplamente utilizadas devido à sua capacidade de alcançar alta precisão com uma demanda reduzida de memória (LI et al., 2023).

Segundo Wang et al. (2017), um *sketch* é um tipo de estrutura de dados composta por "H" tabelas *hash*, de tamanhos "T", utilizada para inserir fluxos de dados de alta dimensão em menos dimensões, estimando estes sinais originais. Os *sketches* são qualificados para realizar tarefas relacionadas ao monitoramento de tráfego, podendo ser implementados em *switches* programáveis para reduzir informações de monitoramento redundantes (SANTOS et al., 2019).

A Figura 3 apresenta o diagrama da estrutura de dados do *sketch*. Na entrada, um fluxo de dados passa por uma função de *hash*, transformando-se em uma chave com um valor associado. Cada linha do *sketch* está vinculada a uma função de *hash* independente, responsável por indexar as chaves recebidas. Após a chegada de um item (chave), a posição da tabela correspondente à chave é atualizada com o valor associado (WANG et al., 2017).

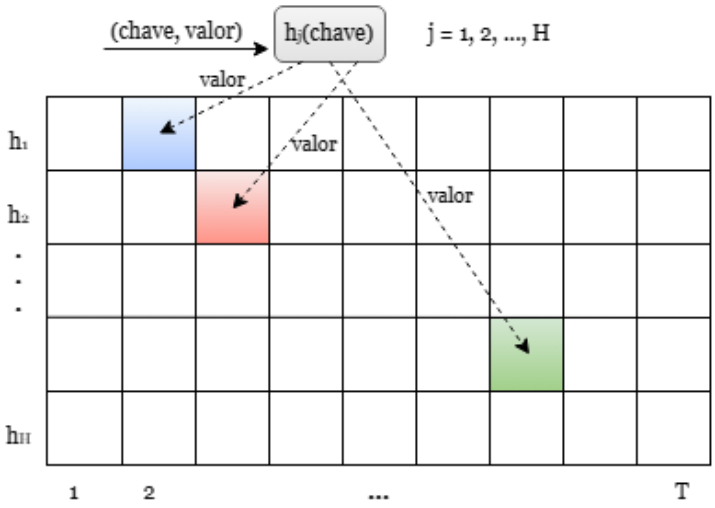


Figura 3 – Diagrama da estrutura de dados do *sketch*.

Fonte: Autoria própria.

Devido as funções *hash* empregarem randomização, a distribuição dos valores na tabela *hash* apresenta-se geralmente estável sob condições normais de tráfego de rede. No entanto, ao lidar com volumes elevados de tráfego, caracterizados por uma maior quantidade de fluxo de dados, os *sketches* podem identificar colisões na tabela. Quanto menor o tamanho do *hash* maior a probabilidade de ocorrência de colisões. Segundo a literatura, uma estratégia para minimizar o número de colisões é utilizar dois *hashes* na criação da chave (WANG et al., 2017). Sendo assim, existem trabalhos que optam por um *hash* enquanto outros preferem dois *hashes* para criação das chaves. Alguns deles serão apresentados no Capítulo 5.

Estruturas probabilísticas encontram aplicação em diversos contextos. Dentre alguns casos de uso podemos citar, as análises de conjunto de *Big Data*, análise estatísticas, mineração de *tera-bytes* de conjuntos de dados e detecção de HHs.

3.2 Fluxos *Heavy Hitters*

Conforme Corrêa, Proto e Cansian (2008), um fluxo de rede consiste em uma sequência unidirecional de pacotes, indo de uma máquina de origem para uma máquina de destino. Além disso, os fluxos são identificados por características comuns, como endereços IP e portas de origem e destino. Os fluxos de dados podem ser caracterizados de várias maneiras, levando em consideração suas características, tais como fluxos pesados, fluxos leves, fluxos longos e fluxos curtos.

Fluxos longos e curtos são categorias baseadas em sua duração temporal, sendo estabelecido um limiar de tempo que, quando ultrapassado, classifica o fluxo como longo. O mesmo princípio se aplica aos fluxos pesados e leves, porém, referindo-se ao tamanho. Em outras palavras, um fluxo é considerado pesado se seu tamanho (em *bytes* ou número

de pacotes) ultrapassar um limite predefinido. Se o fluxo for menor que esse limiar, é classificado como leve (VILELA, 2006).

Nesse contexto, os HHs são categorizados como fluxos pesados, comparando-se a outros fluxos na rede (SIVARAMAN et al., 2017). Em várias pesquisas sobre a detecção de HHs, empregam-se funções *hash* com base nos fluxos observados para criar chaves. Comumente, são mapeados quatro campos de cabeçalhos para criação de uma tupla, sendo eles: IP de origem e destino e porta origem e destino, conforme ilustrado na Figura 4. O valor resultante é convertido em um número que serve como chave de identificação do fluxo. A partir desse valor, realiza-se a contagem dos pacotes transmitidos, indexando essa contagem na tabela na posição correspondente a essa chave (SILVA et al., 2018).

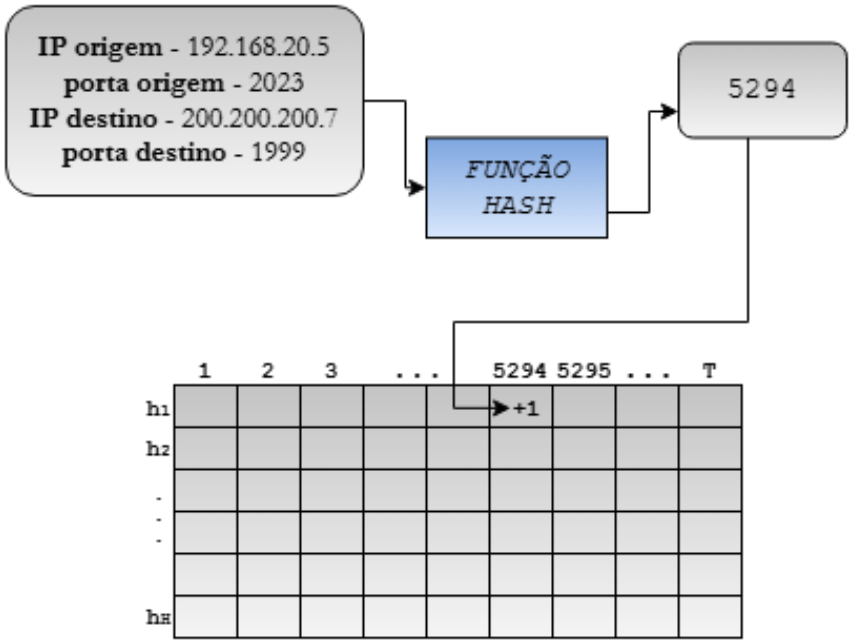


Figura 4 – Criação do *hash* a partir do endereço.

Fonte: Autoria própria.

Na Figura 4, os quatro campos do cabeçalho TCP/IP servem como entrada para a função *hash*. Após passar pela função, é gerado uma chave "5294". Esta chave condensa as informações contidas nos campos, e funciona como um índice em uma tabela *hash*. Na respectiva posição desta tabela, o valor é incrementado com base no conteúdo preexistente. Quando o valor associado à posição "5294" atingir ou ultrapassar o limite estipulado, o fluxo em questão é categorizado como um HH.

Os *switches* de borda são dispositivos de rede que operam nas extremidades da infraestrutura de rede. Esses dispositivos são responsáveis pela conexão dos dispositivos finais, como computadores, impressoras e outros equipamentos de usuário, à rede principal. Operam como pontos de entrada e saída de tráfego de dados, estabelecendo a interface entre os dispositivos terminais e os elementos centrais da rede, incluindo *switches* de distribui-

ção, roteadores e outros componentes essenciais da infraestrutura central (LIBERATO et al., 2018).

Os *switches* de borda são dispositivos de rede que desempenham um papel fundamental na arquitetura das redes de comunicação, localizados nas extremidades da infraestrutura de rede. Esses dispositivos são responsáveis pela conexão dos dispositivos finais, como computadores, impressoras e outros equipamentos de usuário, à rede principal. Em sua função, os *switches* de borda operam como pontos de entrada e saída de tráfego de dados, estabelecendo a interface entre os dispositivos terminais e os elementos centrais da rede, incluindo *switches* de distribuição, roteadores e outros componentes essenciais da infraestrutura central (LIBERATO et al., 2018).

No modelo de monitoramento em toda a rede *Network-Wide*, diversos observadores estão conduzindo observações na rede e buscam trabalhar em conjunto para calcular uma função baseada na combinação de todas as suas observações. No exemplo de uma detecção de HH em *Network-Wide*, o coordenador central reúne informações distribuídas dos diversos *switches*, coleta suas métricas e opera em cima dessas informações. Diversas abordagens podem ser empregadas no modelo de monitoramento *Network-Wide*. Uma estratégia seria instruir todos *switches* da rede a encaminhar suas observações para um único local centralizado, o coordenador. Alternativamente, outra abordagem envolve realizar periodicamente pesquisas nos *switches*.

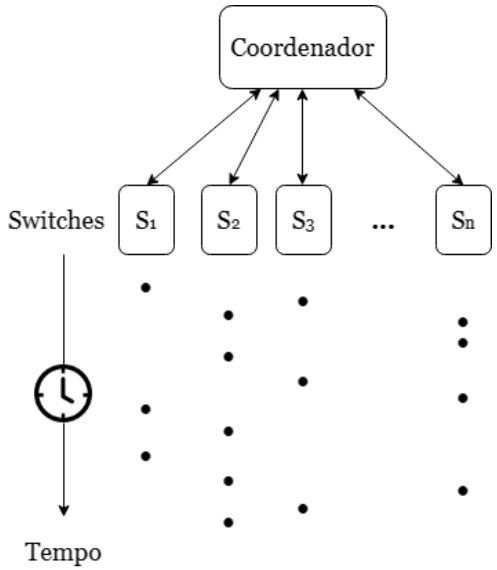


Figura 5 – Modelo de Monitoramento *Network-Wide*.

Fonte: Autoria própria.

Deste modo, um intervalo de tempo fixo é estabelecido e, ao término desse intervalo, o coordenador consulta cada observador para obter suas informações daquele intervalo de tempo (CORMODE, 2011). No caso da detecção de HH, a abordagem mais indicada seria a primeira mencionada, na qual é reportado ao coordenador quando ocorre algum

evento específico.

A Figura 5 ilustra o modelo de monitoramento *Network-Wide*. Por se tratar de uma visão global da rede, é possível observar que a comunicação ocorre entre todos os *switches* e o coordenador. Cada ponto na imagem representa uma observação efetuada pelo respectivo *switch*. Ao longo do tempo, novas observações são continuamente realizadas em diferentes *switches* e em intervalos temporais distintos, gerando mais comunicações entre o coordenador e o *switch*.

Como ilustrado na Figura 5, o monitoramento e a detecção *Network-Wide* empregando múltiplos *switches* aproxima-se das práticas reais das operadoras de rede, uma vez que estas necessitam identificar fluxos HHs em toda a extensão da rede. Exemplos desse tipo de aplicação incluem *port scanners* e *superspreaders*, que requerem uma visão conjunta da contagem e pacotes. Em tais casos, a monitorização em apenas um local não seria suficiente para efetuar a detecção de HH (HARRISON et al., 2018; VIVO et al., 1999; SANKARAN et al., 2019). Em nosso trabalho, aplicamos a detecção em escala de rede, utilizando apenas *switches* de borda, em contraste com o monitoramento *Network-Wide* tradicional que abrange todos os *switches* da rede.

No modelo de monitoramento *Network-Wide* para detecção de HH, os *switches* de borda desempenham o papel da contabilização. Sendo assim, surgem dois tipos de contadores: um contador local no próprio *switch* de borda e um contador global no coordenador da rede (que realiza a contagem de todos *switches* de borda da rede). A contagem dos fluxos é realizada nos *switches* de borda e, quando os contadores indicam que atingiram seu limite para um determinado fluxo, isso sugere a presença de um fluxo HH. Nesse caso, o valor correspondente é enviado ao coordenador, para que medidas sejam tomadas (HARRISON et al., 2018).

3.3 Redes SDN

As Redes Definidas por *Software* surgiram diante da necessidade de adaptar de forma rápida mudanças nas redes tradicionais. Na SDN, o gerente da rede consegue programar e alterar da maneira que preferir o protocolo interno do roteador, visando otimizar o desempenho na troca de dados acerca de seus objetivos. Em sua topologia existe uma divisão em três planos, permitindo uma visão geral da rede: Plano de Dados, Plano de Controle e Plano de Aplicação, além de um controle centralizado (NETO, 2021). De acordo com a literatura, a SDN oferece vantagens, como a simplificação de implementação, flexibilidade para se adaptar às constantes mudanças exigidas pelo ambiente de negócios, programabilidade da rede e uma visão global da rede e seus aplicativos em execução.

Conforme evidenciado na Figura 6, a estrutura da SDN é caracterizada principalmente pelas duas camadas inferiores, sendo o Plano de Controle e Plano de Dados, embora exista os três planos. Na camada superior encontra-se o Plano de Aplicação, onde residem os

serviços e aplicações. Em seguida, em uma camada intermediária, situa-se o Plano de Controle, onde está localizado a inteligência da SDN. Esse plano pode ser comparado a um sistema operacional, dado seu controle logicamente centralizado, embora tal centralização implique em um ponto único de falha (NETO, 2021).

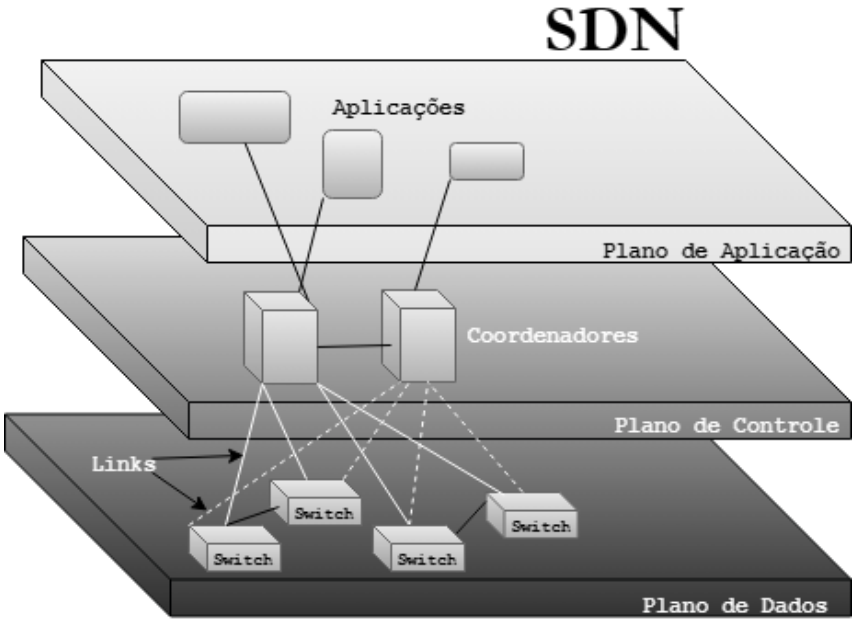


Figura 6 – Infraestrutura de uma SDN.

Fonte: Autoria própria.

Uma das primeiras materializações das redes SDN foi através do protocolo *OpenFlow* (TOURRILHES et al., 2014). Por meio dele os coordenadores do Plano de Controle fornecem uma *Application Programming Interface* (API) que estabelece um método padronizado para realizar operações entre o coordenador e os elementos ativos da rede, onde todas as tomadas de decisões e cálculos são realizados. Isso inclui reunir e gerenciar todas as informações de estado da rede, consultar tabelas de fluxos, instalar e remover regras de encaminhamento, recuperar estatísticas de regras instaladas, entre outras, viabilizando o gerenciamento automático da rede (CRUZ et al., 2014).

Por último, na camada inferior, encontra-se o Plano de Dados, onde estão localizados os dispositivos de encaminhamento responsáveis pelo roteamento dos fluxos de dados. Exemplos desses dispositivos incluem roteadores, pontos de acesso (*Wireless Access Point* (WAP)) e *switches*. Nessa camada, os dispositivos não são encarregados de tomar decisões, dependendo integralmente das regras de fluxo estabelecidas no plano de controle. Sendo assim, os dispositivos recebem instruções do plano de controle e as executam, implementando essas regras, diretamente nas tabelas de fluxo do plano de dados (NETO, 2021; CRUZ et al., 2014).

Conforme mencionado, o monitoramento em escala de rede *Network-Wide* e a contagem para a detecção de fluxos HH são executados no plano de dados. A maioria dos

dispositivos operantes nesse plano utiliza sistemas operacionais fundamentados no Unix ou Linux. Por meio de *drivers* específicos, esses sistemas estabelecem comunicação com o processador *Application Specific Integrated Circuits* (ASIC), possibilitando a execução eficiente das aplicações de rede (SOUSA; ROTHENBERG, 2017).

Aplicações nos *switches*, como armazenamento chave-valor e funções de rede, envolvem considerações de desempenho e utilização de memória. Por exemplo, o tamanho da memória exerce uma influência direta sobre a eficácia do *cache* em rede e a precisão de *sketches* (JIN et al., 2017; LIU et al., 2016). Quando a capacidade da memória é atingida, aplicações como balanceamento de carga e monitoramento necessitam de soluções alternativas, que geralmente apresentam menor desempenho. Até funções básicas dos *switches*, como encaminhamento de pacotes, podem ser comprometidas pela restrição do espaço de memória. Esses problemas são agravados quando essas aplicações são executadas no mesmo *switch*, demandando o compartilhamento de memória com as operações essenciais de encaminhamento (KIM et al., 2018).

Quando o coordenador toma uma decisão, essa informação é comunicada aos elementos integrantes do plano de dados. Nesse contexto, as informações transmitidas são mapeadas diretamente em regras de encaminhamento, que são posteriormente integradas às tabelas de fluxos e inseridas na memória dos *switches* por meio do protocolo *OpenFlow*. É por meio do *OpenFlow* que a lógica de controle é traduzida em comandos específicos, que são então implementados no plano de dados, estabelecendo a maneira com que os dispositivos de encaminhamento se comportarão (NETO, 2021).

3.4 *OpenFlow*

Conforme mencionado por McKeown et al. (2008), o protocolo *OpenFlow* (OF) baseia-se na implementação de tabelas de fluxos nos dispositivos de rede. Sua interface estabelece a conexão entre o *switch*, que é compatível com o protocolo, e o coordenador, alinhando-se assim ao paradigma da SDN. Essa abordagem possibilita que a camada de rede do protocolo seja configurada por meio de programação por *software*.

O OF foi criado com a finalidade de proporcionar alto desempenho e controle de tráfego granular. Dessa forma, o OF estabelece um conjunto de instruções que são integradas ao plano de dados do dispositivo. Consequentemente, é viabilizada a execução de diversas operações sobre os fluxos que ingressam no *switch*, como descartar pacotes, encaminhamento para o coordenador ou a duplicação dos pacotes para uma ou mais portas de saída (CRUZ et al., 2014).

Este protocolo reconhece que os atuais *switches* e roteadores estão equipados com mecanismos essenciais para seu funcionamento, incluindo funções de roteamento baseadas em tabelas de fluxo, utilização de máscaras de sub-rede e análise do fluxo de dados. Dessa forma, o protocolo incorpora e utiliza esses mecanismos preexistentes em seu funciona-

mento, conforme descrito por (XIA et al., 2014).

De acordo com McKeown et al. (2008), em um *switch* OF, o caminho de dados consiste em duas partes: uma tabela de fluxo e uma ação relacionada para cada entrada de fluxo. Os *switches* OF requerem determinados requisitos mínimos para executar suas ações, que podem ser divididos em três partes:

- **Tabela de fluxo:** regras instaladas que informam o *switch* sobre a maneira de processar o fluxo, utilizando uma ação associada a cada entrada;
- **Canal seguro:** estabelece a conexão do *switch* com um coordenador, permitindo que pacotes e comandos sejam transmitidos entre eles;
- **Protocolo *OpenFlow*:** fornece uma forma da comunicação entre um *switch* e um coordenador, em uma arquitetura de rede SDN.

A tabela de fluxo presente no comutador pode conter uma ou mais entradas. Cada entrada da tabela é definida por três campos diferentes, sendo eles: regra, ação e contadores. O componente "regra" define determinadas características do cabeçalho que são empregadas para verificar a entrada na tabela e o pacote que ingressa no *switch*, e se existe alguma combinação entre elas. Nas "ações" são descritas quais as operações que o *switch* deve realizar sobre os pacotes, sendo no mínimo uma. Tais ações podem envolver descarte do pacote, encaminhamento do pacote para uma ou mais portas, envio para o coordenador, modificação do endereço MAC de destino, entre outras. Quanto ao componente "contadores", eles armazenam a quantidade de pacotes e *bytes* que se alinham com a regra, incluindo o intervalo de tempo desde a instalação da regra (CAMERA; ZANETTI, 2019).

O protocolo OF especifica os cabeçalhos de protocolo que utiliza, no entanto, não apresenta uma versatilidade para a adição de novos cabeçalhos. Diante desse cenário, com a popularização do SDN, surgiram novas linguagens de processamento de pacotes, como a P4, que supera certas restrições do OF, proporcionando um aumento na programabilidade das SDN. A partir deste ponto, aplicações envolvendo monitoramento de redes em tempo real começaram a ser criadas (CAMERA; ZANETTI, 2019).

3.5 Linguagem P4

Popularmente conhecida como P4, a linguagem de Programação de Processadores de Pacotes independentes de Protocolo (*Programming Protocol-Independent Packet Processors*) é uma linguagem de alto nível que possibilita a programação em dispositivos de rede do plano de dados. Além de também proporcionar uma programação de processadores de pacotes independentes de protocolos. Outra funcionalidade da P4 é a capacidade de criar soluções customizadas, uma vez que ela opera em conjunto com os protocolos de controle

do SDN (GAVAZZA et al., 2019; CAMERA; ZANETTI, 2019). De acordo com Bosshart et al. (2014), a linguagem possui três objetivos principais, sendo eles:

- **Reconfigurabilidade:** A capacidade de reconfiguração é essencial, demandando dos programadores a habilidade de efetuar modificações na análise e processamento dos pacotes mesmo após a fase de implementação inicial, promovendo flexibilidade e adaptabilidade contínuas.
- **Independência de Protocolo:** O *switch* deve manter-se desvinculado de formatos específicos de pacotes de redes, assegurando uma abstração eficaz que ultrapassa as peculiaridades dos protocolos utilizados.
- **Independência do Alvo:** O compilador da linguagem de programação de redes deve gerar uma descrição desatrelada de dispositivos específicos. No entanto, durante a configuração do dispositivo alvo, é necessário considerar as capacidades essenciais do *hardware* subjacente, garantindo uma adaptação precisa às características específicas do dispositivo em questão.

Após a chegada de um pacote no *switch* de rede P4, os cabeçalhos são inicialmente analisados de acordo com os critérios predefinidos no programa, e nesse momento os campos são extraídos. Posteriormente, as tabelas realizam correspondências *match* com base nos campos extraídos, executando as ações apropriadas, de acordo com as instruções enviadas pelo coordenador, que previamente populou as tabelas (BOSSHART et al., 2014). A especificação da linguagem P4 sugere a subdivisão das tabelas em duas partes:

- ***Ingress:*** Define as portas de saída e a fila associada para o pacote;
- ***Egress:*** Altera o cabeçalho do pacote, realizando operações de remoção ou adição de campos.

Para implementação de um programa escrito na linguagem P4 é necessário em sua estrutura preencher cinco campos: cabeçalhos (*headers*), os analisadores (*parsers*), tabelas (*tables*), ações (*actions*) e a implementação de programas de controle (*control programs*) (GAVAZZA et al., 2019; BOSSHART et al., 2014). A sequência pela qual esses campos são definidos no programa P4 é a seguinte:

- ***Headers:*** São empregados para descrever a sequência e a estrutura de vários campos, referenciando estes campos nos cabeçalhos de pacotes;
- ***Parsers:*** Definem como os campos de cada cabeçalho são identificados e validados;
- ***Tables:*** Representam mecanismos utilizados para o processamento dos pacotes por meio de correspondências (*key/matches*) e suas respectivas ações a serem executadas sobre os pacotes;

- ***Actions:*** São executadas (possivelmente por funções customizadas) com base no processamento dos pacotes;

- ***Control Programs:*** Determinam a ordem de encaminhamento de pacotes no programa P4 e das tabelas aplicadas a um pacote.

Nesse contexto, evidencia-se que a linguagem P4 propicia uma implementação notavelmente mais flexível nos *switches*, conferindo aos programadores uma considerável autonomia na definição do processamento dos pacotes no plano de dados.

A Figura 7 ilustra os elementos e a extensão das capacidades proporcionadas por essa linguagem, apresentando uma abstração do modelo de encaminhamento P4. No segmento superior existe o campo "Configuração do *Switch*", que corresponde ao escopo no qual o programador tem controle por meio do código.

Analisando a imagem, observa-se que os pacotes ingressam por uma interface e são submetidos ao *parser*. O *parser* identifica os campos que compõem o cabeçalho (*header*), removendo-os para a identificação de cada protocolo. Após a retirada desses campos, são encaminhados para as tabelas *match action*, que são divididas entre *ingress match action* e *egress match action*, respectivamente às tabelas *match action* de entrada e saída (CAMERA; ZANETTI, 2019).

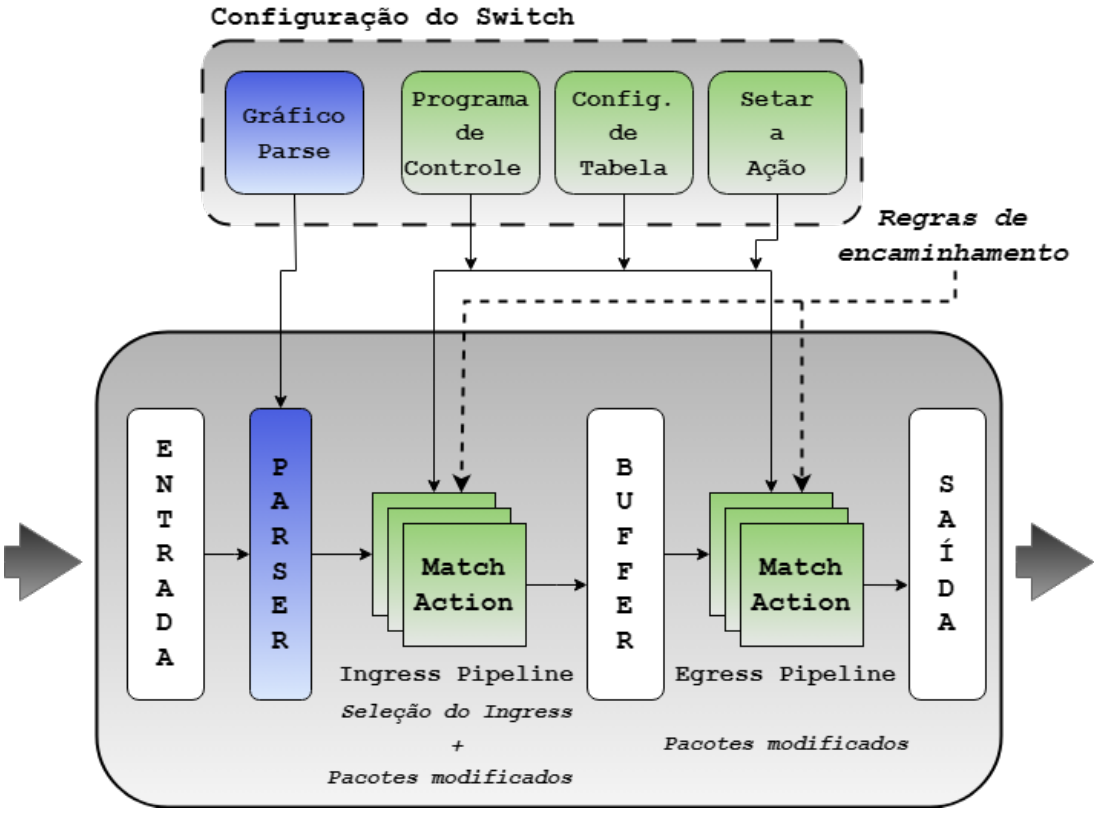


Figura 7 – Modelo abstrato de P4.

Fonte: Autoria própria.

Na *ingress match action*, é possível alterar o *header*, além de determinar as portas de *egress* e a fila na qual o pacote será inserido. Dessa forma, o pacote pode ser encaminhado, replicado para o plano de controle ou descartado. Já na *egress match action*, é possível realizar modificações por instância no *header*, um exemplo seria as cópias *multicast* (CAMERA; ZANETTI, 2019).

A linguagem P4 desempenha um papel fundamental na programação de *switches* programáveis, simplificando o desenvolvimento de códigos para esses dispositivos, que frequentemente apresentam desafios devido a interfaces proprietárias e programação de baixo nível. Embora este trabalho não se baseie na linguagem P4, é importante reconhecer suas qualidades e apresentar seu funcionamento para contextualizar seu impacto na área.

Através da linguagem P4, são alcançados diversos benefícios para o plano de dados, como a possibilidade de balanceamento de carga na camada de transporte, controle de congestionamento de baixa latência, aumento do desempenho na velocidade de rede, entre outros. Essa abordagem simplifica significativamente a programação do comportamento dos equipamentos de rede, proporcionando uma programação de alto nível (LEE et al., 2017; HANDLEY et al., 2017; DANG et al., 2015).

Capítulo 4

Trabalhos Relacionados

Nos últimos anos, temos observado uma evolução gradual de trabalhos que criam soluções que rodam no plano de dados. Uma das aplicações que mais recebeu atenção e variações é justamente a detecção de HH. Neste sentido, o trabalho de Harrison et al. (2018) é apontado como um dos principais no que diz respeito a detecção de HH através de uma *Network-Wide*, o que o torna a base para o método utilizado neste trabalho. Com a possibilidade de programar diretamente em um *switch* de rede dentro do plano de dados, os autores utilizam um método para detecção de HH, distribuído para redes modeladas como um grande *switch*, em *Network-Wide*. São utilizados limites adaptativos baseados em chaves, para reduzir o *overhead* de comunicação entre *switches* e o coordenador, utilizando tabelas *hash* para contar HHs por fluxo. Os *switches* de borda realizam a contagem dos pacotes por meio de chaves nas tabelas *hash*, a partir dos endereços IP de origem/destino e das portas origem/destino. Quando o limite do *switch* é atingido para determinado *hash*, este comunica com o coordenador e, quando o limite global do coordenador é atingido, é computada a média móvel ponderada exponencialmente, baseado na contagem dos demais *switches* da rede, para refinar o limite. Nosso trabalho se baseia nesse conceito, aplicando limites adaptativos, detecção *Network-Wide* e comunicação entre os planos de dados e controle. No entanto, ao contrário dos autores, utilizamos *switches multi-pipes*.

Embora compartilhe algumas similaridades com o trabalho mencionado anteriormente, o estudo de Ding et al. (2020) concentra-se em detectar fluxos HHs em *Network-Wide* em um cenário de rede de Provedor de Serviços de Internet (ISP), empregando uma abordagem incremental. A principal proposta é permitir a implantação incremental de novos *switches* programáveis e, simultaneamente, maximizar a operação de monitoramento de rede, usando um intervalo de tempo para detecção de HHs. Cada *switch* mantém dinamicamente uma lista de amostras contendo apenas fluxos cuja contagem de pacotes excedam

um limite de amostragem dinâmico. Ao término de um intervalo de tempo definido, uma análise é realizada para verificar se algum *switch* atingiu seu limite local. Se isso ocorrer, o *switch* envia um *report* para o coordenador central. Nesse momento, é realizado uma varredura em todos os *switches* na rede para reunir os pares, IP origem/destino e porta-origem/destino, de todos os fluxos em suas listas amostrais. Isso permite estimar o volume total e os fluxos HHs na rede. Após esse processo, os valores são redefinidos e é iniciado um novo intervalo de tempo para uma nova contagem. Podemos observar que os autores também utilizam limites adaptativos, entretanto não são utilizados *switches* programáveis com *multi-pipes*. Além disso, diferentemente do nosso trabalho, o estudo de Ding et al. (2020) utiliza uma técnica distinta para gerar limites adaptativos, empregando intervalos de tempo para a detecção dos HH utilizando apenas um *pipe* em seus *switches*, e abordam um método de *report* único para o plano de controle.

Em outro trabalho, os autores pretendem solucionar a variante de *streaming* do problema de detecção de HHs, detectando-os separadamente no plano de dados, utilizando abordagens de *streaming* de janela deslizante. Dessa forma, Turkovic, Oostenbrink e Kuipers (2019) se baseiam na estrutura de dados probabilísticos *Count-Min sketch* para criar sua própria solução para contagem dos *hashes* encaminhados pelo *switch*. Desenvolveu-se uma abordagem para diminuir as colisões das tabelas *hash*. Para essa finalidade, é utilizado um conjunto de tabelas *hash* que diminuem seu tamanho com a profundidade e um grupo de limites em que seus fluxos menores são filtrados nas primeiras tabelas. Assim, quando um pacote é adicionado ao *sketch*, são reduzidas todas as contagens em uma linha e, ao adicionar memória nos primeiros estágios, ocorre a diminuição das colisões. É mantida uma alta precisão ao longo do tempo na contagem dos *sketches* utilizando a janela deslizante, conseguindo dessa forma a não intervenção do plano de controle. Os autores exploram abordagens distintas das adotadas em nosso estudo para detecção dos HHs, abstendo-se também do uso de *multi-pipes*.

A abordagem proposta por Silva (2019) fundamenta-se na detecção de fluxos HHs no plano de dados, independentemente de sua comunicação com o plano de controle, por meio da implementação de um mecanismo preditivo. Os autores apresentam um mecanismo para identificar esses fluxos antes que atinjam o limite predefinido, prevendo o comportamento dos fluxos. Para isso, utilizam um histórico de fluxos anteriores e correlação temporal para antecipar o comportamento desses fluxos. Esse processo envolve o cálculo do intervalo entre a chegada do último pacote e a entrada do pacote atual, o qual é então comparado com um limite temporal predefinido. Para contagem do volume de tráfego, é utilizada uma indexação de chave *hash*, com pelo menos duas chaves. A utilização de múltiplas chaves *hash* tem como objetivo reduzir o número de possíveis colisões nas tabelas *hash*, mitigando, assim, o risco de detecções falsas de HHs. Dessa forma, quando um pacote alcança um *switch*, são geradas duas chaves para indexar a contagem relacionada ao potencial HH, sendo adotado apenas o menor valor registrado

após a contagem. Os autores verificam se um fluxo já foi identificado como HH e, em caso afirmativo, aplicam uma estratégia baseada no mecanismo *bloom filter* para marcar fluxos previamente reconhecidos. Nesse mecanismo, *slots* da lista são inicializadas com "zeros" e quando um fluxo é identificado como HH os valores são alterados para "um". Em caso negativo, o tamanho e duração do fluxo são estimados utilizando funções *hash*. Diferentemente do nosso trabalho, os autores optam pelo uso da contagem dupla de *hashes*, reduzindo a ocorrência de colisões, e a detecção de HH ocorre exclusivamente no plano de dados, sem interação com o plano de controle.

Outro estudo discute o monitoramento de rede em *switches multi-pipes* (CHIESA; VERDI, 2023). Nesse estudo, os autores abordam o impacto da implantação de soluções de monitoramento existentes em *switches multi-pipes*. Os autores conduziram experimentos utilizando alguns dos melhores mecanismos de monitoramento de plano de dados disponíveis no mercado: *FCM-Sketch* e *Elastic-Sketch*. Dessa forma, concluem que soluções existentes, como as mencionadas, demandam oito vezes mais memória no chip para alcançar um nível de precisão similar a uma implantação de *pipe* único. Desse modo, projetam um sistema denominado *PipeCache*, que adapta soluções de monitoramento para arquiteturas *multi-pipes*, para realizar detecção de HH mantendo o *cache* baixo para alcançar resultados de memória e precisão próximos a um *switch* executando com um *pipe* único. Também pretendem armazenar informações estatísticas sobre cada classe de tráfego em um único *pipe*. As classes de tráfego são particionadas entre os *pipes* existentes e as informações de monitoramento são armazenadas nos *pipes* atribuídos, evitando a dispersão das classes de tráfego monitoradas entre os *pipes*. Esse processo confirma a complexidade de implementar soluções de monitoramento em *switches* utilizando um *pipe* único para um ambiente *multi-pipe*. Ao contrário do *PipeCache*, nossa solução foca na detecção de HH em *Network-Wide* e no impacto dos *reports* do plano de dados para o plano de controle, além de manter a mesma precisão de um *switch* executando em um único *pipe*. Além disso, armazenamos contagens de tráfego em vários *pipes* e posteriormente no plano de controle, não apenas em um único *pipe*.

O trabalho proposto por Li et al. (2023) apresenta um novo *framework* capaz de auxiliar a medição do tráfego em *Network-Wide* de uma maneira mais leve. O *framework* opera dividindo os *sketches* em segmentos e os distribuindo pelas rotas de encaminhamento, implantando-os em vários *switches*. Essa abordagem permite a distribuição dos cálculos entre os *switches*, facilitando o balanceamento de carga por pacote e minimizando o uso de recursos. Cada *switch* armazena um segmento, executando cálculos parciais e acessando a memória por pacote. Dessa forma, a medição é colaborativamente concluída em vários nós. Os autores concluem que sua metodologia atinge equilíbrio de carga por pacote distribuindo de maneira uniforme os cálculos de *hash* e os acessos à memória entre vários *sketches* para cada pacote. Este trabalho se assemelha ao nosso por utilizarem técnicas de *sketches* para os pacotes de rede em *Network-Wide*. Embora alguns experi-

mentos para validar o método tenham envolvido a detecção de HH, os autores optaram por não incorporar o plano de controle, não utilizaram *multi-pipes* de *hardwares* programáveis e tampouco implementaram limites adaptativos para sua detecção. No entanto, apresentaram uma abordagem nova para esse tipo de detecção na rede.

Outro estudo, proposto por Cruz et al. (2014), demonstra uma solução para a detecção *online* de agregações hierárquicas de fluxos, utilizando o *OpenFlow* como uma característica de redes definidas por *software*. Essa generalização hierárquica se baseia no monitoramento de HH e é conhecido como *Hierarchical Heavy Hitters* (HHH). O tráfego de pacotes IP é destacado como um exemplo de dados que podem ser organizados de maneira hierárquica, assim como a agregação dos pacotes em blocos de rede, por protocolos do nível de aplicação ou por sistemas autônomos de roteamento. O HHH consiste basicamente nos HHs organizados em uma estrutura hierárquica, que seguem algumas regras simples, proporcionando informações mais precisas sobre as agregações de tráfego que consomem mais recursos da rede. Sendo assim, o trabalho apresenta uma solução projetada para operar em redes *OpenFlow* para detecção *online* dos HHH, oferecendo uma representação mais compacta e precisa por meio de uma abordagem hierárquica. A solução é composta por um Coletor. Este por sua vez foi projetado como um dispositivo dedicado conectado ao comutador *OpenFlow*, que desempenha o papel de auxiliar a aplicação *OpenFlow* na identificação mais ágil dos HHH. Ao receber o tráfego, o Coletor conduz uma análise em tempo real desse conjunto de pacotes para identificar as agregações hierárquicas correspondentes. Após essa análise, o Coletor encaminha o conjunto de HHH identificados para a aplicação *OpenFlow*. Além de empregarem um método distinto para detectar os HHs, os autores optam por não utilizarem *switches multi-pipes*, além de não realizarem uma detecção em *Network-Wide*.

Em outro estudo, Basat et al. (2018a) descrevem uma abordagem de monitoramento da rede em escala *Network-Wide*, a qual não leva em consideração o roteamento. Em outras palavras, o trabalho não depende de manipulações de tráfego, conhecimento prévio da topologia de rede ou informações sobre protocolos de roteamento para medições em *Network-Wide*. Para alcançar esse objetivo, são distribuídos *Network Monitoring Points* (NMP) pela rede, tornando-a mais flexível em comparação com abordagens convencionais. Os autores identificam três problemas principais para sua solução: estimar o volume total da rede, fornecer estimativas de frequência por fluxo e identificar HH na rede. Para cada um desses desafios, são introduzidos algoritmos eficientes que não modificam o conteúdo dos pacotes. Para a detecção de HH, é empregado um protocolo de amostragem distribuída para obter uma amostra uniforme dos dados, permitindo que o coordenador identifique os HHs por meio de consultas. Embora os autores não tenham utilizado *switches multi-pipes*, nem limites adaptativos, eles afirmam que a abordagem proposta demonstrou eficácia na prática, satisfazendo os parâmetros de desempenho dos dispositivos de rede.

Tabela 1 – Comparação de abordagens para detecção de HH

Artigos	Ano	Multi-pipe	Plano de Dados	Plano de Controle	Network-Wide	Limite Adaptativo
Este trabalho	2024	✓	✓	✓	✓	✓
(CRUZ et al., 2014)	2014	✗	✓	✓	✗	✗
(HARRISON et al., 2018)	2018	✗	✓	✓	✗	✓
(BASAT et al., 2018a)	2018	✗	✓	✓	✓	✗
(TURKOVIC; OOSTENBRINK; KUIPERS, 2019)	2019	✗	✓	✗	✗	✗
(SILVA, 2019)	2019	✗	✓	✗	✗	✗
(DING et al., 2020)	2020	✗	✓	✓	✓	✗
(CHIESA; VERDI, 2023)	2023	✓	✓	✗	✗	✗
(LI et al., 2023)	2023	✗	✓	✗	✓	✗

A Tabela 1 apresenta uma análise detalhada das divergências entre nossa abordagem e outras propostas mencionadas na literatura. Para ressaltar essas diferenças, definimos características distintas, como: a detecção de HH em *switches multi-pipes*, a detecção no plano de dados, a detecção com auxílio do plano de controle, detecção em *Network-Wide* e a detecção empregando limites adaptativos. Observamos que todos os estudos da literatura apresentados utilizam o plano de dados para o monitoramento de rede e, consequentemente, para a detecção de HH. Alguns empregam de maneira conjunta o uso do plano de controle em suas metodologias.

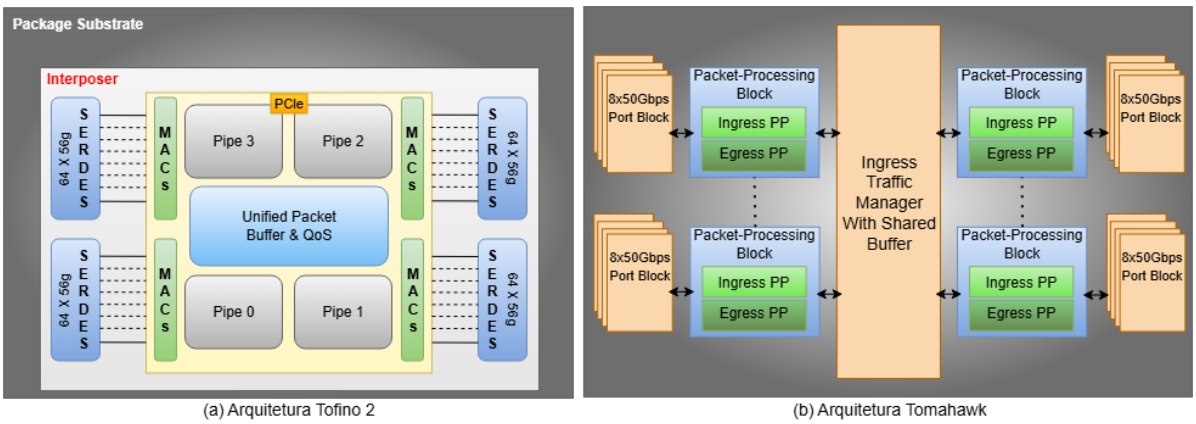
Embora a abordagem *Network-Wide* seja amplamente estudada, pesquisas que se concentram na detecção de HH, revelam uma falta de consenso majoritário quanto à adoção desse método específico. Além disso, observamos que, entre os diversos estudos encontrados na literatura sobre detecção de HH, mencionados anteriormente, apenas um deles trata da utilização de limites adaptativos, assim como a utilização de *switches* programáveis *multi-pipes*.

Sendo assim, os trabalhos apresentados demonstram métodos de detecção em contextos similares, porém não abordam o conceito amplo do nosso trabalho, identificado como avanço necessário no estado da arte deste tema, que atende a uma variedade de requisitos não abordados por outras pesquisas. Este trabalho visa avançar o estado da arte ao combinar a detecção de HH em um cenário *Network-Wide* com a aplicação de limites adaptativos em *hardwares multi-pipes*. Posteriormente, no Capítulo 5, exploramos o funcionamento de um *switch multi-pipe* e apresentamos as duas abordagens desenvolvidas neste estudo.

Capítulo 5

Heavy Hitter em *Multi-pipes*

Nosso trabalho se destaca como o primeiro estudo prático, no contexto *Network-Wide*, que aplica a detecção de HH em *switches multi-pipes*, analisando a maneira mais eficiente de realizar essa detecção e quantificando seu impacto em comparação com um *hardware* que utiliza apenas um *pipe*. Atualmente, dois dos *hardwares* programáveis *multi-pipes* mais conhecidos no mercado são o Intel Tofino e Broadcom Tomahawk. Respectivamente, possuem no máximo quatro e 16 *pipes* em sua arquitetura (WHEELER, 2019; AGRAWAL; KIM, 2020; BROADCOM, 2019; INTEL, 2022). A arquitetura de ambos pode ser visualizada na Figura 8a e na Figura 8b.



(a) Arquitetura Tofino 2 (b) Arquitetura Tomahawk

Figura 8 – Arquitetura do *hardware multi-pipe* Tofino 2 e Tomahawk.

Em uma arquitetura *multi-pipe*, cada *pipe* estabelece uma conexão entre um subconjunto das portas de entrada e saída do *switch* em um módulo de *crossbar*, responsável por mover pacotes entre os *pipes*. Nas figuras Figura 8a e Figura 8b, a *crossbar* está centralmente posicionada na arquitetura. Na Figura 8a a *crossbar* é representada pelo

campo denominado "*Unified Packet Buffer & QoS*", enquanto na Figura 8b, é designada como "*Ingress Traffic Manager with Shared Buffer*".

Além disso, em um *switch multi-pipe*, cada *pipe* dispõe de um conjunto específico de memória e recursos computacionais, que inclui SRAM, TCAM e ALUs. Esses recursos são empregados para armazenar e processar informações necessárias, de acordo com o *pipeline* de processamento de pacotes.

Na arquitetura *multi-pipe*, cada *pipe* possui um bloco dedicado ao processamento de pacotes, que compreende componentes de entrada (*ingress*) e de saída (*egress*), como ilustrado na Figura 8b. A Figura 8a apresenta quatro *pipes* independentes, para processamento paralelo de pacotes. A transmissão e recepção eficiente dos dados são possibilitadas pelos MACs (*Media Access Control*) e SERDES (*Serializer/Deserializer*) de alta velocidade conectados a cada *pipe*. Além disso, a interface PCIe proporciona uma comunicação ágil com outros componentes do sistema.

Embora os *multi-pipes* tenham como objetivo aprimorar o desempenho dos *hardwares* programáveis, é importante destacar que soluções encontradas na literateira, projetadas para operar com apenas um *pipe* (como detecção de HH, balanceamento de carga, etc), não consideram a presença *multi-pipe* em seus *hardwares* (como já mencionado). Ou seja, quando essas soluções são aplicadas em *multi-pipe* podem ocorrer irregularidades, resultando em contagens incorretas ou no mau funcionamento da aplicação. Portanto, torna-se necessário realizar ajustes para garantir um funcionamento ideal.

Aplicações em *hardwares* programáveis *multi-pipe* operam de maneira distinta quando comparadas com a execução com somente um *pipe*. Para exemplificar essa diferença, criamos um exemplo ilustrativo retratado na Figura 9, demonstrando uma situação com um *switch multi-pipe*. Nesse cenário, temos uma divisão para separar o coordenador presente no plano de controle do *switch* presente no plano de dados. Representamos um *switch* que possui quatro *pipes* e está recebendo um pacote da rede. Cada *pipe* possui uma instância do mesmo programa P4 executando individualmente, uma vez que os *pipes* não compartilham memória.

O cenário demonstra uma situação em que a contagem de pacotes da rede é utilizada para identificar um fluxo de HH, semelhante à nossa abordagem. O processo está dividido em quatro etapas:

1. Os pacotes provenientes da rede ingressam em um *switch* de borda;

2. Cada pacote é encaminhado para um dos *pipes* existentes no *switch*, nesse caso, o pacote foi direcionado para o "*pipe* 2";

3. Uma chave *hash* é gerada a partir de quatro campos do pacote (IP origem, IP destino, porta origem e porta destino), representando um fluxo "X";

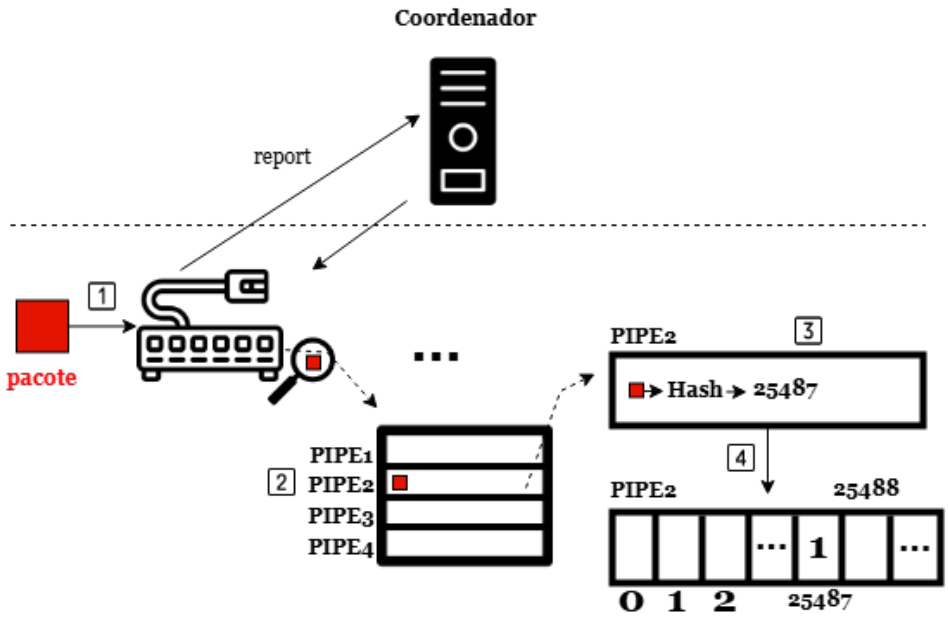


Figura 9 – Funcionamento da rede para detectar os HHs.

4. Com a chave *hash* gerada, a contagem dos pacotes é iniciada em uma tabela *hash* (registro), na posição correspondente à chave obtida.

É importante destacar que qualquer alteração nestes campos, estatisticamente geraria outra chave *hash*, movendo-a para outra posição na tabela *hash*. Suponhamos que um pacote subsequente, pertencente ao mesmo fluxo, seja recebido; no entanto, ele é encaminhado para um *pipe* que ainda não iniciou a contagem. Nessa circunstância, uma nova contagem será iniciada a partir da posição correspondente na tabela *hash*, associada a esse novo *pipe*.

Em nosso trabalho, desenvolvemos duas abordagens de detecção: ***Local-pipe*** e ***Accumulator***. Em resumo, a abordagem ***Local-pipe*** notifica o coordenador sempre que o limite local de um *pipe*, para determinado fluxo, é atingido. Por outro lado, a abordagem ***Accumulator*** incorpora um acumulador local no *switch*, em um de seus *pipes*, que recebe a contagem dos limites dos *pipes* quando atingidos, notificando o coordenador quando o limite do *switch* é atingido.

O coordenador, situado no plano de controle, possui um limite global. Este valor global é distribuído entre todos os *switches* de borda na rede, resultando em um limite específico para cada *switch* de borda. Esse limite global representa o valor estipulado de pacotes para que o fluxo seja considerado um HH. Por sua vez, o "limite do *switch*" é dividido igualmente entre todos os *pipes* disponíveis dentro desse *switch*. Consequentemente, cada *pipe* possui um limite local próprio.

Em relação as nossas estratégias, tanto no *Local-pipe* quanto no *Accumulator*, a distribuição dos limites locais dos *pipes* segue um padrão específico: se o número do limite

local do *switch* for múltiplo da quantidade de *pipes* cada *pipe* receberá uma fração desse valor; resultando em limites locais idênticos para todos os *pipes*. Caso o número do limite local do *switch* não seja múltiplo da quantidade de *pipes*, então alguns *pipes* terão limites maiores que outros.

Em ambas as abordagens, implementamos uma técnica de limite adaptativo, baseado no trabalho de Harrison et al. (2018). Em vez de adotarmos um limite fixo para o tamanho dos HHs, optamos por um limite ajustado dinamicamente. Essa adaptação é baseada na Média Móvel Ponderada Exponencialmente (EWMA) das contagens locais dos *switches* e da contagem do coordenador. A necessidade desse limite adaptativo surge quando um dos *switches* retem a maior parte do tráfego. Nesse caso, o limite local desse *switch* tende a ser atingido mais rapidamente e, para evitar isso, o limite é periodicamente ajustado.

No contexto do limite adaptável, quando o limite global é alcançado para um fluxo específico, um HH é identificado. Nesse cenário, o coordenador efetua um cálculo para ajustar o limite do *switch*. O propósito deste ajuste consiste na otimização do limite para a detecção dos HHs. A variação desse limite pode ser alterada de 10 mil pacotes para 5 mil ou 40 mil pacotes por exemplo, sendo a escolha determinada pela dinâmica do fluxo de dados nos *switches* em questão, no momento da aplicação do ajuste.

Neste ponto do processo, calculamos o EWMA para determinar o novo limite local do *switch* que enviou a última atualização da tabela referente ao fluxo que atingiu o limite global. A técnica EWMA é baseada em dois tipos de valores: antigos e atuais, utilizando um parâmetro de suavização (alfa). O valor deste parâmetro deve sempre estar situado entre 0 e 1. Sendo que, valores mais próximos de 1 conferem maior importância aos dados mais recentes, enquanto valores mais próximos de 0 atribuem mais peso aos dados históricos.

Em nossa equação EWMA, o cálculo é aplicado da seguinte forma:

$$\mathbf{ewma} = \frac{(1-\alpha) * oL_i + \alpha * sReport_i}{\sum_{j=1}^{N}(1-\alpha) * oL_j + \alpha * sReport_j} \tag{1}$$

$$\mathbf{newL} = ewma * (gL - \sum_{j=1}^{N} sReport_j) + sReport_i \tag{2}$$

A equação está adaptada para a abordagem do *Accumulator*. Sendo assim, as variáveis presentes na fórmula representam: o novo limite do *switch* ($newL$), o limite global (gL), o limite anterior do *switch* (oL), e o valor do contador do *switch* em conjunto com os resíduos dos *pipes* também do *switch* ($sReport$). Essa última variável apresenta variação com relação aos índices "i" e "j", na qual o "i" refere-se ao *switch* que atingiu seu limite local e enviou um *report* ao coordenador ($sReport_i$), enquanto o "j" engloba os demais *switches* que não atingiram seus limites locais ($sReport_j$).

A diferença entre a equação adaptada para o *Accumulator*, conforme apresentado, e a equação adotada na abordagem *Local-pipe* reside na diferença da variável *sReport*. Essa

situação se deve à falta de um acumulador no *Local-pipe*, portanto o *sReport* representa apenas os resíduos dos *pipes* de um *switch*. Os valores obtidos das Equações 1 e 2 resultam em valores naturais, pois *switches* programáveis não suportam operações com ponto flutuante.

O procedimento para estabelecer um novo limite é acionado quando um *switch* emite um *report* ao coordenador, resultando no alcance do limite global. Nesse instante, o coordenador realiza o cálculo do novo limite local para o referido *switch*, utilizando o valor do contador fornecido pelo *switch* e verificando a presença de resíduos na contagem de pacotes nos seus *pipes*. Além disso, o coordenador realiza uma varredura em todos os *switches* da rede para capturar os valores dos contadores, bem como os resíduos presentes nos *pipes* de cada *switch*. Após a coleta completa dos dados, o cálculo do EWMA é executado, resultando na geração de um novo limite. Vale ressaltar que essa modificação no limite ocorre exclusivamente no *switch* responsável pelo envio do *report* que provocou o alcance do limite global.

Utilizamos a técnica do EWMA para determinar um limite mais eficiente para a detecção dos HHs. Esta abordagem permite a calibração dinâmica de limites locais, permitindo determinar valores ideais com base em cada *switch* de borda e em cada fluxo. Essa abordagem visa reduzir o *overhead* de comunicação entre o plano de dados e o plano de controle.

Os detalhes do funcionamento das duas propostas para detecção de HHs em *switches multi-pipes* estão nas Seções 5.1 e 5.2.

5.1 *Local-pipe*

Na solução *Local-pipe*, a contagem de pacotes é realizada individualmente por cada *pipe*. A Figura 10 representa um exemplo do funcionamento do *Local-pipe*.

Na Figura 10, ilustramos um exemplo de um *switch* programável com quatro *pipes*. Cada *pipe* possui um limite local de 250 pacotes, resultando em um limite total do *switch* de 1000 pacotes. Na figura, a tabela dentro do *pipe* representa uma tabela *hash*, onde cada linha corresponde a uma posição de *hash* para um fluxo específico.

Quando um pacote de um determinado fluxo chega ao *switch*, esse pacote é direcionado para um dos *pipes* existentes no *switch*, onde a contagem para esse fluxo é realizada. Quando um *pipe* atinge seu limite local, o valor alcançado e o *hash* referente ao fluxo são enviados ao coordenador. As setas na figura representam esse processo de envio de informações para o coordenador. No "*pipe* 1", um dos fluxos atingiu o limite local, resultando na transmissão dessa informação ao coordenador, e o valor na posição do *pipe* que atingiu o limite local é zerado.

Caso o limite do *pipe* para um *hash* específico for atingido novamente, esse valor será enviado ao coordenador e adicionado ao valor existente nessa posição. Esse processo de

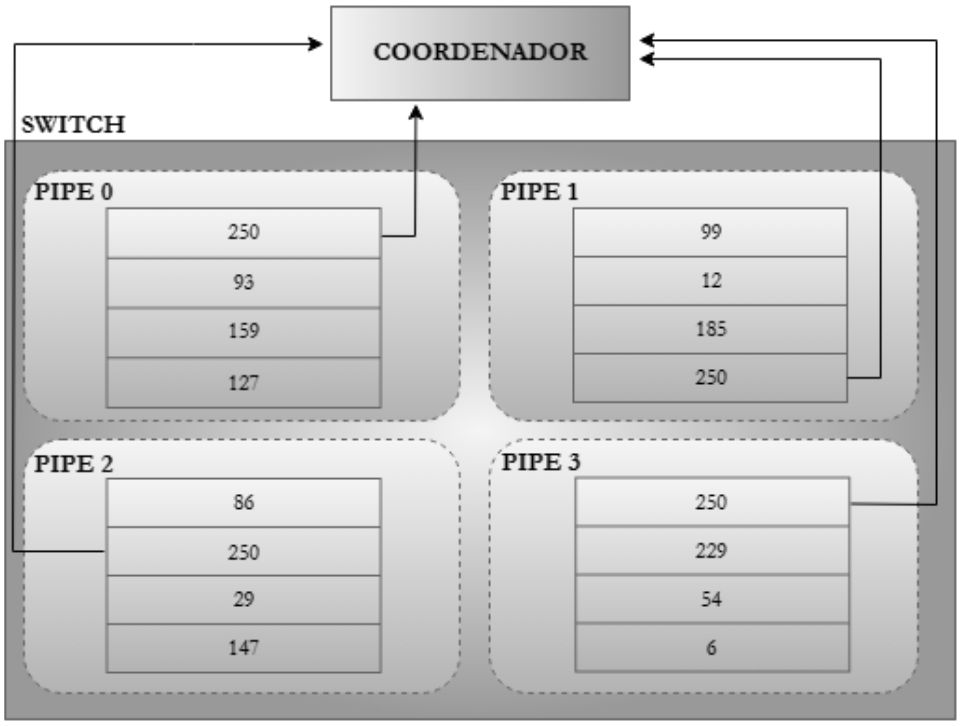


Figura 10 – Execução do *Local-pipe* em um *switch* programável.

atualização continua até que o limite global seja atingido ou excedido para esse *hash* específico. Quando isso ocorre, é um indicativo de que um fluxo de HH pode ter sido identificado.

5.2 *Accumulator*

Na solução do *Accumulator*, a contabilização dos pacotes também é realizada individualmente para cada *pipe*. No entanto, o diferencial é a existência de um acumulador. Um dos *pipes* do *switch* é designado para acumular as contagens de pacotes de todos os *pipes* para cada fluxo de dados. Portanto, o *pipe* designado como o acumulador desempenha duas funções: contar os pacotes individuais transmitidos por ele e acumular as contagens de pacotes de todos os *pipes* do *switch*. O funcionamento do *Accumulator* é ilustrado na Figura 11.

Nesse caso, temos um cenário com um coordenador e um *switch* que possui quatro *pipes*. O *switch* possui um limite de 1000 pacotes, enquanto os *pipes* têm um limite local de 250 pacotes cada. No exemplo fornecido, o "*pipe* 0" não apenas possui uma tabela *hash*, mas também atua como o acumulador (ACC), que possui o mesmo limite do *switch*.

Na solução, quando um *pipe* atinge seu limite local, em vez de encaminhar as informações para o coordenador, as informações são recirculadas dentro do *switch*, para o acumulador na posição correspondente ao *hash* do fluxo. Este cenário é ilustrado pelas linhas contínuas e tracejadas, sendo que a linha sólida representa a recirculação interna

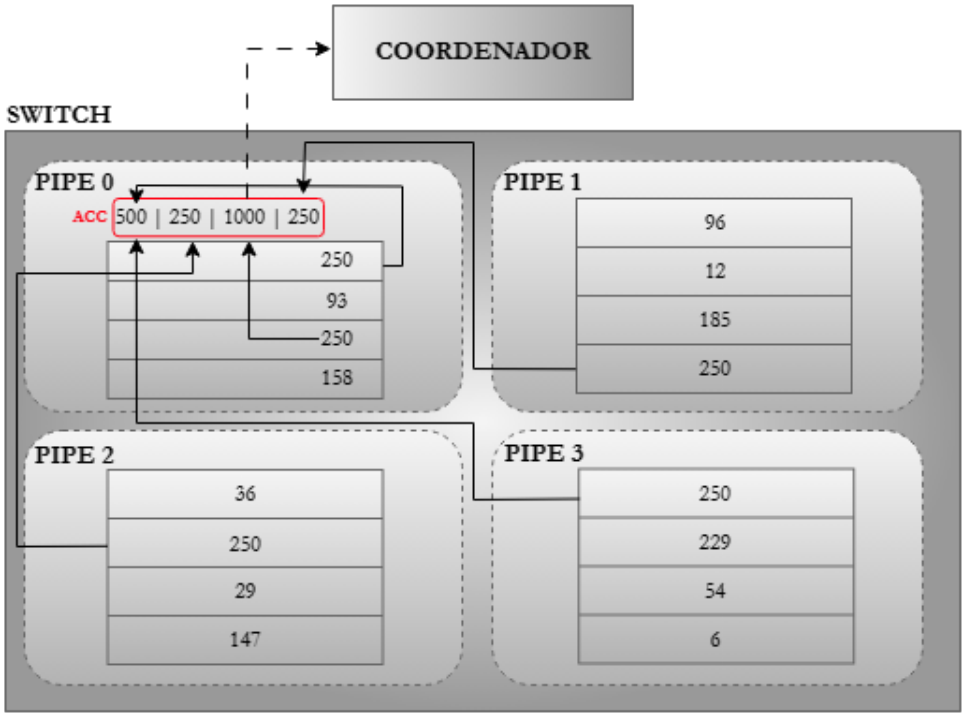


Figura 11 – Execução do *Accumulator* em um *switch* programável.

dentro do *switch*, enquanto a linha tracejada denota a comunicação do *switch* com o coordenador.

Ou seja, no "*pipe* 2", temos um fluxo que atingiu o limite local do *pipe*. Nesse momento, esse valor e sua posição na tabela são recirculados para a segunda posição do acumulador e somados ao valor que estava anteriormente nessa posição. No "*pipe* 2", a contagem nessa posição é redefinida para zero. Também é possível observar que o "*pipe* 1" atinge seu limite local para outro fluxo, e o valor é recirculado para o acumulador.

Da mesma forma, o processo ocorre no "*pipe* 0", recirculando para o próprio *pipe*. Observamos que o "*pipe* 0" atinge o limite para dois fluxos distintos. Um desses fluxos, quando somado ao valor existente no acumulador, atinge uma contagem de 1000 pacotes. No acumulador, quando o limite é atingido para um fluxo, o valor alcançado, bem como seu *hash*, são encaminhados para o coordenador, conforme indicado pela linha tracejada. A posição referente a esse fluxo é zerada no acumulador.

Como o *Accumulator* incorpora um acumulador com um limite maior do que os limites individuais dos *pipes*, temos uma redução das comunicações com o plano de controle, em comparação com o *Local-pipe*, demonstrando ser uma solução mais eficiente.

No Capítulo 6, apresentamos os experimentos realizados e os resultados obtidos. Isso inclui uma comparação entre as duas abordagens desenvolvidas, bem como uma análise comparativa com nossa *baseline*, que consiste em um *switch* executando a aplicação em um único *pipe* com limite fixo.

Capítulo 6

Resultados

6.1 Visão geral

Para conduzir nossos experimentos, desenvolvemos um emulador utilizando a linguagem *Python*, que está disponível no *github*[1] para uso e reprodução. O emulador suporta um número configurável de *switches*, assim como um número configurável de *pipes* por *switch*. Para nossos experimentos, definimos um cenário com três a seis *switches* de rede, todos *multi-pipe*, suportando de dois até 16 *pipes*. Para emular uma distribuição de tráfego do mundo real, utilizamos um *trace* CAIDA *Equinix-NYC* que contem 28 milhões de pacotes (CAIDA, 2019).

A geração do *hash* neste trabalho envolve a aplicação do método MD5 com 16 *bits*. Adicionalmente, definimos um fluxo como uma composição de quatro campos do cabeçalho TCP/IP (IP de origem, IP de destino, porta de origem e porta de destino). A partir disso, identificamos a presença de 1,8 milhão de fluxos distintos no *trace* utilizado. Além disso, estabelecemos um valor arbitrário do limite para classificação de um HH, considerando fluxos que atingem ou ultrapassam o valor de 10 mil pacotes, correspondendo a 0,03% da quantidade total de pacotes registrados. Dessa maneira, após a aplicação desse critério de filtragem no *trace* CAIDA, obtivemos 299 fluxos identificados como HH.

Para representar a variação dos fluxos pelos *pipes*, nosso emulador altera os fluxos entre os *pipes* sempre que o valor de pacotes transmitidos ultrapassa a metade do limite de pacotes do *switch* para um HH em um determinado *pipe*. Dessa forma, o fluxo seleciona aleatoriamente outro *pipe* do *switch* para prosseguir com o encaminhamento de pacotes. Além disso, cada *switch* receberá uma parcela dos pacotes do fluxo, eliminando o risco de

[1] https://github.com/dcomp-leris/Heavy-Hitter-Detection

o mesmo fluxo ser contabilizado erroneamente em mais de um *switch* e garantindo uma contagem precisa dos HH.

Aplicamos as duas técnicas para a detecção de HH propostas neste trabalho, e as comparamos entre si. Uma utiliza um limite fixo no *switch* e a outra utiliza um limite adaptativo no *switch*. Além disso, os mesmos experimentos foram conduzidos para *switches* que utilizaram somente um *pipe*, com limites fixos (*baseline*), ou com limites adaptativos. Isso nos permitiu avaliar que, mesmo aplicando uma detecção de HHs em *multi-pipes*, foi possível manter um elevado *F1-Score*, em comparação com nossa *baseline*.

A métrica de avaliação adotada neste trabalho foi o ***F1-Score***. Essa métrica corresponde à média harmônica entre as métricas de ***Recall*** e **Precisão**, proporcionando um equilíbrio ponderado entre ambas. Sua aplicação se torna particularmente relevante quando é necessário levar em consideração tanto valores de falsos negativos quanto de falsos positivos. A fórmula para o cálculo do *F1-Score* é detalhada na Equação 3.

$$\mathbf{F1} = \frac{2 * precisão * recall}{precisão + recall} \tag{3}$$

Como mencionado, o *Recall* e a Precisão são utilizados na obtenção do *F1-Score*. O *Recall* avalia, dentre todas as instâncias de verdadeiros positivos, quantas foram corretamente identificadas pelo modelo. Por sua vez, a Precisão fornece informações sobre a proporção de observações que o modelo classificou corretamente como HH. As métricas empregadas para calcular Precisão e *Recall* são:

- **Verdadeiro Positivo (VP):** Representa o número de instâncias positivas que foram identificadas corretamente. Ou seja, aqueles que são HH e foram classificados como HH.

- **Verdadeiro Negativo (VN):** Representa o número de instâncias negativas que foram identificadas corretamente. Ou seja, aqueles que não são HH e foram classificados como não HH.

- **Falso Positivo (FP):** Refere-se ao número de instâncias negativas que foram classificadas incorretamente. Ou seja, aqueles que não são HH e foram classificados como HH.

- **Falso Negativo (FN):** Indica o número de instâncias positivas que foram classificadas incorretamente. Ou seja, aqueles que são HH e foram classificados como não HH.

O cálculo do *Recall*, é obtido pela divisão dos verdadeiros positivos pela soma dos verdadeiros positivos e falsos negativos. Por sua vez, a precisão quantifica a porcentagem de predições corretas realizadas pelo modelo, sendo calculada pela relação entre o número

de verdadeiros positivos e falsos positivos. Ambas as equações podem ser formalmente expressas como segue, respectivamente, nas Equações 4 e 5.

$$\textbf{Recall} = \frac{VP}{VP + FN} \tag{4}$$

$$\textbf{Precisão} = \frac{VP}{VP + FP} \tag{5}$$

Desenvolvemos um arquivo utilizando dicionário em *Python*, o qual contém a contagem de pacotes para todos os fluxos no nosso arquivo da CAIDA. Assim, conseguimos realizar uma contagem precisa do número de pacotes associados a cada fluxo de dados que estão passando pelo emulador, excluindo potenciais interferências causadas por colisões, e compará-lo com a nossa abordagem. Posteriormente, essas contagens são utilizadas para a determinação dos VPs, VNs, FPs e FNs. A partir desses valores, o cálculo do *F1-Score* é então realizado.

Para todos os valores das métricas, foram atribuídos tamanhos distintos para geração do *hash*. Observa-se que, à medida que o tamanho do *hash* diminui, há uma maior probabilidade de ocorrência de colisões na *hash table*, resultando em uma menor precisão na detecção dos HHs. O número máximo de entradas que podem ser armazenadas nas *hash tables* estabelece, consequentemente, o número máximo de chaves que podem ser monitoradas no plano de dados.

Ao empregarmos um *switch multi-pipe*, conseguimos manter um alto *F1-Score*, alcançando um valor de 0,99, tanto com o *Local-pipe* quanto com o *Accumulator*. Além disso, com o *Accumulator*, reduzimos o número de *reports* para o plano de controle em comparação com nossa *baseline*. Os resultados desses experimentos serão apresentados em detalhes nas próximas seções.

6.2 Sensibilidade aos *Pipes*

No mundo real, as redes frequentemente passam por flutuações durante a transmissão de pacotes, tanto aquelas resultantes de alterações planejadas, quanto aquelas oriundas de falhas inesperadas. O objetivo do nosso primeiro experimento foi encontrar o melhor fator de suavização (valores maiores dão mais peso a dados recentes, enquanto valores menores dão mais peso a dados históricos) para cada quantidade de *pipes*.

Para conduzir o experimento, realizamos variações no fator de suavização em conjunto com a variação na quantidade de *pipes*. Na Figura 12, pode-se observar o comportamento do *Accumulator* para diferentes quantidades de *pipes*. Utilizamos no experimento *switches* programáveis *multi-pipes* variando de 2 até 16 *pipes* com um fator multiplicativo de 2, além de um *switch* que executa com apenas um *pipe*. Também utilizamos um limite de 10 mil pacotes. O eixo X representa o fator de suavização, variando de 0,2 a 0,99, enquanto o eixo Y representa a quantidade de *reports* em relação aos *pipes*.

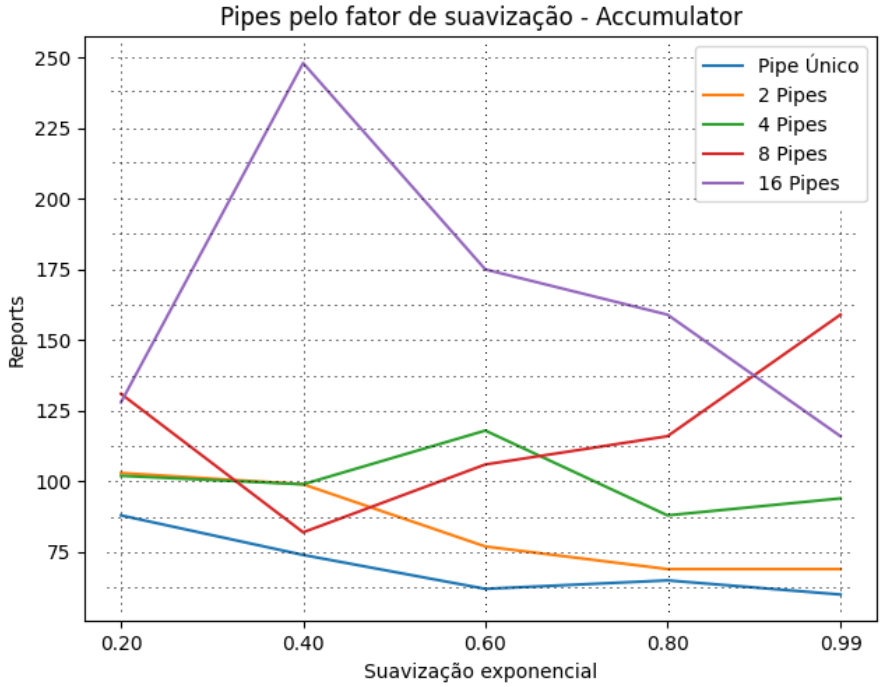


Figura 12 – *Reports* por *pipe* em relação ao fator de suavização do EWMA - *Accumulator*.

É possível identificar uma variação para a quantidade de *pipes*, uma vez que os fluxos na rede podem sofrer falhas, balanceamento de carga ou ajustes de desempenho e otimização durante a transmissão de pacotes e alternar entre os *pipes* durante a execução. Observamos que cada quantidade de *pipes* possui um fator de suavização mais adequado, resultando em uma redução no número de *reports* enviados para o plano de controle. Em particular, constatamos que um *switch* com 2 ou 4 *pipes* apresenta um desempenho superior com um fator de suavização de 0,8, enquanto que para um *switch* de 16 *pipes*, seria mais adequado utilizar um fator de suavização de 0,99.

É possível observar variações na quantidade de *pipes*, uma vez que os fluxos na rede podem sofrer falhas durante a transmissão de pacotes, alternar entre os *pipes* devido a técnicas de balanceamento de carga, ou ainda por outras razões operacionais durante a execução.

O cenário se inverte quando empregamos oito *pipes*. Nesse contexto, observamos que a quantidade de *reports* diminui quando o fator de suavização atribui maior peso aos dados históricos, que é de 0,4 neste caso. O pior cenário para o *Accumulator* em nossos experimentos ocorre quando se utiliza 16 *pipes* com um fator de suavização de 0,4, resultando em um pico de 248 *reports* enviados ao plano de controle. Apesar desse valor parecer substancial, ainda é consideravelmente menor em comparação com qualquer configuração de *pipes* por fator de suavização do *Local-pipe*, como ilustrado na Figura 13.

Um fator que pode explicar o pico observado está relacionado à divisão dos pacotes

entre os *pipes* em relação ao limite adaptativo. À medida que o número de *pipes* aumenta, o limite para cada *pipe* individual diminui. O limite adaptativo ajusta o peso atribuído ao último limite atingido pelo *switch*. Por exemplo, com a utilização de 16 *pipes* e um fator de suavização de 0,4, um *switch* pode ter atingido seu limite mais rapidamente e com mais frequência em comparação com outros cenários. Devido ao fator de suavização de 0,4, o peso maior atribuído a esses limites alcançados pode ter contribuído para um aumento no número de *reports*.

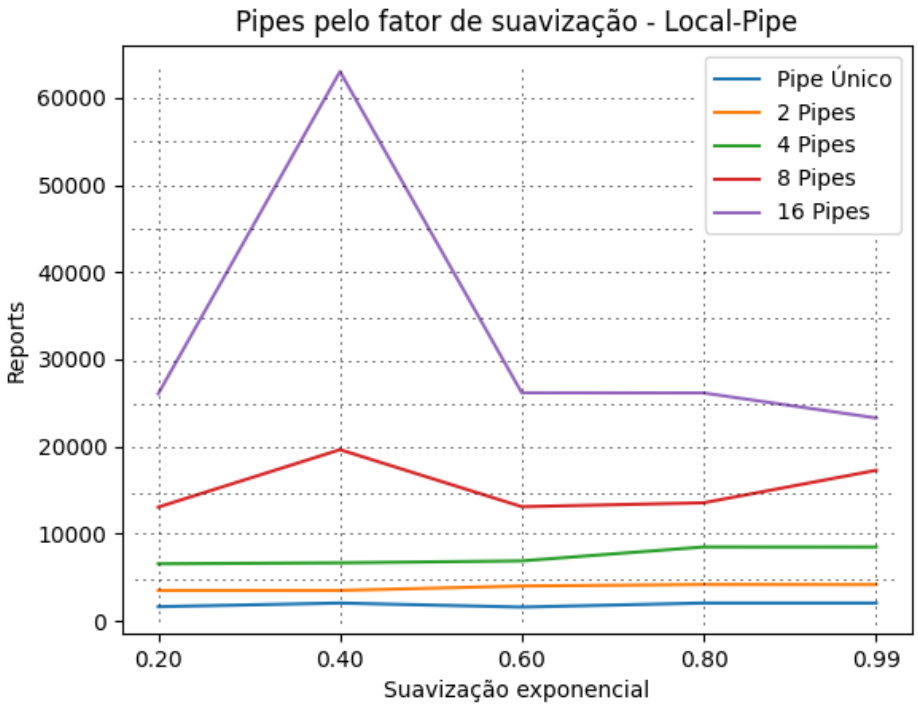


Figura 13 – *Reports* por *pipe* em relação ao fator de suavização do EWMA - *Local-pipe*.

Na Figura 13, apresentamos os resultados do mesmo experimento, porém aplicado à abordagem *Local-pipe*. É evidente que os *reports* exibem uma tendência mais linear, com pouca variação, à medida que aumentamos o fator de suavização, com exceção de um pico notável com 8 e 16 *pipes* para 0,4 de suavização. Além disso, é importante destacar que o *overhead* de comunicação é bem maior em comparação com o *Accumulator*, uma vez que o *Local-pipe* não incorpora um acumulador no *switch*.

Ao considerarmos o fator de suavização que resultou o menor valor *report* para o método *Local-pipe* (0,2 com 2 *pipes*) e comparando com o mesmo fator de suavização do método *Accumulator*, observamos um aumento significativo no número de *reports* para o *Local-pipe* de 3277% (*Accumulator* 103 *reports* e *Local-pipe* 3481 *reports*). Além disso, constatamos que a abordagem *Local-pipe* também é impactada em um *switch* que utiliza apenas um *pipe*, sofrendo um aumento na quantidade de *reports* em relação a abordagem *Accumulator*. Para um *switch* que utiliza somente um *pipe*, a abordagem *Local-pipe*, com

uma suavização de 0,2, gera 1618 *reports* ao plano de controle. Em contrapartida, a configuração com o *Accumulator*, também com suavização de 0,2, resulta em apenas 88 *reports* enviados ao plano de controle, representando uma redução de 18 vezes.

Em ambas as abordagens, observamos que cada quantidade de *pipes* apresenta sua suavização ideal, impossibilitando definir um valor universalmente ideal para qualquer configuração de *pipes*. No entanto, notamos que quanto maior a quantidade de *pipes*, maior a probabilidade de aumentar o *overhead* de comunicação. Exceto ao utilizar um acumulador em um dos *pipes*, como na Figura 12. Esse fenômeno ocorre devido ao limite menor por *pipe*. Assim, os *pipes* atingem seus limites locais mais rapidamente, resultando em um maior número de *reports*. Portanto, para um melhor desempenho, torna-se necessário identificar, para a quantidade de *pipes* desejada, se o fator de suavização utilizado terá mais peso aos dados recentes ou aos dados históricos.

Nos demais experimentos, optamos por utilizar um valor padrão de suavização em 0,8, independentemente do quantidade de *pipes*. Essa decisão foi tomada para evitar alterações constantes no fator.

6.3 Influência dos *Pipes*

Neste experimento, avaliamos o impacto da variação da quantidade de *pipes* mediante a utilização de um *switch* com um limite fixo, comparando-o com um *switch* com limite adaptativo. O experimento foi conduzido utilizando três *switches*, com um fator de suavização de 0,8, associado a um limite estabelecido em 10 mil pacotes. A Figura 14 apresenta um gráfico para abordagem *Accumulator*, em que o eixo X representa a quantidade de *pipes* empregada nos *switches*, enquanto o eixo Y indica a quantidade de *reports* encaminhados ao plano de controle.

Observando a Figura 14, notamos que ao empregar limites fixos, a relação entre a quantidade de *pipes* e a taxa de *reports* é inversamente direcionada. Ou seja, à medida que a quantidade de *pipes* aumenta, ocorre uma redução na quantidade de *reports*. Em contrapartida, quando os *switchs* utilizam o limite adaptativo, observa-se uma relação direta entre essas variáveis, apresentando um leve aumento na quantidade de *reports* à medida que a quantidade de *pipes* aumenta.

Entretanto, embora haja um aumento à medida que aumentamos a quantidade de *pipes*, a quantidade de *reports* ainda é extremamente menor quando comparada com a solução de limites fixos, constatando a vantagem do uso de limites adaptativos e dos *multi-pipes*. Por exemplo, considerando 4 *pipes* no *Accumulator*, observamos uma redução de aproximadamente 95% nos *reports* enviados ao coordenador, comparando o limite adaptativo com 88 *reports* e o limite fixo com 1672 *reports*.

Se analisarmos os valores do *Accumulator*, observamos um padrão crescente no número de *reports*, comparando o *switch* executando *multi-pipes*, com o *switch* executando 1

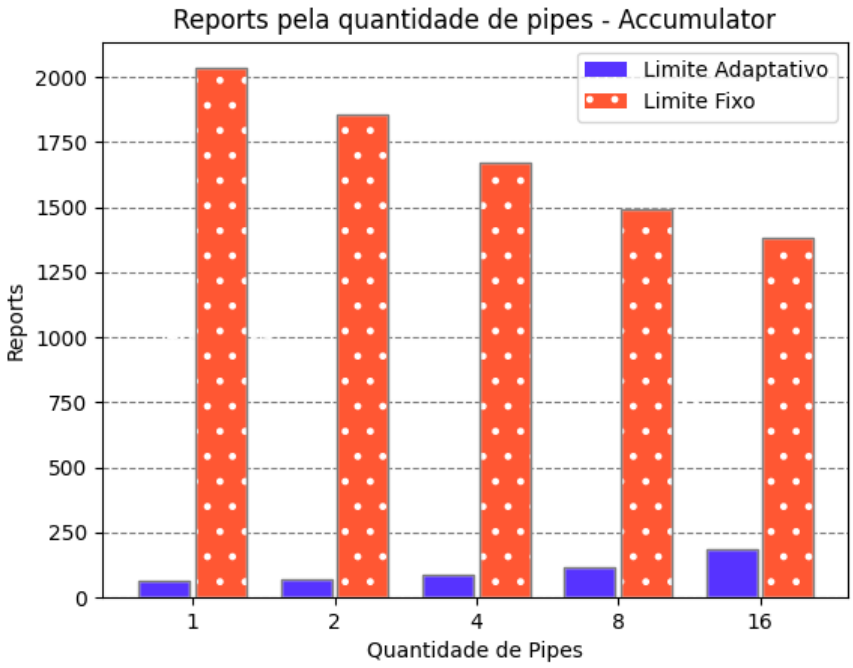


Figura 14 – Relação entre a quantidade de *reports* por *pipe* - *Accumulator*.

pipe. Embora haja um pequeno aumento, o número de *reports* para *multi-pipes* ainda é extremamente menor quando comparado com a solução de limites fixo e *pipe* único. Em outras palavras, mesmo no melhor cenário com limite fixo, onde há 1383 *reports* com 16 *pipes*, o número de *reports* enviados ao coordenador ainda é significativamente maior, superando o pior caso do *switch* com limites adaptativos (186 *reports*) em 643%.

Na abordagem *Local-pipe*, conforme ilustrado na Figura 15, o cenário é invertido. Conforme esperado, o *Local-pipe* apresenta uma quantidade excessivamente maior de *reports* ao plano de controle. À medida que aumentamos a quantidade de *pipes*, observamos um aumento da quantidade de *reports*, independentemente de estarmos utilizando um limite fixo no *switch* ou um limite adaptativo.

Embora ambas as soluções apresentem um aumento na quantidade de *reports* à medida que aumentamos o número de *pipes*, o limite fixo para o *switch* ainda tende a ter mais *reports* do que utilizando um limite adaptativo, como podemos observar aplicando oito ou 16 *pipes*. Como o limite global é dividido entre os *switches* de borda e, consequentemente, entre os *pipes*, quanto mais *pipes* são utilizados, menor é o limite local para cada um, levando a atingi-los mais rapidamente.

Ao analisarmos todos os resultados, com um limite fixo proveniente dos *switches multi-pipes*, tanto para o *hardware* que aproveitou os múltiplos *pipes* quanto para o *hardware* que aplicou apenas um *pipe*, destacamos uma melhor eficiência derivada da utilização *multi-pipes*. Pelos experimentos, a *baseline* alcançou 2035 *reports* ao plano de controle, enquanto nossa melhor metodologia, incorporando um acumulador no *switch multi-pipe*

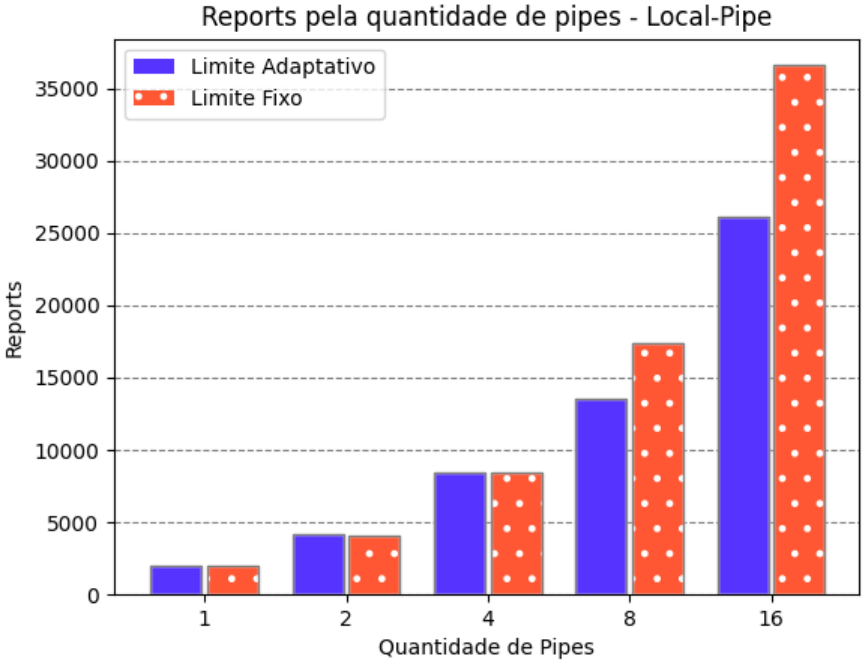


Figura 15 – Relação entre a quantidade de *reports* por *pipe* - *Local-pipe*.

com dois *pipes*, atingiu 69 *reports* ao plano de controle. Neste cenário, observamos uma notável redução de 96% na quantidade de relatórios.

Este resultado sugere uma eficiência significativamente aprimorada em relação à abordagem convencional de *switches* com um único *pipe*. Assim, como Harrison et al. (2018) e diversos estudos da literatura não utilizam dos *multi-pipes* em seus *switches* programáveis, observamos como seu uso pode contribuir para uma queda no *overhead* de comunicação entre os planos.

6.4 *Switches Network-Wide*

Neste experimento, investigamos se a quantidade de *switches* na rede tem algum impacto na detecção de fluxos HH. Todos os *switches* utilizados possuem 2 *pipes* com fator de suavização de 0,8 e limite de 10 mil pacotes. A Figura 16 apresenta um gráfico que representa a relação entre o quantidade de *reports* enviados ao plano de controle, no eixo Y, pela quantidade de *switches* presentes na rede, no eixo X.

Na figura, ao analisarmos o *Accumulator* com limites adaptativos, notamos um aumento na quantidade de *reports* enviados ao coordenador à medida que se aumenta a quantidade de *switches*. Diferentemente de quando os *switches* empregam limites fixos, onde não temos um padrão visível para o resultado apresentado.

Quando um limite fixo é aplicado ao *switch*, é perceptível uma redução na incidência de *reports* em dois casos específicos: com quatro e cinco *switches*. Essa redução pode ser

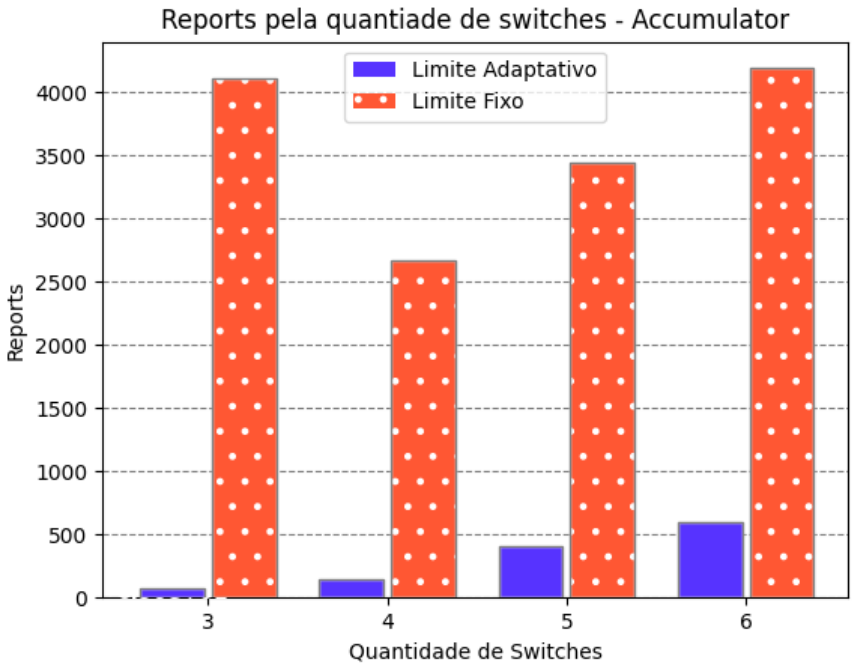


Figura 16 – Aumento na quantidade de *switches* em relação aos *reports* - *Accumulator*

atribuída à distribuição do limite do *switch*. O limite de pacotes para que seja identificado um HH (limite global), localizado no coordenador, é preestabelecido em 10 mil pacotes. Este limite é dividido pela quantidade de *switches* de borda existentes na rede. Portanto, ao considerar quatro *switches* na rede, o valor atribuído ao limite do *switch* será de 2500, enquanto que, com cinco *switches*, esse valor será reduzido para 2000.

Isso ocorre devido à redução no número de *reports* enviados ao plano de controle quando o limite global é divisível pela quantidade de *switches*. Por exemplo, consideremos um cenário com três ou seis *switches*. Ao dividir 10 mil pelo número de *switches*, obtemos um valor não inteiro, e o algoritmo utiliza o valor inteiro resultante. Assim, com três *switches*, o limite do *switch* será de 3333,3, que é arredondado para 3333. O valor fracionado após a vírgula resulta na geração de *reports* adicionais à medida que os pacotes são processados, levando a um aumento significativo na quantidade de *reports*, em comparação com uma divisão que resultasse em um valor inteiro.

Apesar da redução nos *reports* utilizando um quantidade de *switches* divisível pelo limite global, a implementação de limites adaptativos nos *switches* resulta em uma redução relevante na quantidade de *reports* em comparação com limite fixo. Comparando o melhor caso do limite adaptativo, que seria 69 *reports* ao plano de controle, observamos que o uso de limite fixo permanece alto, atingindo 4110 *reports*, alcançando um aumento de 5857% na quantidade de *reports*.

Mesmo se considerarmos o cenário mais favorável para limites fixos (quatro *switches*), em que registramos 2665 *reports*, a disparidade em relação aos limites adaptativos perma-

nece substancial, demonstrando um aumento de 1726% na quantidade de *reports* gerados utilizando *switches* com limites fixos. Neste caso específico, os *switches* com limite adaptativo apresenta apenas 146 *reports*.

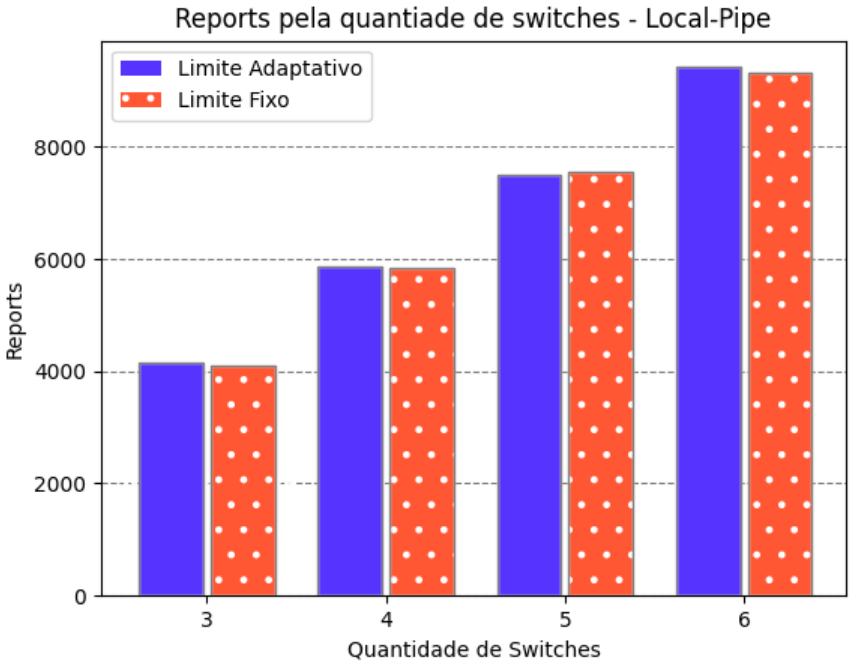


Figura 17 – Aumento na quantidade de *switches* em relação aos *reports* - *Local-Pipe*.

Além disso, ao examinar o cenário ideal de limites adaptativos (seis *switches*) em comparação com o mesmo cenário implementando limites fixos, evidencia-se que os limites fixos ainda resultam em uma elevada quantidade de *reports*, ocasionando um desempenho inferior quando comparado aos limites adaptativos. Isso leva a um número consideravelmente maior de *reports*, com um aumento de 598%. Neste caso, os *switches* com limite adaptativo registram 600 *reports*, enquanto os *switches* com limite fixo atingem 4189 *reports*.

Sendo assim, independentemente do número de *switches* presentes na rede ao adotar o método de limites fixos, as soluções com limites adaptativos mantêm uma baixa quantidade de *reports* enviados ao plano de controle.

No que diz respeito à estratégia *Local-pipe*, conforme evidenciado nos resultados representados na Figura 17, esta continua a ser a abordagem menos eficiente, destacando-se pela excessiva quantidade de *reports*. Tomando como exemplo a configuração com quatro *switches* aplicando limites adaptativos, observa-se que a quantidade de *reports* gerados pelo *Local-pipe* é 39 vezes maior do que o mesmo cenário utilizando a solução do *Accumulator*. Especificamente, o *Local-pipe* atingiu 5862 *reports*, enquanto o *Accumulator* registrou apenas 146 *reports*.

Além disso, é importante destacar que o *Local-pipe* apresenta uma tendência de ge-

rar um número crescente de *reports* à medida que a quantidade de *switches* aumenta. Isso demonstra que o *overhead* de comunicação entre os planos aumentará à medida que adicionamos mais *switches*, independentemente de utilizar limites adaptativos ou fixos.

6.5 Alternância dos *Pipes*

Neste experimento, visamos realizar uma análise da viabilidade e vantagem de empregar distintas configurações de *pipes* em relação aos diversos *switches* presentes na rede. Conduzimos este experimento em um cenário composto por três *switches*, cada um com uma limite de 10 mil pacotes. Por se tratar de uma alternância na quantidade de *pipes*, procedemos com dois experimentos fundamentais.

O primeiro envolveu a utilização de um único parâmetro de suavização para todos os *pipes*. Optamos pelo valor de 0,8, alinhando com a suavização empregada em experimentos anteriores. Essa escolha também foi baseada no fato de ser uma valor médio de suavização que satisfaz uma baixa quantidade de *reports* para a maioria dos *pipes*.

Já o segundo experimento contemplou a aplicação de distintos valores para o fator de suavização. Para determinar tais valores, nos baseamos no experimento descrito na Seção 6.2, no qual cada agrupamento de *pipes* apresentou um valor ideal de suavização, incorporado no *switch*. Essa abordagem se baseia na premissa de que diferentes *pipes* podem demandar valores específicos de suavização.

Realizamos os experimentos, para as abordagens *Accumulator* e *Local-pipe*, adotando configurações de valores distintas e arbitrárias para os *pipes* em cada *switch*. Os resultados desse experimento para a abordagem *Accumulator* estão apresentados na Figura 18. O eixo X representa a quantidade de *pipes* em cada *switch*. Por exemplo, no primeiro valor do eixo, temos "2-4-8", indicando que um *switch* é composto por 2 *pipes*, outro por 4 *pipes*, e o último por 8 *pipes*. O eixo Y representa a quantidade de *reports* associados ao plano de controle.

Observa-se que a diversificação no uso de diferentes combinações de *pipes* na rede é viável, com o limitante que a quantidade de *pipes* utilizada deve estar dentro dos padrões do *switch*. Na Figura 18, observamos duas colunas distintas referentes ao experimento. Ambas representam a quantidade de *reports* ao plano de controle, sendo que uma delas representa os resultados obtidos com valores de suavização variados (em azul), enquanto a outra demonstra o desempenho mantido sob um único valor de suavização, fixado em 0,8 (em laranja).

Identificamos nuances relevantes neste experimento. Uma delas é uma discreta variação na quantidade de *reports*, realizando uma análise comparativa entre ambas as colunas. De acordo com experimentos anteriores, constata-se que, na abordagem *Accumulator*, a presença de um maior número de *pipes* no *switch* está associada a uma elevada quantidade de *reports* encaminhados ao coordenador. No entanto, observamos que variações na

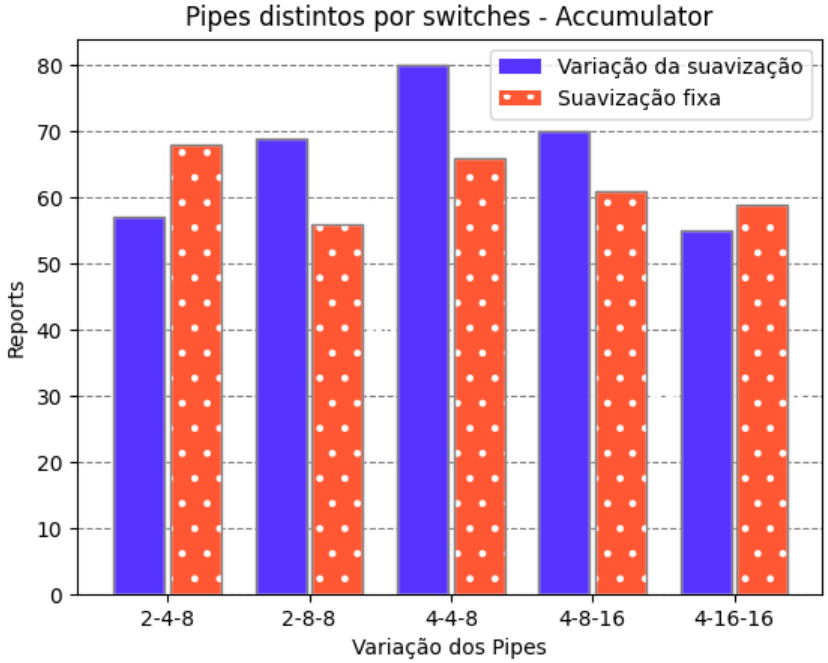


Figura 18 – Configurações distintas de *pipes* por *switch* - *Accumulator*.

distribuição da quantidade de *pipes* nos *switches* da rede não necessariamente resultam em uma correlação direta com um aumento na quantidade geral de *reports*.

Ao observarmos cada combinação de diferentes *pipes*, verificamos uma redução geral na quantidade de relatórios encaminhados ao coordenador, para este experimento; seja quando é utilizado uma suavização fixa, ou uma variação na suavização. Essa redução é derivada da comparação da média dos resultados obtidos no experimento da Seção 6.3 (que mantém a mesma quantidade de *pipes* para todos os *switches* na rede), com os resultados para cada configuração de *pipes* na rede deste experimento. Ou seja, por exemplo, para a variação da suavização, tendo em vista os resultados da Figura 14 que utiliza três contagens com dois, quatro e oito *pipes*, notamos um aumento de 27,8% na quantidade de *reports* em comparação com o uso dessas três contagens em *switches* distintos, ainda que ambos tenham o mesmo propósito.

Dessa forma, além de ser viável identificar os HH com diferentes contagens de *pipes* distribuídas em vários *switches*, essas variações de *pipes* demonstraram um desempenho ligeiramente superior. Além disso, uma avaliação desse experimento empregando um fator de suavização fixo resulta em uma média de 62 *reports* para as combinações do experimento. Entendemos que a variação para os experimentos de 6.3 com 6.5 vem da quantidade de *pipes* na rede, em conjunto com a mudança de limite para cada *switch*, assim como o tempo em que cada *switch* identificou seus distintos limites.

Também realizamos esses experimentos para a abordagem *Local-pipe* e apresentamos os resultados na Figura 19. Observamos que é viável empregar uma quantidade variada de

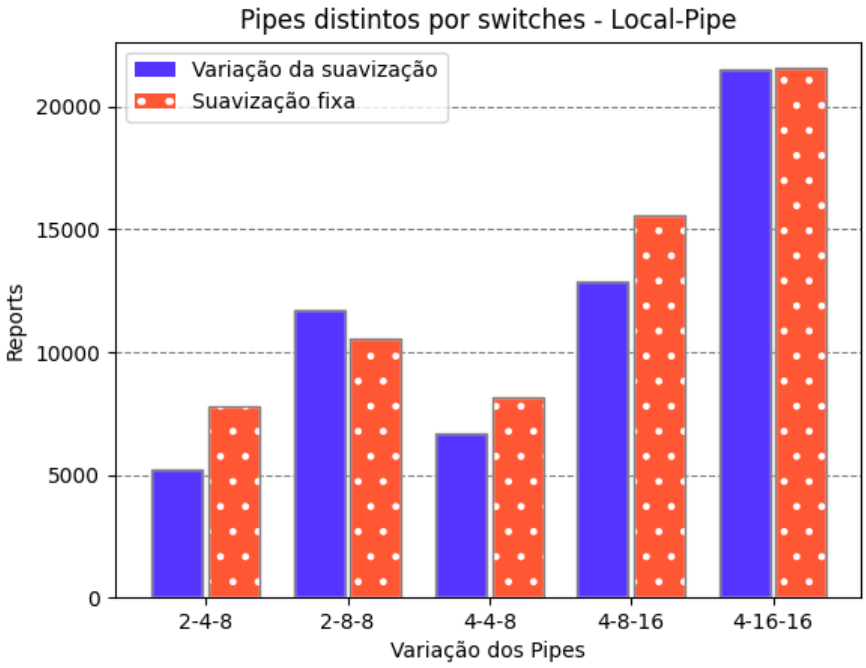


Figura 19 – Configurações distintas de *pipes* por *switch* - *Local-Pipe*.

pipes por *switch* na rede. Também constatamos que, nesta abordagem, o *overhead* de comunicação persiste sendo superior ao da abordagem *Accumulator*. Entretanto, evidencia-se uma melhora em relação aos experimentos utilizando a mesma quantidade de *pipes* nos diferentes *switches* da rede.

Após realizar uma análise média dos resultados provenientes dos experimentos com *pipes* distintos em comparação com *pipes* iguais, observou-se que a configuração de *pipes* distintos resultou em uma redução de 24% na quantidade de *reports* direcionados ao plano de controle; considerando a versão com variação nos parâmetros de suavização. No entanto, é importante ressaltar uma distinção relevante no caso específico do experimento com 4, 8 e 16 *pipes*, em que foi adotada uma suavização constante. Nesse cenário, houve uma maior quantidade de relatórios enviadas ao coordenador, contrastando com a média dos resultados do experimento anterior (Seção 6.3). A análise realizada para esta última combinação resulta um aumento de 18% na quantidade de *reports* em comparação com a média dos resultados do experimento na Seção 6.3.

Observamos que a aplicação com menor *overhead* de comunicação tende a ser aquela em que cada *switch* de borda da rede possui uma quantidade pequena e variável de *pipes* ativos. Por exemplo, em uma rede com três *switches* de borda, um *switch* poderia utilizar 2 *pipes*, outro 4 *pipes*, e o terceiro 8 *pipes*. Essa distribuição heterogênea de *pipes* ativos pode contribuir para uma eficiência otimizada na comunicação.

Conforme mencionado anteriormente e corroborado pelos dados apresentados na Tabela 2, todos os experimentos alcançaram um nível significativo de precisão, devido à

Tabela 2 – Resultados das métricas de avaliação *F1-Score* (2 pipes)

Métricas	**Accumulator Limite Fixo**	**Local-pipe Limite Fixo**	**Accumulator Limite Adap.**	**Local-pipe Limite Adap.**
F1-Score	0,996	0,996	0,996	0,996
Precisão	0,993	0,993	0,993	0,993
Recall	1,0	1,0	1,0	1,0

eficaz manipulação do *hash*, assim como um *F1-Score* elevado. A Tabela 2 apresenta os resultados das métricas de avalização para um cenário com 3 *switches* que utilizavam 2 *pipes*, porém para os experimentos com 4, 8 e 16 *pipes* os valores são os mesmos.

Nosso *baseline* (*switches* com apenas um *pipe* e limites fixos) também foi avaliado, resultando em um *F1-Score* de 0,99. Ao ser comparado com a Tabela 2, observa-se que tais valores se assemelham com os obtidos por *switches* que empregam apenas um *pipe*. Os experimentos envolvendo configurações distintas de *pipes* em diferentes *switch* também demonstraram o mesmo valor da precisão e *F1-Score*, comparáveis aos resultados obtidos em cenários com uma mesma quantidade de *pipes* em cada *swtich*.

Considerações finais

Realizações

Um artigo derivado desta dissertação foi publicado no SBRC 2024, intitulado "Detectando *Heavy Hitters* globalmente em dispositivos programáveis multi-pipes", enquanto uma versão resumida (*short*) foi publicada no IEEE NOMS 2024, sob o título "*Detecting Network-Wide Heavy Hitters in Multi-Pipe Programmable Devices*". Ambas foram bem recebidas pelos revisores, confirmando sua contribuição significativa para a literatura e as vantagens do uso de *multi-pipes* na detecção de *Heavy Hitters*.

Conclusão

A detecção de fluxos HH destaca-se como um método crucial no gerenciamento e defesa das redes modernas. Sua aplicação em *switches* programáveis *multi-pipes* é pouco explorada na literatura, apesar de oferecer vantagens consideráveis. Dessa forma, neste estudo, projetamos um algoritmo eficiente e implementamos um emulador capaz de detectar HH em *Network-wide* diretamente no plano de dados, através de *switches* programáveis *multi-pipes*.

Nossa análise utilizando tráfego real revela uma redução significativa no *overhead* de comunicação, entre o plano de dados e o plano de controle, para a detecção de HH em *Network-wide*, mitigando potenciais sobrecargas, latência e atrasos. Essa redução é alcançada através da combinação de *multi-pipes* com a utilização de limites adaptativos, mantendo um alto *F1-Score* comparando-o com nossa *baseline*, a abordagem comumente usada com um único *pipe*.

Ao comparar nossas duas abordagens, observamos que para atingir os melhores resultados é crucial empregar um acumulador no *switch*, para que este se comunique com o coordenador, além do uso de limites adaptativos. Se um limite fixo for utilizado, observamos um aumento médio de 92% no *overhead* de comunicação entre os planos de dados

e o plano de controle. O limite fixo, neste caso, seria como as abordagens atuais que não aplicam a detecção com *multi-pipes*.

Caso contrário, o aumento na quantidade de *pipes* resulta em um aumento no *overhead* de comunicação entre os planos. Portanto, a abordagem *Accumulator* com limites adaptativos demonstrou a capacidade de manter o mesmo *F1-Score* em comparação com a abordagem de um único *pipe*, ao mesmo tempo em que reduziu o *overhead* de comunicação em 96,65%. Além disso, identificamos a viabilidade de utilizar diferentes quantidades de *pipes* em *switches* na rede, sem a restrição de que todos os *switches* possuam a mesma quantidade de *pipes*, tornando essa abordagem mais aplicável em um ambiente real de distribuição de fluxos na rede. Juntamente com uma precisão elevada, nossa solução proporcionou uma detecção mais ágil de ocorrências na rede.

Em relação aos limites adaptativos, observamos que o fator de suavização ideal deve variar de acordo com a quantidade de *pipes* utilizados no *switch*, para obter melhores resultados. Portanto, é importante analisar e escolher o melhor fator de suavização para melhores resultados. É crucial essa adaptação, independentemente da quantidade de *pipes* escolhidos, para minimizar a quantidade de *reports* entre os planos.

Para os trabalhos futuros, planejamos implementar a abordagem *Accumulator* em linguagem P4, com o objetivo de executá-la em um *hardware* real Tofino. Pretendemos explorar de maneira eficaz os *multi-pipes* disponíveis nesse *hardware* para otimizar a performance da solução proposta. Além disso, pretendemos aplicar uma técnica distinta para detecção, conhecida como *Inter Packet Gap* (IPG) (SINGH et al., 2022), que utiliza uma janela deslizante para a contagem de pacotes dos fluxos HH, e compará-la com nossa solução ideal. Adicionalmente, consideramos abordar a questão da adaptação do limite global, em cenários nos quais o valor absoluto do limite global não corresponde ao valor real. Nesse contexto, será necessário encontrar um valor adequado para o limite global, a fim de aprimorar a precisão e eficácia do sistema, talvez possuindo uma porcentagem para quantidade de fluxos.

Referências

AGRAWAL, A.; KIM, C. Intel tofino2–a 12.9 tbps p4-programmable ethernet switch. In: IEEE COMPUTER SOCIETY. **2020 IEEE Hot Chips 32 Symposium (HCS)**. [S.l.], 2020. p. 1–32.

BASAT, R. B. et al. Designing heavy-hitter detection algorithms for programmable switches. **IEEE/ACM Transactions on Networking**, IEEE, v. 28, n. 3, p. 1172–1185, 2020.

_____. Network-wide routing-oblivious heavy hitters. In: **Proceedings of the 2018 Symposium on Architectures for Networking and Communications Systems**. [S.l.: s.n.], 2018. p. 66–73.

_____. Memento: Making sliding windows efficient for heavy hitters. In: **Proceedings of the 14th International Conference on Emerging Networking EXperiments and Technologies**. [S.l.: s.n.], 2018. p. 254–266.

BASAT, R. B.; EINZIGER, G.; TAYH, B. Cooperative network-wide flow selection. In: IEEE. **2020 IEEE 28th International Conference on Network Protocols (ICNP)**. [S.l.], 2020. p. 1–11.

BEN-BASAT, R. et al. Efficient measurement on programmable switches using probabilistic recirculation. In: IEEE. **2018 IEEE 26th International Conference on Network Protocols (ICNP)**. [S.l.], 2018. p. 313–323.

BOSSHART, P. et al. P4: Programming protocol-independent packet processors. **ACM SIGCOMM Computer Communication Review**, ACM New York, NY, USA, v. 44, n. 3, p. 87–95, 2014.

BROADCOM. **Tomahawk3 - BCM56980 12.8 Tb/s Multilayer Switch**. 2019. Disponível em: <https://docs.broadcom.com/doc/56980-DS>.

CAIDA. **The caida ucsd anonymized internet traces - 20190117.** 2019. Disponível em: <https://docs.broadcom.com/doc/12398014>. Acesso em: 8 abril. 2024.

CALLEGARI, C. et al. Detecting heavy change in the heavy hitter distribution of network traffic. In: IEEE. **2011 7th International Wireless Communications and Mobile Computing Conference**. [S.l.], 2011. p. 1298–1303.

CAMERA, P. E.; ZANETTI, A. B. Introdução à linguagem de programação p4, o futuro das redes. **Sociedade Brasileira de Computação**, 2019.

CHIESA, M.; VERDI, F. L. Network monitoring on multi-pipe switches. **Proc. ACM Meas. Anal. Comput. Syst.**, Association for Computing Machinery, New York, NY, USA, v. 7, n. 1, mar 2023. Disponível em: <https://doi.org/10.1145/3579321>.

CORMODE, G. Continuous distributed monitoring: a short survey. In: **Proceedings of the first international workshop on algorithms and models for distributed event processing**. [S.l.: s.n.], 2011. p. 1–10.

CORRÊA, J. L.; PROTO, A.; CANSIAN, A. M. Modelo de armazenamento de fluxos de rede para análises de tráfego e de segurança. In: SBC. **Anais do VIII Simpósio Brasileiro em Segurança da Informação e de Sistemas Computacionais**. [S.l.], 2008. p. 73–86.

CRUZ, M. A. d. et al. Detecção online de agregações hierárquicas bidimensionais de fluxos em redes definidas por software. Universidade Federal de Goiás, 2014.

DANG, H. T. et al. Netpaxos: Consensus at network speed. In: **Proceedings of the 1st ACM SIGCOMM Symposium on Software Defined Networking Research**. [S.l.: s.n.], 2015. p. 1–7.

DING, D. et al. An incrementally-deployable p4-enabled architecture for network-wide heavy-hitter detection. **IEEE Transactions on Network and Service Management**, IEEE, v. 17, n. 1, p. 75–88, 2020.

GAVAZZA, J. A. et al. Implementação de um switch em p4 com suporte simultâneo a múltiplas arquiteturas de internet do futuro. In: SBC. **Anais do X Workshop de Pesquisa Experimental da Internet do Futuro**. [S.l.], 2019. p. 13–18.

HANDLEY, M. et al. Re-architecting datacenter networks and stacks for low latency and high performance. In: **Proceedings of the Conference of the ACM Special Interest Group on Data Communication**. [S.l.: s.n.], 2017. p. 29–42.

HARRISON, R. et al. Network-wide heavy hitter detection with commodity switches. In: **Proceedings of the Symposium on SDN Research**. [S.l.: s.n.], 2018. p. 1–7.

_____. Carpe elephants: Seize the global heavy hitters. In: **Proceedings of the Workshop on Secure Programmable Network Infrastructure**. [S.l.: s.n.], 2020. p. 15–21.

HOFSTEDE, R. et al. Flow monitoring explained: From packet capture to data analysis with netflow and ipfix. **IEEE Communications Surveys & Tutorials**, IEEE, v. 16, n. 4, p. 2037–2064, 2014.

INTEL. **Intel Tofino 2 Intelligent Fabric Processor Brief**. 2022. Disponível em: <https://www.intel.com/content/www/us/en/products/network-io/programmable-ethernet-switch/tofino-3-brief.html>.

IVKIN, N. et al. Qpipe: Quantiles sketch fully in the data plane. In: **Proceedings of the 15th International Conference on Emerging Networking Experiments And Technologies**. [S.l.: s.n.], 2019. p. 285–291.

JIN, X. et al. Netcache: Balancing key-value stores with fast in-network caching. In: **Proceedings of the 26th Symposium on Operating Systems Principles**. [S.l.: s.n.], 2017. p. 121–136.

KIM, D. et al. Generic external memory for switch data planes. In: **Proceedings of the 17th ACM Workshop on Hot Topics in Networks**. [S.l.: s.n.], 2018. p. 1–7.

LEE, J. et al. Stateful layer-4 load balancing in switching asics. In: **Proceedings of the SIGCOMM Posters and Demos**. [S.l.: s.n.], 2017. p. 133–135.

LI, F. et al. Effective network-wide traffic measurement: A lightweight distributed sketch deployment. **IEEE Conference on Computer Communications**, 2023.

LIBERATO, A. et al. Rdna: Residue-defined networking architecture enabling ultra-reliable low-latency datacenters. **IEEE Transactions on Network and Service Management**, IEEE, v. 15, n. 4, p. 1473–1487, 2018.

LIN, Y.-B.; HUANG, C.-C.; TSAI, S.-C. Sdn soft computing application for detecting heavy hitters. **IEEE Transactions on Industrial Informatics**, IEEE, v. 15, n. 10, p. 5690–5699, 2019.

LIU, Z. et al. One sketch to rule them all: Rethinking network flow monitoring with univmon. In: **Proceedings of the 2016 ACM SIGCOMM Conference**. [S.l.: s.n.], 2016. p. 101–114.

LOPES, J. P.; PINTO, P. E.; OLIVEIRA, F. d. S. Estruturas de dados probabilísticas para representação de conjuntos. In: SBC. **Anais do I Encontro de Teoria da Computação**. [S.l.], 2016. p. 899–902.

LOPES, J. P. A. et al. Estruturas de dados probabilísticas aplicadas à representação implícita de grafos. Universidade do Estado do Rio de Janeiro, 2017.

MACHADO, D. et al. Avaliação de desempenho de heavy hitters utilizando p4 e xdp. In: SBC. **Anais da XVII Escola Regional de Redes de Computadores**. [S.l.], 2019. p. 122–123.

MCKEOWN, N. et al. Openflow: enabling innovation in campus networks. **ACM SIGCOMM computer communication review**, ACM New York, NY, USA, v. 38, n. 2, p. 69–74, 2008.

NETO, M. J. d. S. Detecção de ataque ddos em sdn utilizando entropia e machine learning. Universidade de Brasília, 2021.

SANKARAN, A. et al. On the nature of the superspreaders. **Advances in colloid and interface science**, Elsevier, v. 263, p. 1–18, 2019.

SANTOS, E. R. et al. Aplicações de monitoramento de tráfego utilizando redes programáveis ebpf. In: SBC. **Anais do XXXVII Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos**. [S.l.], 2019. p. 417–430.

SENE, R. **Estruturas de Dados Probabilísticas**. 2020. Disponível em: <https://www.zup.com.br/blog/estruturas-de-dados-probabilisticas>. Acesso em: 10 fevereiro. 2023.

SILVA, M. V. B. d. Prevendo e identificando fluxos elefantes em redes de ponto de troca de tráfego com suporte à programabilidade. Universidade Federal do Rio Grande do Sul, 2019.

SILVA, M. V. B. da et al. Identificação de fluxos elefantes em redes de ponto de troca de tráfego com suporte à programabilidade p4. In: SBC. **Anais do XXXVI Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos**. [S.l.], 2018. p. 1131–1144.

SILVEIRA, F. Á. F. Minimização do conjunto de observadores para geração de matrizes de tráfego. Universidade Federal do Espírito Santo, p. 1–66, 2017.

SINGH, S. K. et al. Hh-ipg: Leveraging inter-packet gap metrics in p4 hardware for heavy hitter detection. **IEEE Transactions on Network and Service Management**, IEEE, 2022.

SIVARAMAN, V. et al. Heavy-hitter detection entirely in the data plane. In: **Proceedings of the Symposium on SDN Research**. [S.l.: s.n.], 2017. p. 164–176.

SONG, C. H. et al. Fcm-sketch: generic network measurements with data plane support. In: **Proceedings of the 16th International Conference on emerging Networking EXperiments and Technologies**. [S.l.: s.n.], 2020. p. 78–92.

SOUSA, N. F. S. de; ROTHENBERG, C. E. Softwarização em redes: Do plano de dados ao plano de orquestração. Universidade Estadual de Campinas, 2017.

TOURRILHES, J. et al. Sdn and openflow evolution: A standards perspective. **Computer**, IEEE, v. 47, n. 11, p. 22–29, 2014.

TURKOVIC, B.; OOSTENBRINK, J.; KUIPERS, F. Detecting heavy hitters in the data-plane. **arXiv preprint arXiv:1902.06993**, 2019.

VILELA, G. S. Caracterização de tráfego utilizando classificação de fluxos de comunicação. **Mestre em ciências em engenharia de sistemas e computação, Universidade Federal do Rio de Janeiro, Rio de Janeiro, Brasil**, 2006.

VIVO, M. D. et al. A review of port scanning techniques. **ACM SIGCOMM Computer Communication Review**, ACM New York, NY, USA, v. 29, n. 2, p. 41–48, 1999.

WANG, C. et al. Skyshield: A sketch-based defense system against application layer ddos attacks. **IEEE Transactions on Information Forensics and Security**, IEEE, v. 13, n. 3, p. 559–573, 2017.

WHEELER, B. Tomahawk 4 switch first to 25.6 tbps. **Microprocessor Report**, 2019.

XIA, W. et al. A survey on software-defined networking. **IEEE Communications Surveys & Tutorials**, IEEE, v. 17, n. 1, p. 27–51, 2014.

YAN, Q. et al. Software-defined networking (sdn) and distributed denial of service (ddos) attacks in cloud computing environments: A survey, some research issues, and challenges. **IEEE communications surveys & tutorials**, IEEE, v. 18, n. 1, p. 602–622, 2015.