Detectando Heavy Hitters globalmente em dispositivos programáveis multi-pipes*

Thiago Henrique Silva Rodrigues[1], **Thiago Caproni Tavares**[2], **Fábio Luciano Verdi**[3]

[1]Departamento de Computação (Dcomp) – Universidade Federal de São Carlos (UFSCar) São Carlos – SP – Brasil

[2]Coordenação de Ensino – Instituto Federal do Sul de Minas Gerais (IFSULDEMINAS) Poços de Caldas – MG – Brasil

[3]Departamento de Computação (Dcomp) – Universidade Federal de São Carlos (UFSCar) Sorocaba – SP – Brasil

tigohsr@gmail.com, thiago.caproni@ifsuldeminas.edu.br, verdi@ufscar.br

Resumo. *Uma forma de contribuir para a gestão de redes, envolve detectar fluxos de tráfego de grande impacto, conhecidos como "Heavy Hitters". Heavy Hitters são fluxos que conduzem a maior parcela de bytes transmitidos pela rede, consequentemente consumindo mais recursos. O uso de hardware programável, como switches e DPUs, permite a detecção desses fluxos diretamente no plano de dados da rede. Embora a literatura revele uma extensa análise da detecção em switches de pipe único, este estudo apresenta duas abordagens para identificar Heavy Hitters em switches programáveis com múltiplos pipes. Uma abordagem possui um acumulador no switch, que centraliza os dados provenientes de todos os pipes e se comunica com o plano de controle. Já na outra, as comunicações com o plano de controle são independentes para cada pipe. Ambas abordagens foram desenvolvidas, e validadas através de um emulador, demonstrando eficácia e melhoria na detecção em switches multi-pipes, comparado a switches de pipe único.*

Abstract. *A way to contribute to network management involves detecting high-impact traffic flows, known as "Heavy Hitters". Heavy Hitters are flows that account for the majority of bytes transmitted over the network, consequently consuming more resources. The use of programmable hardware, such as switches and DPUs, enables the detection of these flows in-line rate, meaning direct detection in the network's data plane. While the literature reveals extensive analysis of detection in single-pipe switches, this study presents two approaches to identify Heavy Hitters in programmable switches with multiple pipes. One approach features an accumulator in the switch that centralizes data from all pipes and communicates with the control plane. In the other approach, communications with the control plane are independent for each pipe. Both approaches were developed and validated through an emulator, demonstrating effectiveness and improvement in detection in multi-pipe switches compared to single-pipe switches.*

*Uma versão reduzida (short) deste artigo foi aceita no IEEE NOMS 2024 (https://leris.dcomp.ufscar.br/three-papers-accepted-in-ieee-noms-2024/).

1. Introdução

O processo de monitoramento de rede desempenha um papel crucial na identificação de padrões e anomalias, como na Detecção de Ataques de Negação de Serviço Distribuídos [Ding et al. 2020]. Operadores de rede frequentemente monitoram o tráfego em busca de fluxos maliciosos ou sobrecarga [Hofstede et al. 2014]. Sendo assim, uma prática comum é a identificação dos *Heavy Hitters* (HH). Esses fluxos representem menos de 10% dentre os fluxos que passam a rede, porém, consomem a maior parte desses recursos, e monitorá-los pode contribuir para uma gestão eficiente da rede [Lin et al. 2019, Vilela 2006].

Portanto, torna-se essencial distinguir e reagir prontamente a esses fluxos. Caso contrário, podem comprometer o tráfego de fluxos leves que compartilham a mesma rede, apresentando o potencial de esgotar rapidamente os recursos dos dispositivos de rede [Silva 2019, Chiesa and Verdi 2023]. Uma forma é detectar os HH no plano de dados, pois reduz o *overhead* de comunicação e a transferência de dados entre os planos de dados e controle [Machado et al. 2019]. Além de possibilitar a análise de informações de todos os *switches* de uma rede [Ding et al. 2020].

Para isto a utilização de *switches* programáveis de alta velocidade seria mais eficaz, pois oferece um controle melhor do plano de dados, permitindo a rápida implementação de algoritmos de rede, assim como soluções de monitoramento mais sofisticadas diretamente no *hardware* do *switch* [Basat et al. 2020]. Estes *switches*, utilizam o conceito de *multi-pipes* para processar pacotes de forma paralelizada [Chiesa and Verdi 2023]. Na qual, cada *pipe* opera com 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*.

No entanto, a detecção precisa de fluxos HH em redes com *hardware multi-pipes* enfrenta desafios significativos. Trabalhos na literatura, como [Ding et al. 2020, Harrison et al. 2018, Sivaraman et al. 2017, Turkovic et al. 2019, Basat et al. 2018b], não detectam fluxos HH nesse contexto. Isso se deve à complexidade dos *switches multi-pipes*, pois os fluxos podem ingressar em qualquer *pipe* e alternar entre eles durante sua duração. Devido à independência dos *pipes*, essa ação gera uma detecção incorreta da quantidade real de pacotes por fluxo passados pelo *switch*, resultando em uma contagem errada de um fluxo HH na rede.

Esse problema é agravado quando a detecção é realizada em toda a rede (*Network-Wide*), considerando que a contagem de um fluxo HH é executada em múltiplos *switches* na rede. O termo *Network-Wide* implica em um monitoramento global distribuído na rede, envolvendo uma coleta parcial distribuída nos *switches* espalhados pela rede e uma contagem global no plano de controle. Dessa forma, esses dados são agrupados e uma decisão é tomada [Basat et al. 2018a, Tang et al. 2020].

Estendemos um dos trabalhos presentes na literatura, destacado como um dos principais pela detecção de fluxos HH em *Network-Wide*. Dessa forma, nosso trabalho detecta globalmente HH em *switches* de borda programáveis *multi-pipe*, empregando uma técnica de limites adaptativos. Desenvolvemos um emulador para a detecção, e elaboramos duas abordagens. Ambas abordagens incluem um contador local em cada *pipe*, porém a abordagem "*Accumulator*" possui um acumulador local do *switch* que calcula a somatória das contagens de cada *pipe*, e comunica com o plano de controle quando esse

limite for atingido. Enquanto a abordagem "*Local-pipe*" não utiliza um acumulador no *switch*, e assim cada *pipe* se comunica individualmente com o plano de controle.

Conduzimos experimentos para ambas as abordagens, comparando o uso de um limite adaptativo no *switch* com um limite fixo, além da diferença do uso de um *switch multi-pipe* com um *switch* de único *pipe*. Pretendemos realizar a detecção minimizando a comunicação entre o plano de dados e de controle, uma característica priorizada por [Harrison et al. 2018]. As principais contribuições do artigo incluem:

- Uma estratégia eficaz para detectar e mitigar fluxos HH em *switches* de borda *multi-pipes*;
- Adaptação da detecção para ambientes com uma escala de rede maior;
- Uma abordagem flexível que permite ajustar os limites de detecção, reduzindo o *overhead* de comunicação entre os *switches* e o coordenador;
- Uma estratégia que mantém um alto valor de F1-Score, garantindo precisão na detecção de HH.

O restante deste artigo se organiza da seguinte maneira. A Seção 2 apresenta estudos e trabalhos relacionados que contribuem para o contexto deste trabalho. A Seção 3 aborda a motivação que impulsionou este estudo, bem como o funcionamento de um *switch multi-pipe* e nossas estratégias de detecção de HH. A Seção 4 apresenta os resultados obtidos na avaliação, incluindo uma comparação com um *switch* de *pipe* único. Por fim, a Seção 5 sintetiza as principais conclusões e contribuições deste artigo.

2. Trabalhos relacionados

Nos últimos anos, houve uma evolução gradual nas soluções do plano de dados, particularmente na área de detecção de HH. Uma contribuição fundamental para a detecção de HH em toda a rede é o trabalho de [Harrison et al. 2018], onde a programação ocorre diretamente em *switches* de rede dentro do plano de dados. Os autores empregam limites adaptativos baseados em chaves para reduzir o *overhead* de comunicação entre *switches* e o coordenador, utilizando tabelas *hash* para contar HH por fluxo. Os *switches* de borda realizam a contagem de pacotes por meio de chaves nas tabelas *hash*, utilizando os endereços IP de origem/destino e as portas. Quando um limite local é atingido, o *switch* comunica ao coordenador, e quando o limite global é atingido, o coordenador calcula a média móvel ponderada exponencialmente, refinando o limite. Nosso trabalho se baseia nesse conceito, aplicando limites adaptativos, detecção em toda a rede e comunicação entre os planos de dados e controle. No entanto, ao contrário dos autores, estaremos utilizando *switches multi-pipes*.

O estudo de [Ding et al. 2020] permite a implantação incremental de novos *switches* programáveis na rede, otimizando simultaneamente a eficácia da operação de monitoramento por meio de intervalos de tempo para detecção, para detectar HH. Cada *switch* mantém dinamicamente uma lista dos fluxos que excedem o limite de amostragem dinâmico. Após o intervalo de tempo definido, verifica-se se algum *switch* atingiu seu limite local; em caso afirmativo, o *switch* relata ao coordenador central. Assim, realiza-se uma varredura nos *switches* da rede, coletando os quatro campos do cabeçalho TCP/IP dos fluxos, permitindo estimar o volume total e identificar os HH na rede. Os valores são então redefinidos, iniciando um novo intervalo de tempo para a contagem. Os autores

empregam uma técnica de limites adaptativos diferente do nosso trabalho, além de não utilizarem *multi-pipes* e abordam um método de uma única notificação ao controlador.

No estudo [Chiesa and Verdi 2023], os autores investigam o impacto da implantação de soluções de monitoramento existentes em *switches multi-pipes*. Posteriormente, desenvolvem um sistema denominado PipeCache, o qual adapta soluções de monitoramento para arquiteturas *multi-pipes*. O objetivo é realizar a detecção de HH mantendo o cache baixo, a fim de alcançar resultados de memória e precisão próximos a um *switch* de *pipe* único. O processo utilizado confirma a complexidade de implementar soluções de monitoramento em *switches* de *pipe* único para ambientes *multi-pipe*. Concluindo que soluções existentes, como as mencionadas, exigem oito vezes mais memória no *chip* para atingir uma precisão equivalente à implementação em um *pipe* único. Ao contrário do PipeCache, nossa solução concentra-se na detecção de HH em toda a rede e no impacto das notificações do plano de dados para o plano de controle, mantendo a mesma precisão de um *switch* de *pipe* único. Além disso, armazenamos contagens de tráfego em vários *pipes* e, posteriormente, no plano de controle, não apenas em um único *pipe*.

Tabela 1

Artigos	Ano	Multi-pipe	Plano de Dados	Plano de Controle	Netwrk-Wide	Limite Adaptativo
Este trabalho	2024	✓	✓	✓	✓	✓
[Harrison et al. 2018]	2018	✗	✓	✓	✗	✓
[Ding et al. 2020]	2020	✗	✓	✗	✗	✗
[Chiesa and Verdi 2023]	2023	✓	✓	✗	✗	✗

A Tabela 1 fornece uma análise detalhada das divergências entre nossa abordagem e outras propostas mencionadas na literatura. Definimos métricas distintas para destacar essas diferenças, evidenciando uma abordagem abrangente e inovadora do nosso trabalho, que atende a uma variedade de requisitos não abordados por outros estudos. Em seguida, na Seção 3, exploramos o funcionamento de um *switch multi-pipe* e apresentamos as duas abordagens desenvolvidas para este trabalho.

3. Heavy Hitters em *Multi-Pipes*

Nosso trabalho se destaca como o primeiro estudo prático, no contexto *Network-Wide*, a aplicar a detecção de HH em *switches* de borda *multi-pipes*, analisando a maneira mais eficiente de realizar essa detecção e quantificando seu impacto em comparação com uma arquitetura baseada em um único *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 and Kim 2020].

Embora dispositivos *multi-pipes* busquem aprimorar o desempenho de *hardwares* programáveis, é importante destacar que aplicações projetadas para operar em *hardwares* de único *pipe*, como detecção de HH e balanceamento de carga, podem apresentar irregularidades ao considerar uma arquitetura *multi-pipe*. Isso pode resultar em contagens incorretas ou no mau funcionamento da aplicação. Portanto, torna-se necessário realizar ajustes para garantir um funcionamento ideal.

Hardwares programáveis *multi-pipes* operam diferentemente de *hardwares* de *pipe* único, principalmente devido à natureza isolada de cada *pipe*. Cada *pipe* é uma unidade de processamento separada e paralela dentro de um *switch* que podem ser programado. Esses *pipes* permitem que o *switch* processe múltiplos fluxos de dados simultaneamente, aprimorando a capacidade de processamento e a eficiência do dispositivo. Para ilustrar essa diferença, criamos um exemplo ilustrativo retratado na Figura 1, que demonstra um *switch multi-pipe*. Nesse cenário, temos um coordenador (plano de controle) e um *switch* (plano de dados). O *switch* possui quatro *pipes* e está recebendo um pacote da rede.

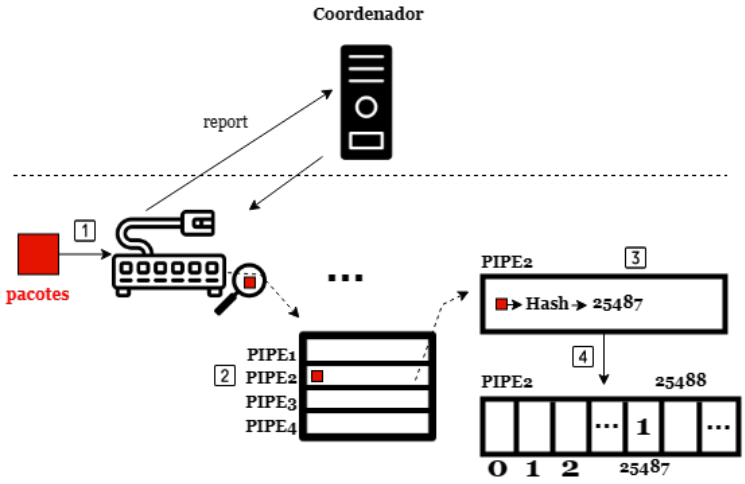

Figura 1. Funcionamento da rede para detectar os HH.

Este cenário ilustra uma situação de uma contagem de pacotes para identificar um fluxo de HH, semelhante à nossa abordagem. Inicialmente, os pacotes presentes na rede chegam a um *switch* de borda(1) e são direcionados para um dos *pipes* disponíveis. No exemplo, os pacotes são encaminhados para o "*pipe* 2"(2). Uma chave *hash* é gerada a partir de quatro campos do cabeçalho TCP/IP do pacote(3). Com a chave gerada, a contagem é iniciada em uma tabela de *hash*, na posição correspondente à chave obtida(4).

Considere que o limite para um fluxo HH seja de 100 pacotes e que o fluxo passado pelo "*pipe* 2" (fluxo f1) possua um total de 200 pacotes. Se esses 200 pacotes estivessem alocados em um único *pipe*, o HH seria contabilizado duas vezes. Entretanto, em um cenário que o "*pipe* 2" tenha acumulado 50 pacotes do f1, e devido a fatores como balanceamento de carga, oscilações na rede ou falhas, ocorre uma reconfiguração e o f1 passa a ser encaminhado pelo "*pipe* 3", onde é encaminhado outros 50 pacotes. Como resultado, a detecção de HH não ocorrerá, pois um *pipe* não possui conhecimento das informações armazenadas em outro *pipe*. É importante destacar que qualquer alteração nesses campos, estatisticamente geraria outro *hash*, movendo-a para outra posição na tabela.

Este problema se torna ainda mais complexo quando consideramos uma detecção *Network-Wide*, em que a contagem de um fluxo HH é realizada em *switches* distribuídos por toda a rede. Neste contexto, além da complexidade dos *multi-pipes*, surge a questão da divisão dos pacotes dos fluxos entre diferentes *switch* na rede. Vale ressaltar que nossa detecção de fluxos HH é realizada em *switches* de borda.

Em nosso trabalho, desenvolvemos duas abordagens de detecção: ***Local-pipe*** e

Accumulator. Resumidamente, a abordagem ***Local-pipe*** notifica o coordenador sempre que o limite local de um *pipe*, para determinado fluxo, é alcançado. Já a abordagem ***Accumulator*** possui um acumulador local no *switch*, em um de seus *pipes*, que recebe a contagem dos limites dos *pipes* quando são alcançados e notifica o coordenador quando atinge o limite do *switch*.

O coordenador possui um limite global, e esse valor é dividido entre todos os *switches* de borda na rede, resultando em um limite para cada *switch*. O "limite do *switch*" é dividido entre todos os *pipes* desse *switch*. Consequentemente, cada *pipe* possui um limite local do *pipe*. Se o limite local do *switch* for múltiplo da quantidade de *pipes* cada *pipe* receberá uma fração desse valor; caso contrário, os valores que excedem a divisão são atribuídos aos "últimos *pipes*".

Nas duas abordagens, em vez de usarmos um limite fixo como tamanho do HH, implementamos uma técnica de um limite adaptado dinamicamente. Essa adaptação é fundamentada na média móvel ponderada exponencialmente *(Exponentially Weighted Moving Average - EWMA)* das contagens locais e globais. Sempre que o limite global é atingido, o limite do *switch* é ajustado. Neste ponto, calcula-se o EWMA e determina-se o novo limite local para o *switch* de borda que enviou a última atualização da tabela. O EWMA se baseia em valores antigos e atuais. Isso é feito utilizando um parâmetro de suavização (α), que controla o peso atribuído aos dados mais recentes. Esse valor deve estar sempre entre zero e um, sendo que, valores mais próximos de zero atribuem mais peso aos dados históricos, enquanto valores próximos de um conferem mais importância a dados recentes. Em nossa equação de 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 ($newL$), o limite global (gL), o limite anterior (oL), o valor do acumulador e os resíduos dos *pipes* de um *switch* ($sReport$). A variável "$sReport$", apresenta variação com relação aos índices "i" e "j", em que o "i" refere-se ao *switch* que atingiu seu limite local e enviou um *report* ao coordenador, e o "j" engloba os demais *switches* que não atingiram seus limites locais. Os valores obtidos das Equações 1 e 2 resultam em valores reais, pois *switches* programáveis não suportam operações com ponto flutuante.

Empregamos a EWMA para determinar um limite mais eficiente para a detecção de HH. Por exemplo, um limite fixo poderia resultar em um valor muito baixo. Assim, grande parte dos possíveis HH poderiam ser falsos positivos, o que provavelmente satisfaria uma parte significativa dos fluxos que passam pela rede. Por meio dessa técnica de limite adaptativo, podemos calibrar dinamicamente os limites locais, ajustando-os conforme necessário, permitindo encontrar valores ideais com base em cada *switch* de borda

e em cada fluxo. Isso possibilita um valor de limite mais próximo de um verdadeiro HH na rede. Observaremos essa comparação nos resultados na Seção 4.

Os detalhes das duas propostas para detecção de HH em *switches* de borda *multi-pipes* estão detalhadas nas Seções 3.1 e 3.2.

3.1. *Local-pipe*

Na solução *Local-pipe*, a contagem de pacotes é realizada individualmente por cada *pipe*. A Figura 2 representa um exemplo do funcionamento do *Local-pipe*.

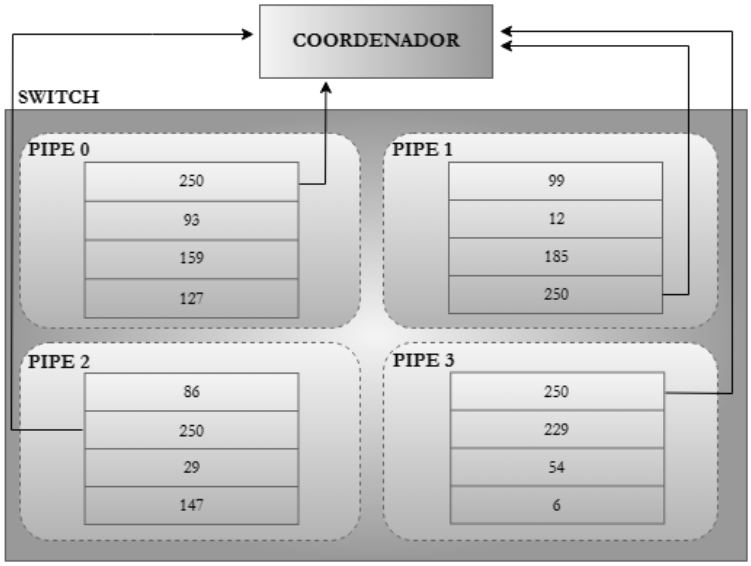

Figura 2. Execução do *Local-pipe* em um *switch* de borda.

Na Figura 2, 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 o 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.

Se 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 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.

3.2. *Accumulator*

Na solução do *Accumulator*, a contabilização dos pacotes também é realizada individualmente para cada *pipe*. No entanto, o diferencial reside na presença 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: conta os pacotes individuais transmitidos por ele e acumula as contagens de pacotes de todos os *pipes* do *switch*.

Na Figura 3 temos o funcionamento do *Accumulator*, 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*. Para que um pacote chegue no acumulador, é implementada uma recirculação interna no *switch*.

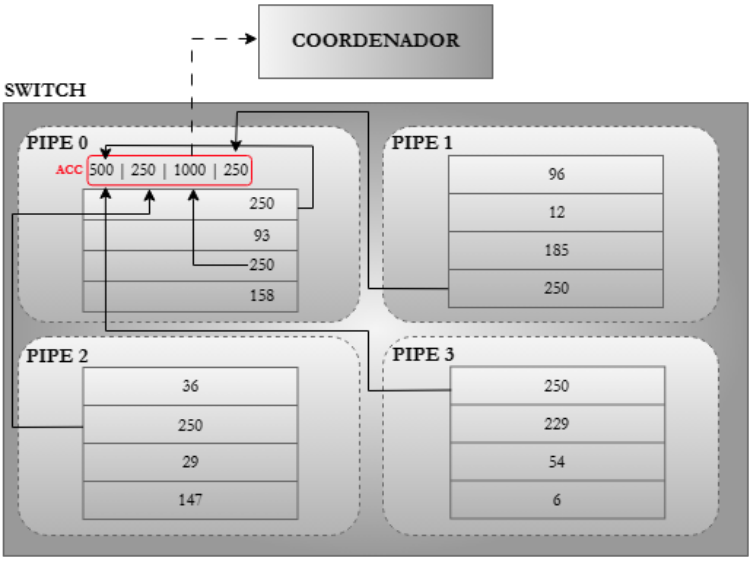

Figura 3. Execução do *Accumulator* em um *switch* de borda.

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. Esse cenário é ilustrado pelas linhas contínuas e tracejadas, sendo que a linha sólida representa a recirculação interna no *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* incorporar 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 do que o *Local-pipe*.

Na Seção 4, 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*.

4. Resultados

Para realizar nossos experimentos, desenvolvemos um emulador em Python, que suporta um número configurável de *switches* e *pipes* por *switch*. Nosso emulador foi implementado utilizando estruturas de dados que são encontrados no *switch*. Para nossos experimentos, definimos um cenário com três a seis *switches* de rede, todos multi-pipe, suportando de dois até 16 *pipes*. Sendo que utilizamos *switches* de borda para detecção dos HH. Para emular uma distribuição de tráfego do mundo real, utilizamos um trace CAIDA *Equinix-NYC* contendo 28 milhões de pacotes de rastreamento [CAIDA 2019].

Para a criação do *hash* neste trabalho, foi utilizada o método MD5 com 16 bits, armazenando a contagem dos HH em *arrays*. Além disso, definimos um fluxo como sendo os quatro campos do cabeçalho TCP/IP (IP de origem, IP de destino, porta de origem e porta de destino). Dessa forma, verificamos a existência de 1,8 milhão de fluxos distintos no trace utilizado. Além disso, estabelecemos o limite para HH como sendo os fluxos que possuem ou ultrapassam o valor de 10 mil pacotes. Importante ressaltar que esse limiar foi estabelecido de forma arbitrária. Assim, obtivemos 299 fluxos após a realização dessa filtragem no *trace* CAIDA. Para representar a variação do fluxo pelos *pipes* nosso emulador altera entre o fluxo do *pipe* sempre ultrapassado a metade do limite de pacotes do *switch* para um HH em um *pipe*, escolhendo aleatoriamente o próximo *pipe*.

Aplicamos ambas as abordagens propostas para a detecção de HH, e as comparamos entre si. Uma utiliza um limite fixo do *switch*, enquanto a outra emprega um limite adaptativo no *switch*. Além disso, os mesmos experimentos foram conduzidos para *switches* que possuem uma arquitetura baseada em um único *pipe* com limite fixo (*baseline*). Isso nos permitiu avaliar que, ao aplicar a detecção de HH em *switches multi-pipes*, conseguimos manter um F1-Score elevado em comparação com nossa *baseline*.

Tabela 2

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

Ao empregar um *switch multi-pipe*, conseguimos manter um elevado F1-Score, atingindo um valor de 0,99, tanto com o *Local-pipe* quanto com o *Accumulator*. Dessa forma, em todos experimentos, preservamos a mesma quantidade de HH reais detectados. Adicionalmente, com o uso do *Accumulator*, reduzimos o número de notificações para o plano de controle em comparação com nossa *baseline*. Os resultados detalhados desses experimentos serão apresentados em detalhes na Seção 4.1.

4.1. Sensibilidade aos *pipes*

O objetivo do nosso primeiro experimento foi determinar o melhor fator de suavização (valores maiores atribuem peso a dados recentes, enquanto valores menores favorecem dados históricos) para cada quantidade de *pipes*. No decorrer do experimento, exploramos o fator de suavização em uma faixa de 0,2 até 0,99, variando a quantidade de *pipes* de dois até 16, incluindo um *switch* de *pipe* único. Estes valores foram analisados com a quantidade de notificações ao plano de controle. Observamos que o fator de suavização para os *pipes* influencia nas notificações. Concluímos que a suavização está relacionada à quantidade de *pipes* ativos no *switch*, ou seja, cada quantidade de *pipes* possui uma suavização mais adequada para reduzir o número de notificações enviados ao controlador.

Na abordagem *Accumulator*, observamos que um *switch* com dois *pipes* apresenta um desempenho superior com um fator de suavização de 0,8 (69 notificações). Em contraste, ao lidar com um *switch* de 16 *pipes*, a configuração ideal seria 0,99 (116 notificações). O cenário se inverte quando empregamos oito *pipes*. Neste caso, a quantidade de notificações diminui quando a suavização atribui maior peso aos dados históricos, 0,4 (82 notificações).

Ao analisarmos a abordagem *Local-pipe*, percebemos que o *overhead* de comunicação é significativamente maior em comparação com o *Accumulator*, uma vez que o *Local-pipe* não incorpora um acumulador no *switch*. Considerando a suavização que gerou menos notificações para o *Local-pipe* (0,2 com dois *pipes*), e compararmos o mesmo cenário para o *Accumulator*, notamos um aumento de notificações para o *Local-pipe* em 3277% (*Accumulator* 103 notificações e *Local-pipe* 3481 notificações).

Constatamos que, em ambas abordagens cada quantidade de *pipes* apresenta sua suavização ideal, impossibilitando definir um valor universal para qualquer configuração de *pipes*. No entanto, notamos que, quanto maior a quantidade de *pipes*, maior a probabilidade de aumentar o *overhead* de comunicação. Esse fenômeno ocorre devido ao limite menor por *pipe*. Assim, os *pipes* atingem seus limites locais mais rapidamente, resultando em um maior número de notificações.

Nos demais experimentos, optamos por utilizar um valor padrão de 0,8 para o fator de suavização, independentemente do número de *pipes*. Esse valor foi escolhido por ser uma média adequada para todas as variações de *pipes*. Isso foi feito para evitar alterações constantes no fator, proporcionando uma base para comparações entre as quantidades de notificações ao coordenador.

4.2. Influência dos *pipes*

Neste experimento, avaliamos o impacto da quantidade de *pipes* empregando um *switch* com limite fixo, comparando-o com um *switch* com limite adaptativo. Utilizamos três *switches*, adotando uma suavização de 0,8 e limite de 10 mil. Observando o *Accumulator* na Figura 4, notamos que à medida que a quantidade de *pipes* aumenta, ocorre uma redução na quantidade de notificações para um *switch* com limite fixo. Isso ocorre devido à relação entre o aumento do número de *pipes* no *switch* e a distribuição dos pacotes entre eles. Como os fluxos podem variar de *pipes* durante a execução, algum *pipe* pode obter uma contagem de pacotes próxima de seu limite, e alterar o encaminhamento dos pacotes antes de atingirem seus limites. Deste modo, sem a necessidade de notificação ao coordenador.

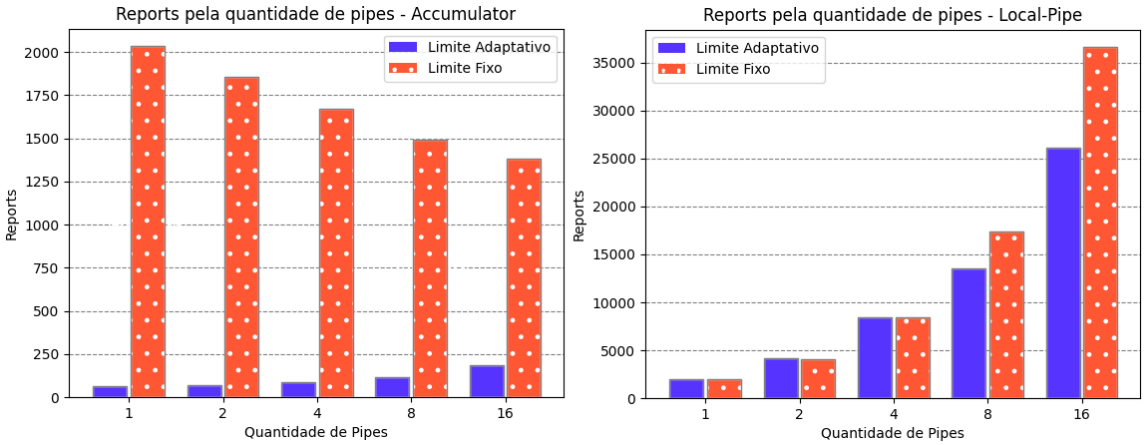

Figura 4. Relação entre a quantidade de notificações por *pipe*.

Em contrapartida, com um limite adaptativo no *switch*, nota-se um pequeno aumento nas notificações à medida que o número de *pipes* aumenta. Embora o aumento seja pequeno, a quantidade de notificações ainda é extremamente menor em comparação com os limites fixos, constatando a vantagem dos limites adaptativos, à medida que mais *pipes* são adicionados ao *switch*. Por exemplo, considerando quatro *pipes* no *Accumulator*, observamos uma redução de aproximadamente 95% nas notificações enviadas ao coordenador, comparando ambos os limites (limite adaptativo com 88 notificações e limite fixo com 1672 notificações). Da mesma forma, quando utilizamos o melhor cenário do *switches* com limite fixo, com 16 *pipes* (1383 notificações), ele supera o pior caso do *switch* com limites adaptativos (186 notificações) em 643%.

Na abordagem *Local-pipe*, conforme ilustrado na Figura 4, o cenário é invertido. Conforme esperado, o *Local-pipe* apresenta uma quantidade excessivamente maior de notificações ao controlador. À medida que aumentamos a quantidade de *pipes*, observamos um incremento nas notificações, independentemente do uso de limite fixo ou limite adaptativo.

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

4.3. *Switches* em toda rede

Neste experimento, investigamos se a quantidade de *switches* na rede tem algum impacto para detecção de fluxos HH. Todos os *switches* utilizados possuem dois *pipes*, a suavização é de 0,8 e o limite é 10 mil. A Figura 5 exibe um gráfico que ilustra a relação entre a quantidade de notificações pela quantidade de *switches* na rede. Analisando o *Accumulator* com limites adaptativos, notamos um aumento na quantidade de notificações enviadas ao coordenador à medida que o número de *switches* na rede aumenta, o que não ocorre ao utilizar *switches* com limites fixos.

No *Accumulator* da Figura 5, quando um limite fixo é aplicado ao *switch*, nota-se

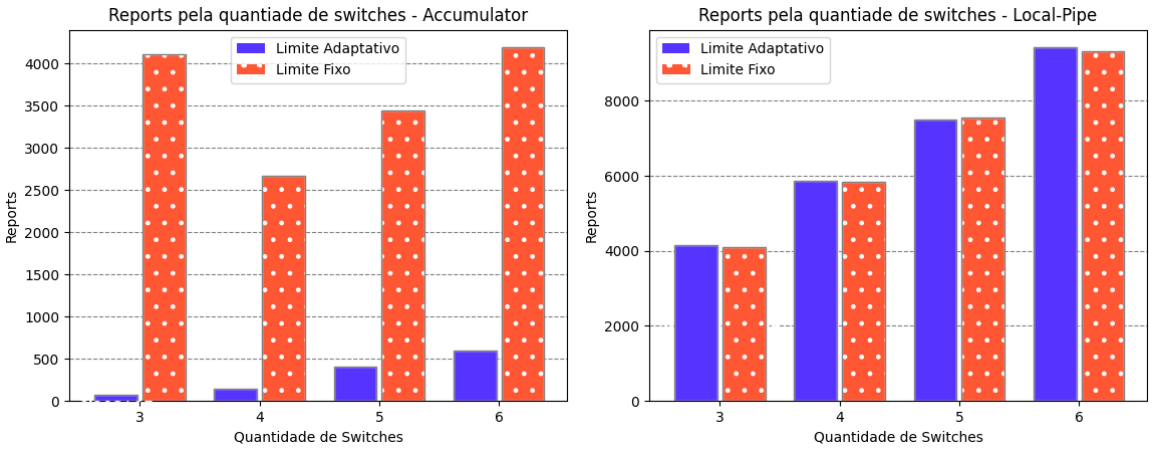

Figura 5. Aumento da quantidade de *switches* em relação as notificações.

uma diminuição na quantidade de notificações, visto em quatro e cinco *switches*. Essa redução pode ser atribuída à distribuição do limite do *switch*. O limite global, localizado no coordenador, é definido em 10 mil pacotes. Esse limite global é dividido pela quantidade de *switches* existentes na rede. Consequentemente, ao ter quatro e cinco *switches* na rede, os limites individuais seriam de 2500 e 2000, respectivamente.

Isso ocorre quando o limite global é divisível pela quantidade de *switches* na rede, resultando em menos notificações enviadas ao plano de controle. 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 notificações adicionais à medida que os pacotes são processados, levando a um aumento significativo na quantidade de notificações, em comparação com uma divisão que resultasse em um valor inteiro.

Apesar da redução nas notificações utilizando uma quantidade de *switches* divisível pelo limite global, a implementação de limites adaptativos nos *switches* resulta em uma redução significativa na quantidade de notificações em comparação com limite fixo. Ainda do *Accumulator*, comparando o melhor caso do limite adaptativo (69 notificações), observamos que o uso de limite fixo permanece alto (4110 notificações), alcançando um aumento de 5857% na quantidade de notificações. Mesmo considerando o melhor caso para limites fixos (2665 notificações), a discrepância em relação ao limite adaptativo (146 notificações) ainda é substancial, com um aumento de 1726% na quantidade de notificações geradas utilizando *switches* com limites fixos.

Além disso, ao analisarmos o cenário ideal de limites fixos em comparação ao mesmo cenário com limites adaptativos, constatamos que o limite fixo ainda mantém uma elevada quantidade de notificações, resultando em desempenho inferior em comparação com os limites adaptativos. Isso leva a um número consideravelmente maior de notificações, com um aumento de 598% (limite adaptativo com 600 notificações e limite fixo com 4189 notificações). Portanto, independentemente do número de *switches* no método de limite fixo, as soluções com limites adaptativos mantêm uma baixa quantidade de notificações para o coordenador.

Como esperado, o *Local-pipe* mantém sendo a abordagem menos eficiente, com um número excessivo de notificações. Tomando como exemplo a configuração com quatro *switches* com limites adaptativos, observa-se que a quantidade de notificações do *Local-pipe* é 39 vezes maior do que o mesmo cenário com a solução do *Accumulator* (*Local-pipe* com 5862 notificações e *Accumulator* com 146 notificações). Além disso, é importante destacar que o *Local-pipe* apresenta uma tendência de gerar um número crescente de notificações à medida que a quantidade de *switches* aumenta. Isso evidencia que o *overhead* de comunicação entre os planos aumentará à medida que adicionamos mais *switches*, independentemente de estarmos utilizar limites adaptativos ou fixos.

5. 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. Neste estudo, projetamos um algoritmo eficiente e implementamos um protótipo emulado capaz de detectar HH em toda rede com *switches multi-pipes*. Nossa análise, utilizando tráfego real, revela uma redução significativa no *overhead* de comunicação necessário para a detecção de HH em toda a rede, mitigando potenciais sobrecargas, latência e atrasos. Essa redução é alcançada por meio da utilização de limites adaptativos em conjunto com *switches multi-pipes*, mantendo um alto F1-Score comparando-o com nossa abordagem de *switch* de *pipe* único.

Observamos que a estratégia mais eficaz com *switches multi-pipes* é a inclusão de um acumulador em um dos *pipes*, ressaltando a importância dos limites adaptativos. Com o uso de um limite fixo, notamos um aumento médio de 92% no *overhead* de comunicação entre os planos de dados e de controle. A abordagem *Accumulator*, com limites adaptativos, manteve um F1-Score equivalente à abordagem de um único *pipe*, ao mesmo tempo em que reduziu o *overhead* de comunicação em 96,65%. 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, essa adaptação é crucial, independentemente da quantidade de *pipes* escolhidos, para minimizar a quantidade de notificações entre os planos.

Agradecimentos

O presente trabalho foi realizado com apoio da Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Código de Financiamento 001, FAPESP (processo 2021/14297-1) e CNPq (processo 305665/2022-7).

Referências

Agrawal, A. and Kim, C. (2020). Intel tofino2–a 12.9 tbps p4-programmable ethernet switch. In *2020 IEEE Hot Chips 32 Symposium (HCS)*, pages 1–32. IEEE Computer Society.

Basat, R. B., Chen, X., Einziger, G., and Rottenstreich, O. (2020). Designing heavy-hitter detection algorithms for programmable switches. *IEEE/ACM Transactions on Networking*, 28(3):1172–1185.

Basat, R. B., Einziger, G., Feibish, S. L., Moraney, J., and Raz, D. (2018a). Network-wide routing-oblivious heavy hitters. In *Proceedings of the 2018 Symposium on Architectures for Networking and Communications Systems*, pages 66–73.

Basat, R. B., Einziger, G., Keslassy, I., Orda, A., Vargaftik, S., and Waisbard, E. (2018b). Memento: Making sliding windows efficient for heavy hitters. In *Proceedings of the 14th International Conference on Emerging Networking EXperiments and Technologies*, pages 254–266.

CAIDA (2019). The caida ucsd anonymized internet traces - 20190117.

Chiesa, M. and Verdi, F. L. (2023). Network monitoring on multi-pipe switches. *Proc. ACM Meas. Anal. Comput. Syst.*, 7(1).

Ding, D., Savi, M., Antichi, G., and Siracusa, D. (2020). An incrementally-deployable p4-enabled architecture for network-wide heavy-hitter detection. *IEEE Transactions on Network and Service Management*, 17(1):75–88.

Harrison, R., Cai, Q., Gupta, A., and Rexford, J. (2018). Network-wide heavy hitter detection with commodity switches. In *Proceedings of the Symposium on SDN Research*, pages 1–7.

Hofstede, R., Čeleda, P., Trammell, B., Drago, I., Sadre, R., Sperotto, A., and Pras, A. (2014). Flow monitoring explained: From packet capture to data analysis with netflow and ipfix. *IEEE Communications Surveys & Tutorials*, 16(4):2037–2064.

Lin, Y.-B., Huang, C.-C., and Tsai, S.-C. (2019). Sdn soft computing application for detecting heavy hitters. *IEEE Transactions on Industrial Informatics*, 15(10):5690–5699.

Machado, D., de Castro, A. G., Vogt, F., and Luizelli, M. C. (2019). Avaliação de desempenho de heavy hitters utilizando p4 e xdp. In *Anais da XVII Escola Regional de Redes de Computadores*, pages 122–123. SBC.

Silva, M. V. B. d. (2019). Prevendo e identificando fluxos elefantes em redes de ponto de troca de tráfego com suporte à programabilidade.

Sivaraman, V., Narayana, S., Rottenstreich, O., Muthukrishnan, S., and Rexford, J. (2017). Heavy-hitter detection entirely in the data plane. In *Proceedings of the Symposium on SDN Research*, pages 164–176.

Tang, L., Huang, Q., and Lee, P. P. (2020). A fast and compact invertible sketch for network-wide heavy flow detection. *IEEE/ACM Transactions on Networking*, 28(5):2350–2363.

Turkovic, B., Oostenbrink, J., and Kuipers, F. (2019). Detecting heavy hitters in the data-plane. *arXiv preprint arXiv:1902.06993*.

Vilela, G. S. (2006). 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*.

Wheeler, B. (2019). Tomahawk 4 switch first to 25.6 tbps. *Microprocessor Report*.