Aplicações de monitoramento de tráfego utilizando redes programáveis eBPF

Elerson R. S. Santos[1], **Eduardo P. M. Câmara Júnior**[1]
Marcos A. M. Vieira[1], **Luiz F. M. Vieira**[1]

[1]Departamento de Ciência da Computação – Universidade Federal de Minas Gerais
Av. Antônio Carlos, 6627 – Pampulha – Belo Horizonte – 31270-010 – Brazil

Abstract. *Traffic monitoring is an important tool for traffic engineering, allowing network managers to detect attacks and abnormal network behaviors. The SDN paradigm provides a new way of doing this, as it separates the data and control planes and allows them to be programmed. Here we propose the implementation of traffic monitoring applications in BPFabric, an SDN architecture that uses the packet processing virtual machine eBPF to provide flexibility and programmability in switches. We use sample-based and sketch-based applications to demonstrate the capabilities of BPFabric for traffic monitoring.*

Resumo. *Monitoramento de tráfego é uma ferramenta importante em engenharia de tráfego, permitindo que gerentes de rede possam detectar ataques e comportamentos anormais da rede. O paradigma de SDN provê uma nova forma para fazer isso, separando os planos de dados e controle e permitindo que eles sejam programáveis. Neste trabalho propõem-se a implementação de aplicações para monitoramento de redes utilizando o BPFabric, uma arquitetura SDN flexível e programável que utiliza uma máquina virtual eBPF para o processamento de pacotes. As capacidades do BPFabric para tarefas de monitoramento de tráfego são apresentadas através de aplicações baseadas em amostragens e sketches.*

1. Introdução

O monitoramento de tráfego em redes pode ter um impacto importante na qualidade de serviços prestados por provedores de Internet. Exemplos de aplicações que podem utilizar informações obtidas por monitoramento de tráfego incluem engenharia de tráfego, detecção de eventos anômalos na rede, qualidade de serviço e qualidade de experiencia.

Além disso, com os avanços mais recentes em redes de computadores e em hardware, hoje é possível realizar a programação de switches. Esse paradigma é conhecido como redes definidas por software (SDN, do inglês *Software-Defined Network*) e permite a separação dos planos de controle e de dados. O plano de controle decide como os fluxos da rede devem ser tratados, enquanto o plano de dados encaminha os fluxos de acordo com as regras que foram definidos pelo plano de controle [Kreutz et al. 2015].

SDN tem permitido a criação de aplicações inovadoras, propiciando, por exemplo, testes de algoritmos em redes já em produção. Apesar disso, o monitoramento de tráfego têm ganhado pouca atenção do ponto de vista das arquiteturas SDN. Por exemplo, o OpenFlow [McKeown et al. 2008], a implementação *de facto* da arquitetura SDN, provê uma tabela de roteamento programável a partir de um controlador. Entretanto, sozinha esta arquitetura não é suficiente para as aplicações de monitoramento.

Neste trabalho são propostos mecanismos para monitoramento utilizando o BPFabric [Jouet and Pezaros 2017]. BPFabric é uma arquitetura programável para SDN que utiliza o eBPF (*extended Berkley Packet Filter*) para o processamento de pacotes. O eBPF, por sua vez, é uma máquina virtual utilizada principalmente para a filtragem de pacotes e que faz parte do *kernel* do Linux desde a versão 3.18 [Pacífico et al. 2018]. O BPFabric permite a utilização de uma linguagem de programação de alto nível, baseada em C, que permite que pacotes sejam analisados e encaminhados para a rede ou para o controlador. A arquitetura não contém um mecanismo dedicado para monitoramento de tráfego, mas permite a implementação e a extensão de suas funcionalidades através da adição de novas estruturas de dados e algoritmos. Este trabalho propõe a implementação de mecanismos que permitam que o BPFabric seja utilizado para aplicações de monitoramento de trafego.

O crescente aumento de tráfego têm tornado a tarefa de análise de fluxos cada vez mais desafiadora. Devido a isto, as aplicações de monitoramento de tráfego têm se tornado cada vez mais relevantes, principalmente aquelas de tempo real. Para resolver o problema, atualmente existem três soluções principais que podem ser consideradas: a utilização de *heaps*, de técnicas de amostragem e de sketches. As soluções baseadas em *heap* mantém um pequeno sumário do tráfego de redes em memória [Alipourfard et al. 2015]. Soluções baseadas em amostragem utilizam algum mecanismo para selecionar apenas um pequeno conjunto dos dados para serem analisados [Duffield et al. 2003, Estan and Varghese 2002]. Já as soluções baseadas em sketches utilizam estruturas de dados probabilísticas, trocando precisão por tempo de processamento e memória.

Este trabalho utiliza sketches para a implementação das seguintes aplicações de monitoramento: detecção de mudanças bruscas, detecção de *Heavy Hitters*, estimativa da distribuição do tamanho dos fluxos e contagem de tráfego. Essa escolha é feita considerando que sketches são capazes de lidar com tarefas associadas ao monitoramento de tráfego com consumo de memória e precisão parametrizados. Desta forma, foram implementadas, utilizando BPFabric, e analisadas as seguintes estruturas probabilísticas: Bloom Filter, Count-Min Sketch, k-ary Sketch e PCSA. Soluções baseadas em amostragem são também utilizadas para contabilizar a quantidade de fluxos ativos e o número de pacotes por protocolo IP (TCP, UDP, ICMP, dentre outros) na rede ao longo do tempo.

As contribuições deste trabalho são: (i) desenvolvimento de estruturas probabilísticas para a arquitetura BPFabric com o objetivo de auxiliar tarefas de monitoramento de tráfego; (ii) implementação de aplicações de monitoramento baseadas em sketches e amostragem na mesma arquitetura; e (iii) avaliação das aplicações implementadas que mostra, através de simulações com dados reais, os pontos fortes de cada uma delas e os problemas de desempenho encontrados.

Este trabalho é dividido como descrito a seguir. Primeiro são apresentados alguns trabalhos relacionados, mostrando aqueles que são mais relevantes e comparáveis a este. Em seguida, descreve-se as estruturas de dados probabilísticas utilizadas em algumas das aplicações de monitoramento implementadas. As aplicações de monitoramento são então descritas em conjunto com resultados de simulações que fazem uso de dados reais para mostrar o desempenho delas em cenários reais. Finalmente, tem-se as conclusões obtidas e as intenções para trabalhos futuros.

2. Trabalhos Relacionados

Tsai *et al.* [Tsai et al. 2018] destacam que o monitoramento em redes têm sido um tópico ativo de pesquisa nas últimas décadas. Muitos trabalhos nessa área têm se preocupado em desenvolver novos mecanismos para monitoramento, que incluem novas estruturas de dados e novas arquiteturas. Alguns desses trabalhos são apresentados a seguir.

No trabalho apresentado em [Gupta et al. 2017], os autores propõem um sistema de análise e coleta de dados chamado Sonata. O Sonata permite que um operador expresse consultas utilizando uma linguagem em alto nível. Este trabalho pode ser considerado complementar com o apresentado aqui, dado que o *back-end* das consultas poderiam ser implementados utilizando o BPFabric enquanto as consultas poderiam utilizar o Sonata.

Já em [Tsai et al. 2018] são apresentados os trabalhos mais recentes que consideram arquiteturas SDN e monitoramento em redes. No trabalho, os autores discutem a possibilidade de criar funções primitivas que possam ser agregadas para a criação de aplicações de monitoramento. A maior dificuldade para a implementação desse tipo de sistema em arquiteturas tradicionais de SDN é que elas geralmente não são desenvolvidas pensando em monitoramento de dados. Isso mostra as capacidades do BPFabric, sendo ela uma arquitetura genérica o suficiente para permitir a introdução de primitivas de monitoramento, tais como estruturas de dados probabilísticas.

Enquanto isso, em [Alipourfard et al. 2015] é analisado as diferentes alternativas existentes para monitoramento baseado em: amostragem, funções de hash, *heaps* e estruturas probabilísticas. O objetivo principal do trabalho foi avaliar a tendência das técnicas de monitoramento, tendo como objetivo a eficiência do ponto de vista de memória. Outro ponto interessante levantado é que a técnica utilizada no *pipeline* do processamento dos pacotes pode impactar o desempenho do sistema, sendo possível que ela seja seu gargalo. Um dos objetivo do presente trabalho é apontar as vantagens da utilização de SDN no monitoramento de pacotes, sendo escolhido a utilização de estruturas de dados probabilísticos para algumas aplicações por suas inúmeras vantagens apresentadas na literatura.

Considerando o padrão OpenFlow, o OpenNetMon [van Adrichem et al. 2014] é um controlador modular. Essa arquitetura utiliza as estatísticas presentes no OpenFlow para sumarizar as estatísticas da rede. Essa é uma solução limitada considerando que as aplicações de monitoramento podem utilizar apenas as informações providas pelo padrão OpenFlow. Análises mais complexas, como por exemplo, aquelas providas por aplicações que avaliam destinos e origens de pacotes IP, não podem ser feitas nesse modelo.

SketchVisor [Huang et al. 2017] é uma ferramenta robusta para medição de estatísticas de redes. O trabalho aborda o problema da quantidade de processamento requerido por soluções baseadas em sketch e propõe, como solução, a adição de um caminho de dados rápido na arquitetura com o objetivo de permitir que medições ocorram o mais rápido possível. Este trabalho não foca na eficiência das estrutura de dados implementadas, visto que o BPFabric pode ser otimizado de várias formas, inclusive sendo possível a adição de um caminho de dados que favoreça tarefas de monitoramento.

Já o OpenSketch [Yu et al. 2013] é uma arquitetura SDN baseada em sketches para o monitoramento de tráfego de redes. Ele provê uma biblioteca de monitoramento que disponibiliza um conjunto de estruturas de dados probabilísticas já implementadas, que permite que gerentes de redes foquem na construção de aplicações de monitoramento.

Este trabalho também provê um conjunto de sketches, mas eles são implementados utilizando o BPFabric para permitir a construção de aplicações de monitoramento de maneira similar à construção das aplicações tradicionais de SDN.

Considerando o BPFabric [Jouet and Pezaros 2017], a arquitetura foi desenvolvida com objetivo de ser independente de linguagens de programação, protocolos e hardware. A ideia principal do BPFabric é utilizar a máquina virtual eBPF para permitir a programação das funcionalidades de processamento e encaminhamento dos pacotes, podendo essas serem ainda permutadas *online*. O eBPF é uma versão aprimorada do BPF (*Berkeley Packet Filter*) que possui uma arquitetura de 64 bits e 11 registradores, além de ser programável através da linguagem de programação C e atualmente ser integrado ao *kernel* do Linux[1]. Neste trabalho o objetivo será adicionar a capacidade de monitoramento à arquitetura do BPFabric.

Por último, P4 [Bosshart et al. 2014] é uma linguagem que tem propósitos similares àqueles da arquitetura do BPFabric. A linguagem P4 é utilizada em [Sivaraman et al. 2017] para identificar fluxos que trafegam com grandes volumes de dados pela rede. Apesar de propósitos similares aos do BPFabric, o P4 possui algumas limitações que acabam reduzindo sua aplicabilidade. Como exemplo, cita-se a não aceitação de *loops* nos programas. Assim, o BPFabric se torna mais interessante do ponto de vista de monitoramento de redes.

3. Estruturas de Dados Probabilísticas

Esta seção apresenta as estruturas de dados probabilísticas implementadas no BPFabric[2]. Elas foram utilizadas na implementação de algumas das aplicações de monitoramento de dados apresentadas na próxima seção.

3.1. Bloom Filter

O Bloom Filter é uma estrutura probabilística eficiente em termos de espaço que é utilizada para verificar se um elemento pertence ou não a um determinado conjunto [Bloom 1970]. Essa estrutura de dados é formado por um *array* B composto por m bits, todos com valor inicial igual a 0. Ela também utiliza k funções de hash $h_1, h_2, ..., h_k$ para mapear um elemento para k posições do *array* B. Uma implementação de um Bloom Filter com uma dada uma tolerância a erros, expressa com uma probabilidade p, e com um número esperado de elementos no conjunto n, requer a utilização de $k = -\log_2 p$ funções de hash e um *array* de tamanho $m = -(n \ln p)/(\ln 2)^2$ bits.

A inserção de um novo elemento e à esta estrutura de dados requer a ativação dos k bits (fazer os bits iguais a 1) de B relacionados a e. Para isto, obtém-se uma chave para cada uma das k funções de hash definidas e elas definirão os bits serão ativados no *array* $(h_1(e), h_2(e), ..., h_k(e))$. Dessa forma, para inserir um elemento e no conjunto, utiliza-se a seguinte operação: $B[h_i(e)] = 1$ para $i = 1, 2, ..., k$. Para determinar se um determinado elemento e_x pertence ao conjunto, é necessário apenas verificar se todos os bits $B[h_i(e_x)]$,para $i = 1, 2, ..., k$, estão ativados. Caso um bit não esteja ativado, este elemento não está no conjunto.

[1] A partir da versão 3.18

[2] Disponíveis em `https://github.com/elerson/BPFabric`

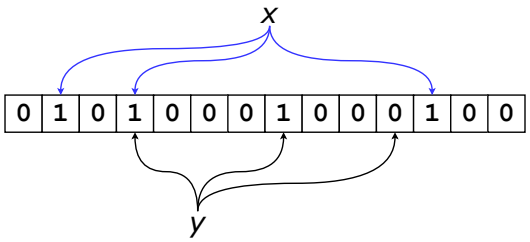


Figura 1. Bloom Filter: Elementos *x* e *y* são mapeados para, por exemplo, 3 posições cada, que indicam quais bits devem ser ativados.

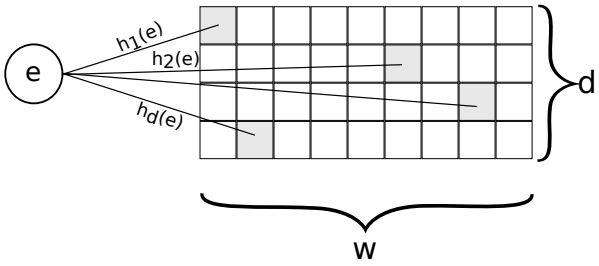


Figura 2. Count-Min Sketch: Entrada *e* é mapeada para *d* contadores utilizando *d* funções de hash par-a-par independentes.

É interessante notar que o Bloom Filter não apresenta falsos negativos, ou seja, ele não acusa a existência de um elemento no conjunto que realmente não faça parte dele. Entretanto, falsos positivos são possíveis. Um elemento que não está no conjunto pode ser acusado como existente caso os bits associados a ele tenham sido ativados durante inserções de outros elementos no conjunto. É possível, entretanto, determinar e reduzir a taxa de erros esperada nessa estrutura de dados, sendo ela igual a $(1 - e^{-kn/m})^k$ para uma quantidade n de elementos contidos no conjunto. A figura 1 apresenta a estrutura de dados do Bloom Filter.

3.2. Count-Min Sketch

O Count-Min Sketch [Cormode and Muthukrishnan 2005] é uma estrutura de dados compacta utilizada para contagem de frequência. Esta estrutura de dados é composta por uma matriz de ordem $w \times d$ de contadores inteiros que são inicializados com zero. Ela também considera um conjunto de funções de hash par-a-par independentes $h_1, h_2, ..., h_d$ utilizado para mapear um elemento de entrada à um contador em cada uma das linhas da matriz. A figura 2 contém uma representação dessa estrutura de dados.

Existem dois procedimentos fundamentais no Count-Min Sketch: a atualização e a estimativa. O procedimento de atualização é utilizado para registrar o acontecimento de um evento e. Para isso, as saídas das d funções de hash são calculadas para a entrada e e depois elas são utilizadas no processo de incrementar os contadores nas posições $(1, h_1(e))$, $(2, h_2(e))$, ..., $(d, h_d(e))$.

O procedimento de estimativa é utilizado para estimar a frequência de ocorrência da entrada e. Ela também requer o cálculo das d funções de hash para a entrada para que os contadores associados à ela possam ser localizados. A estimativa da frequência de e é então obtida tomando-se o valor mínimo dentre aqueles dos d contadores.

A principal vantagem dessa estrutura de dados é a necessidade de uma quantidade de espaço menor para computar a frequência de eventos. Essa vantagem, porém, vêm ao custo de precisão, visto que a estrutura pode superestimar a frequência de algum evento. Entretanto, como a ocorrência de qualquer evento é sempre registrada (através do procedimento de atualização), o Count-Min Sketch nunca subestima alguma frequência. Essa estrutura possui uma estimativa de erros igual a $2n/w$ com a probabilidade de pelo menos $1 - 0.5^d$, one n é o valor estimado de elementos a serem contados.

3.3. K-ary Sketch

O k-ary Sketch é uma estrutura de dados probabilística projetada para contar a atividade total de uma dada chave durante certos períodos de tempo [Krishnamurthy et al. 2003]. Similar ao Count-Min Sketch, ela consiste de uma matriz H de ordem $w \times d$, onde a j-ésima linha é associada a uma função de hash h_j. As funções de hash utilizadas devem ser 4-universais [Carter and Wegman 1979] para que a precisão das estimativas tenham garantias probabilísticas. Essa estrutura (figura 3) oferece operações para atualizar o sketch, estimar um valor através de uma chave e computar a combinação linear de múltiplos sketches.

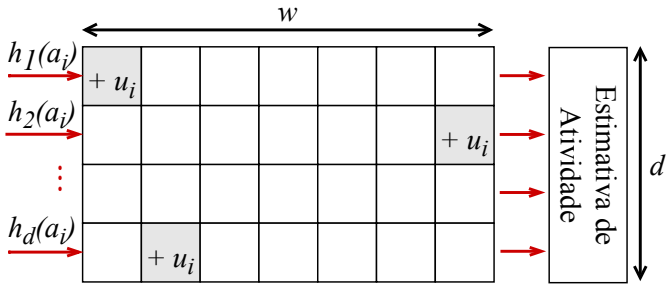


Figura 3. K-ary Sketch: Computa-se uma função de hash para cada uma das *d* linhas e os contadores associados são incrementados.

Considerando um fluxo de dados $S = \alpha_1, \alpha_2, ...$, onde $\alpha_i = (a_i, u_i)$ é um par de chave (a_i) e valor de atualização (u_i), a operação de atualização de um item é dada por:

$$\forall j \in \{1, ..., d\}, H[j][h_j(a_i)] = H[j][h_j(a_i)] + u_i \tag{1}$$

Já a operação de estimativa para uma chave a_i computa a seguinte equação:

$$mediana_{j \in \{1,...,d\}} \left\{ \frac{H[j][h_j(a_i)] - soma(S)/w}{1 - 1/w} \right\}, \tag{2}$$

onde $soma(S) = \sum_{k \in \{1,...,w\}} H[0][k]$ só precisa ser computado uma vez antes de qualquer estimativa de valores.

A operação de combinação linear une n sketches fazendo $S = \sum_{l=1}^{n} c_l S_l$, onde c_l é o peso associado ao sketch S_l. Para isso, ela combina cada entrada da matriz H fazendo:

$$H[j][k] = \sum_{l=1}^{n} c_l \times H_l[j][k] \tag{3}$$

Considerando esta forma de combiná-los, a estimativa provida pelo k-ary Sketch é estatisticamente sem viés. Além disso, a linearidade desse sketch possibilita que modelos de previsão sejam implementados sobre ele.

3.4. PCSA

A Contagem Probabilística com Média Estocástica (PCSA, do inglês *Probabilistic Counting with Stochastic Averaging*) foi proposta por Flajolet and Martin [Flajolet and Martin 1985] e seu principal objetivo é estimar o número de elementos

distintos em uma grande coleção. Para se ter uma ideia, o PCSA é capaz de contar bilhões de elementos utilizando apenas um número comparavelmente pequeno de bits.

A inserção de um novo elemento ao PCSA utiliza uma função de hash e outras duas funções $r(x)$ and $R(x)$. Aqui considera-se que as funções de hash retornam inteiros (de 32 ou 64 bits). A função $r(x)$ conta o número de 1's à direita no número de entrada x, enquanto $R(x) = 2^{r(x)}$. A função $R(x)$ é computada utilizando as instruções $\neg x \,\&\, (x{+}1)$.

O PCSA armazena uma matriz de ordem $w \times d$, onde w pode ser igual a 32 ou 64 bits e d, o número de linhas na matriz, é utilizado para controlar a precisão dessa estrutura. Quando um novo elemento é adicionado, a função de hash é computada e retorna um número k que indica qual a linha da matriz deverá receber o novo dado. Dada a linha k selecionada, executa-se a operação $matriz[k] = matriz[k] \mid R(x)$.

A contagem do número de elementos no PCSA requer que a soma $s = \sum_{i=1}^{d} r(matriz[i])$ seja computada. Esse valor é utilizado para obter o resultado da contagem, que é dado por $d * 2^{(s/d)/0.77351}$. A precisão do PCSA, dada uma matriz de $d \times 64$ bits, é igual a $0.78\sqrt{d}$. Esta operação é apresentada no algoritmo 1.

Algorithm 1 Operação de contagem do PCSA

```
1: procedure CONTAELEMENTOS(Matriz)
2:     soma = 0.0
3:     for i = 0; i < tamanho(Matriz) do
4:         soma+ = r(Matriz[i])
5:     end for
6:     media = soma/tamanho(Matriz)
7:     return tamanho(Matriz) * 2^(media/0.77351)
8: end procedure
```

4. Aplicações de Monitoramento de Tráfego

Nesta seção são apresentadas as aplicações de monitoramento de tráfego de rede implementadas utilizando o BPFabric. Elas podem ser encontradas na literatura e possuem o objetivo de prover informações sobre o tráfego de redes. Essas ferramentas permitem a detecção de anomalias na rede, estimativa de recursos necessários, detecção pontos de vulnerabilidade e obtenção de estatísticas de padrão de tráfego.

A avaliação das implementações foi feita como descrito a seguir. Utilizou-se o software Mininet [Lantz et al. 2010] para criar uma rede composta por um *host* conectado a um switch em software (o mesmo disponibilizado na implementação do BPFabric [Jouet 2017]) com duas portas. O *host* foi responsável por gerar todo o tráfego da rede, enviando diretamente ao switch pacotes provenientes do *dataset* do CAIDA de 2013 [CAIDA 2013]. Esse *dataset* contém *traces* coletados em *backbones* comerciais de alta velocidade e todos os 9319569 pacotes contidos nele, e utilizados nos testes, não possuem *payload*. O *dataset* foi pré-processado para determinar quais eram os valores esperados e eles foram utilizados nas comparações com os resultados obtidos.

4.1. Detecção de Mudanças Bruscas

Para essa aplicação, considera-se o modelo de fluxo apresentado em [Krishnamurthy et al. 2003]. Seja $S = \alpha_1, \alpha_2, ...$ um fluxo de dados, onde

$\alpha_i = (a_i, u_i)$ é um par formado por uma chave a_i e um valor de atualização $u_i \in \mathbb{R}$. Cada chave é associada a um sinal $U[a_i](t)$ no tempo t, que é incrementado em u_i a cada unidade de tempo. Assim, tem-se que $U[a_i](t_{j+1}) = U[a_i](t_j) + u_i, \forall j \in \mathbb{N}$. O objetivo desta aplicação é encontrar sinais com mudanças consideráveis entre intervalos de tempos consecutivos. Isto é chamado de detecção de mudanças bruscas.

A estimativa de mudanças bruscas pode ser feita utilizando-se um modelo de previsão, tal como a média móvel (MA), a média móvel exponencial (EWMA) ou ainda modelos mais complexos, como o modelo integrado auto-regressivo (ARIMA) [Krishnamurthy et al. 2003] por exemplo. Utilizando algum destes modelos, uma mudança brusca é detectada quando a diferença entre o valor previsto e aquele observado é maior do que um valor predefinido.

Outra alternativa, como proposto por [Schweller et al. 2007], é utilizar um modelo simplificado que quebra uma sequência de dados em dois ou mais pedaços temporalmente adjacentes. Nesse caso, a mudança brusca é detectada quando a diferença entre a quantidade de dados em intervalos adjacentes é maior do que um valor previamente definido. Por simplicidade, esse é o modelo utilizado nesta aplicação.

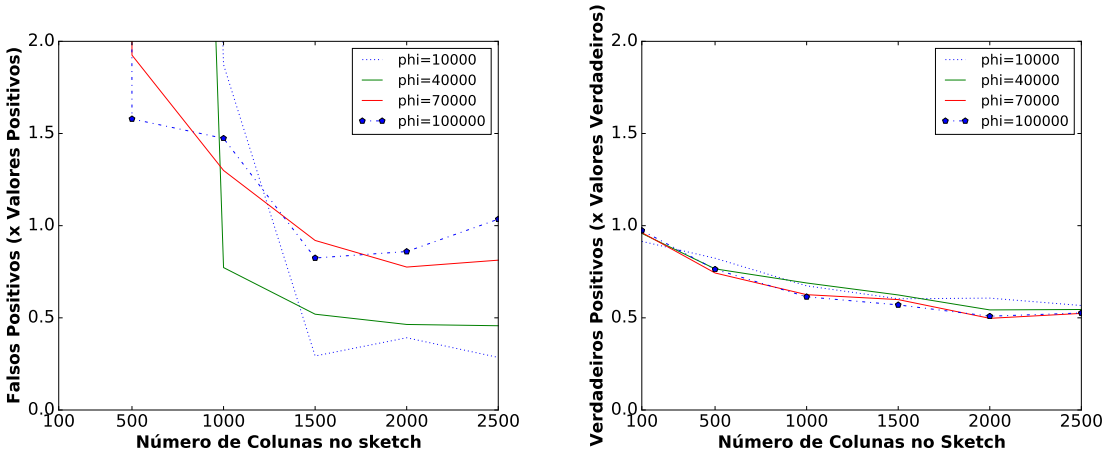
A aplicação implementada utiliza um k-ary Sketch para detectar mudanças bruscas online. Ela divide a sequência de dados em pedaços temporalmente adjacentes e preenche o sketch no tempo t, enquanto busca por mudanças bruscas verificando a estrutura preenchida no tempo $t-1$. Uma mudança brusca só é detectada quando a chave associada a ela é requisitada novamente no momento seguinte. Isto é aceitável para aplicações realistas, como por exemplo aquelas utilizadas para detectar ataques de negação de serviço (DoS), onde o objetivo pode ser evitar que o atacante consiga atingir o nó destino no futuro. Soluções mais avançadas, que utilizam conceitos como o de sketches reversíveis, onde conhecer as chaves não seria necessário para detectar mudanças bruscas são descritas em [Schweller et al. 2007].

A figura 4a mostra o número de falsos positivos para a implementação realizada desta aplicação. Para isto, calculou-se a diferença entre o número de chaves retornadas pela aplicação e aquele que era esperado, divididos pelo número total de chaves. É possível observar que o número de falsos positivos decresce com o aumento do número de colunas do k-ary sketch. Entretanto, como mostra a figura 4b, o número de positivos verdadeiros também decresce quando o número de colunas aumenta. Acredita-se que tal diminuição seja resultado de problemas de desempenho, visto que aumentar o número de colunas causa um aumento no número de operações.

4.2. Detecção de Heavy Hitters

Essa aplicação visa identificar fluxos que carregam grandes volumes de dados, os chamados *heavy hitters*. Tal identificação é útil para detectar os nós que estejam consumindo a maioria dos recursos de uma rede. Ela pode ser utilizada para, por exemplo, detectar ataques DDoS [Harrison et al. 2018].

A aplicação é pensada com o objetivo de identificar endereços internos da rede que utilizam mais do que um limite predefinido de bytes da rede em um determinado período de tempo. Ela utiliza um Count-Min Sketch para contar o número de bytes transitados por fluxo e assim ser capaz de detectar os *heavy hitters*. Ela também utiliza um Bloom Filter para determinar se o endereços são internos ou externos à rede. Para isto, primeiro



(a) Número de falsos positivos **(b) Número de verdadeiros positivos**

Figura 4. Resultados observados na detecção de mudanças bruscas. O parâmetro *phi* é o limiar (em bytes) utilizado para detectar estas mudanças.

é necessário inserir os endereços de todos os nós da rede no Bloom Filter. Quando a fonte de um *heavy hitter* for identificada como externa à rede (endereço não presente no Bloom Filter), o controlador pode ser notificado para que alguma ação possa ser tomada.

A figura 5a mostra o número de falsos positivos em relação aos verdadeiros positivos para a implementação desta aplicação. Foram variados o número de linhas do Count-Min Sketch e o valor de phi, que aqui representa o limiar em bytes, para que uma fonte seja classificada como um *heavy hitter*. Os resultados mostram que valores maiores de phi resultaram em números maiores de falsos positivos observados. Já o número de linhas do sketch aparentou ter pouca influência sobre a quantidade de falsos positivos.

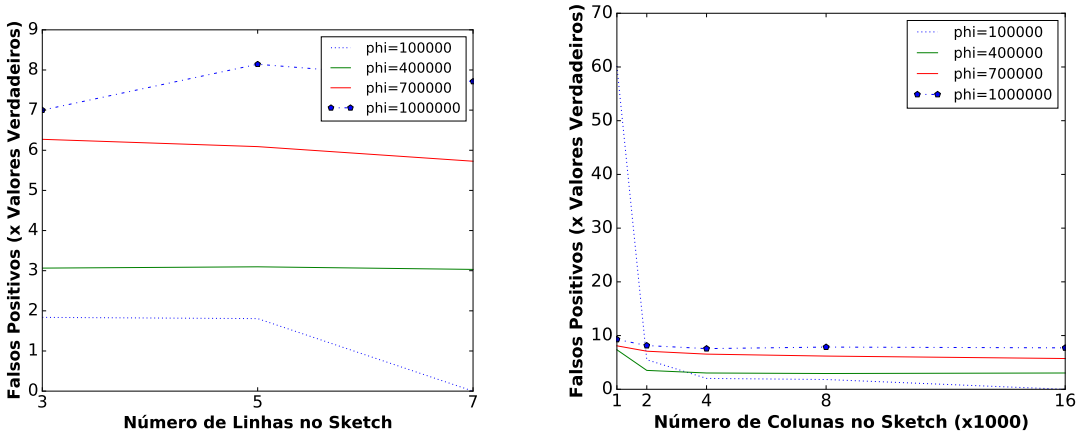
A figura 5b mostra a mesma relação da figura 5a, mas agora variando-se o número de colunas do Count-Min Sketch. Observa-se que o uso de valores baixos de phi resultou em menores quantidades de falsos positivos. Além disto, é possível observar que, assim como o número de linhas, o número de colunas teve pouca influência no número de falsos positivos. É importante notar que, como foram utilizados o Count-Min Sketch e o Bloom Filter, falsos negativos não eram esperados.

4.3. Estimativa da Distribuição do Tamanho dos Fluxos

Esta aplicação possui o objetivo de estimar a distribuição do tamanho dos fluxos de uma rede. Ou seja, deseja-se obter a distribuição dos tamanhos dos pacotes que trafegam em uma rede. Uma aplicação como essa pode ser utilizada para engenharia de infraestrutura e planejamento de recursos de uma rede [Kumar et al. 2004].

Para obter tal estimativa, a aplicação faz o uso de um *array* de contadores. A ideia é dividir o intervalo de possíveis tamanhos de pacotes em subintervalos de mesmo tamanho e então atribuir um contador para cada um deles. Dessa forma, quando um pacote passa pelo switch, seu tamanho é extraído e o contador correspondente ao intervalo em que ele se encaixa é incrementado. Essa aplicação também poderia incluir um Bloom Filter para obter a estimativa referente somente a um conjunto predefinido de fontes.

A figura 6 mostra os resultados observados para a implementação desta aplicação



(a) Número de linhas **(b) Número de colunas**

Figura 5. Resultados observados na detecção de *heavy hitters* variando-se as dimensões do Count-Min Sketch.

na forma de um histograma com barras que representam intervalos de 7 bytes. Através dela é possível notar que a maioria dos pacotes recebidos tinham tamanhos próximos de 100 bytes, enquanto pacotes com mais de 1500 bytes não eram esperados. Observa-se que a aplicação implementada utiliza os cabeçalhos dos pacotes para determinar o tamanho deles e logo não foi influenciada pela ausência de *payload* dos pacotes de teste.

4.4. Contagem de Tráfego

Aplicações de contagem de tráfego contabilizam o número de fluxos distintos em uma rede em um dado momento. Através disso, é possível, por exemplo, realizar a detecção de comportamentos anômalos em redes. Essas aplicações também podem ser úteis para tarefas de gerenciamento de rede onde existe interesse em qualidade de serviço (QoS).

A aplicação desenvolvida nesta parte utiliza o PCSA para realizar a contagem de tráfego. Ela tem por objetivo registrar, a cada segundo, a quantidade de IPs de fontes distintas dos pacotes que circularam na rede.

A figura 7 mostra a taxa de erros obtidas com a implementação desta aplicação. Cada uma das curvas desta figura corresponde a um número diferente de bits (x32) utilizados na aplicação. A divergência que pode ser vista nos resultados foi causada principalmente pelas características de funções de hash, que possuem diferentes valores para diferentes tamanhos da estrutura de dados. Além disto, é possível observar que a taxa de erros oscilou bastante para os diferentes cenários. Enquanto valores próximos de 10% foram alcançados quando 1024 bits foram utilizados, a utilização de 512 bits resultou em valores próximos de 40%. Tal variação pode ser explicada pelo fato de que o PCSA foi projetado para conjuntos grandes de objetos (em torno de milhões), enquanto apenas aproximadamente 20000 IPs únicos foram utilizados no teste.

4.5. Contagem de Fluxos Ativos

Essa aplicação possui o objetivo de rastrear o número de fluxos ativos ao longo do tempo em uma rede. Aqui, considera-se um fluxo como sendo qualquer troca de mensagens

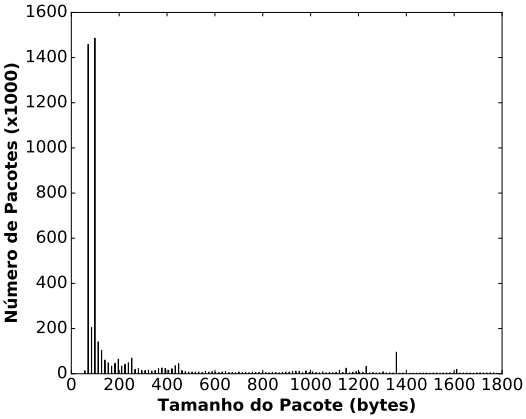


Figura 6. Distribuição do tamanho dos pacotes dos fluxos do *dataset*.

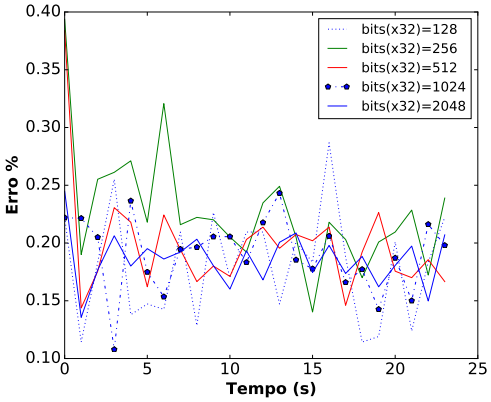


Figura 7. Taxas de erros na aplicação de contagem de tráfego para diferentes número de bits utilizados.

entre um par de endereços IP. Um fluxo é considerado ativo se algum switch da rede tiver recebido pelo menos um pacote dele nos últimos t ms.

A aplicação do switch utiliza uma tabela hash para armazenar quando foi a última vez que um pacote de um determinado fluxo foi recebido. Para cada pacote recebido, o switch primeiro extrai os endereços IP da fonte e do destino e então cria uma chave para a tabela hash através da concatenação dos endereços em ordem crescente. Essa chave é então utilizada para armazenar na tabela o tempo de chegada do pacote.

A aplicação do controlador requisita periodicamente ao switch sua tabela com os tempos de chegada dos pacotes. Ele então verifica se os fluxos registrados na tabela estão ativos checando quanto tempo se passou desde que os últimos pacotes foram recebidos. Quando um fluxo não está mais ativo, o controlar requisita ao switch que a entrada correspondente a ele seja apagada da tabela a fim de evitar novas verificações desnecessárias.

Vale a pena notar que a frequência com que o servidor requisita os dados do switch pode afetar o desempenho da aplicação. Requisições frequentes melhoram a precisão da aplicação, mas aumentam a quantidade de processamento no switch e no controlador. Já requisições infrequentes causam os efeitos inversos.

A implementação desta aplicação define o tempo de requisição do controlador e t como 500 ms por questões de desempenho. Os *traces* foram utilizados em um enlace simulado de 0.5 Mbps. A figura 8 mostra o quanto a medição do número de fluxos ativos pela aplicação diferiu da quantidade real durante os primeiros 30 minutos da transmissão de pacotes. A quantidade medida foi sempre similar à real, sendo o erro médio observado aproximadamente igual 2% e o erro máximo próximo de 10%. Isto indica que a aplicação consegue obter uma boa aproximação do número real de fluxos ativos na rede.

A aplicação também foi testada com um enlace de 1 Mbps, mas registrou-se problemas de desempenho. Por vezes, o switch não conseguiu processar todos os pacotes que chegavam e assim muitos deles foram descartados. A causa do problema parece ser a grande quantidade de trabalho exigida do switch, que precisa processar todos os pacotes que chegam e ainda atender aos pedidos de envio de tabela e remoção de entradas do controlador. Mais investigações são necessárias para confirmar/refutar tal hipótese.

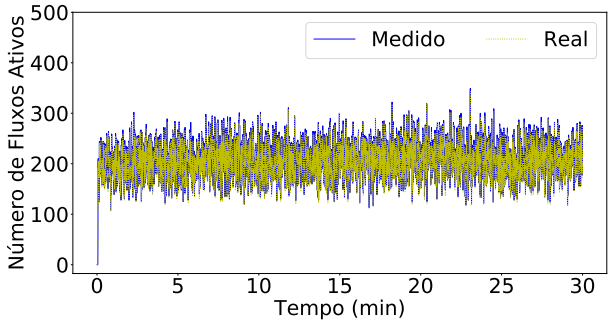


Figura 8. Contagem do número de fluxos ativos durante os 30 primeiros minutos de transmissão.

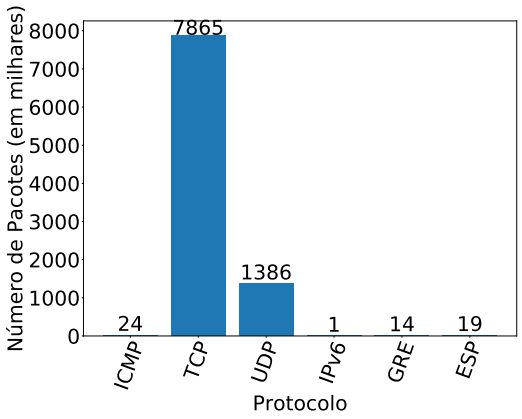


Figura 9. Contagem de pacotes presentes no *dataset* por protocolo IP.

4.6. Contagem da Quantidade de Pacotes por Protocolo IP

A última aplicação é capaz de contar o número de pacotes por protocolo IP em uma rede. Ela pode ser utilizar para prover estatísticas sobre o padrão do tráfego de pacotes ao administrador da rede.

A aplicação do switch é responsável por armazenar o número de pacotes por protocolo IP observados. Ela utiliza um *array* com 256 contadores (um para cada número de procolo IP), que são incrementados a medida que novos pacotes passam pelo switch. Por exemplo, quando um pacote TCP passa pelo switch, seu campo de número de protocolo IP é extraído e o contador associado a ele é incrementado.

Novamente, a aplicação do controlador requisita periodicamente o *array* de contadores ao switch. Ela então utiliza os dados recebidos para plotar um gráficos de barras onde é possível visualizar as estatísticas. Por exemplo, a figura 9 mostra o número de pacotes por protocolo IP contidos no *trace*. O controlador pode também periodicamente requisitar ao switch para zerar seus contadores para evitar a ocorrência de *overflows*.

5. Conclusões e Trabalhos Futuros

Este trabalho apresenta um conjunto de aplicações de monitoramento de rede utilizando a arquitetura BPFabric. Aplicações para a detecção de mudanças bruscas, detecção de *heavy hitters*, estimativa de distribuição dos tamanhos dos fluxos e contagem de tráfego foram implementadas utilizando estruturas de dados probabilísticas como o Bloom Filter, Count-Min Sketch, k-ary Sketch e o PCSA. Também foram implementadas aplicações baseadas em amostragem para realizar a contagem do número de fluxos ativos ao longo do tempo e o número de pacotes por protocolo IP que trafegam em uma rede.

As aplicações foram avaliadas utilizando dados reais coletados e disponibilizados pela CAIDA. Os resultados mostraram que as aplicações baseadas em sketches apresentaram resultados compatíveis com o esperado, utilizando pouca memória e apresentando resultados aproximados. Além disso, foi observado que as taxas de erros foram impactadas pelos tamanhos das estruturas utilizadas. As aplicações baseadas em amostragem também conseguiram cumprir seus objetivos, sendo o desempenho delas afetado pela quantidade de tráfego na rede, o que já era esperado de ocorrer.

Como trabalho futuro, pretende-se realizar a execução de mais testes com o objetivo de melhor determinar os limites das aplicações apresentadas. Além disto, deverão ser executadas avaliações de desempenho do sistema, de forma a garantir que não ocorram problemas como perdas de pacotes nas aplicações.

Agradecimentos

O presente trabalho foi realizado com apoio da Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Código de Financiamento 001.

Referências

Alipourfard, O., Moshref, M., and Yu, M. (2015). Re-evaluating measurement algorithms in software. In *Proceedings of the 14th ACM Workshop on Hot Topics in Networks*, HotNets-XIV, pages 20:1–20:7, New York, NY, USA. ACM.

Bloom, B. H. (1970). Space/time trade-offs in hash coding with allowable errors. *Communications of the ACM*, 13(7):422–426.

Bosshart, P., Daly, D., Gibb, G., Izzard, M., McKeown, N., Rexford, J., Schlesinger, C., Talayco, D., Vahdat, A., Varghese, G., and Walker, D. (2014). P4: Programming protocol-independent packet processors. *SIGCOMM Comput. Commun. Rev.*, 44(3):87–95.

CAIDA (2013). The CAIDA UCSD Anonymized Internet Traces - 2013. `http://www.caida.org/data/passive/passive_dataset.xml`.

Carter, J. L. and Wegman, M. N. (1979). Universal classes of hash functions. *Journal of computer and system sciences*, 18(2):143–154.

Cormode, G. and Muthukrishnan, S. (2005). An improved data stream summary: the count-min sketch and its applications. *Journal of Algorithms*, 55(1):58–75.

Duffield, N., Lund, C., and Thorup, M. (2003). Estimating flow distributions from sampled flow statistics. In *Proceedings of the 2003 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, SIGCOMM '03, pages 325–336, New York, NY, USA. ACM.

Estan, C. and Varghese, G. (2002). New directions in traffic measurement and accounting. *SIGCOMM Comput. Commun. Rev.*, 32(4):323–336.

Flajolet, P. and Martin, G. N. (1985). Probabilistic counting algorithms for data base applications. *J. Comput. Syst. Sci.*, 31(2):182–209.

Gupta, A., Harrison, R., Pawar, A., Birkner, R., Canini, M., Feamster, N., Rexford, J., and Willinger, W. (2017). Sonata: Query-driven network telemetry. *CoRR*, abs/1705.01049.

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*, SOSR '18, pages 8:1–8:7, New York, NY, USA.

Huang, Q., Jin, X., Lee, P. P. C., Li, R., Tang, L., Chen, Y.-C., and Zhang, G. (2017). Sketchvisor: Robust network measurement for software packet processing. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, SIGCOMM '17, pages 113–126, New York, NY, USA. ACM.

Jouet, S. (2017). BPFabric Implementation. Available at: `https://github.com/UofG-netlab/BPFabric`.

Jouet, S. and Pezaros, D. P. (2017). Bpfabric: Data plane programmability for software defined networks. In *Proceedings of the Symposium on Architectures for Networking and Communications Systems*, pages 38–48. IEEE Press.

Kreutz, D., Ramos, F. M., Verissimo, P. E., Rothenberg, C. E., Azodolmolky, S., and Uhlig, S. (2015). Software-defined networking: A comprehensive survey. *Proceedings of the IEEE*, 103(1):14–76.

Krishnamurthy, B., Sen, S., Zhang, Y., and Chen, Y. (2003). Sketch-based change detection: Methods, evaluation, and applications. In *Proceedings of the 3rd ACM SIGCOMM Conference on Internet Measurement*, IMC '03, pages 234–247, New York, NY, USA. ACM.

Kumar, A., Sung, M., Xu, J. J., and Wang, J. (2004). Data streaming algorithms for efficient and accurate estimation of flow size distribution. In *Proceedings of the Joint International Conference on Measurement and Modeling of Computer Systems*, SIGMETRICS '04/Performance '04, pages 177–188, New York, NY, USA. ACM.

Lantz, B., Heller, B., and McKeown, N. (2010). A network in a laptop: rapid prototyping for software-defined networks. In *Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks*, page 19. ACM.

McKeown, N., Anderson, T., Balakrishnan, H., Parulkar, G., Peterson, L., Rexford, J., Shenker, S., and Turner, J. (2008). Openflow: enabling innovation in campus networks. *ACM SIGCOMM Computer Communication Review*, 38(2):69–74.

Pacífico, R. D., Coelho, G. R., Vieira, M. A., and Nacif, J. A. (2018). Roteador sdn em hardware independente de protocolo com análise, casamento e ações dinâmicas. In *Simpósio Brasileiro de Redes de Computadores (SBRC)*, volume 36.

Schweller, R., Li, Z., Chen, Y., Gao, Y., Gupta, A., Zhang, Y., Dinda, P. A., Kao, M. Y., and Memik, G. (2007). Reversible sketches: Enabling monitoring and analysis over high-speed data streams. *IEEE/ACM Transactions on Networking*, 15(5):1059–1072.

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*, SOSR '17, pages 164–176, New York, NY, USA. ACM.

Tsai, P. W., Tsai, C. W., Hsu, C. W., and Yang, C. S. (2018). Network monitoring in software-defined networking: A review. *IEEE Systems Journal*, pages 1–12.

van Adrichem, N. L. M., Doerr, C., and Kuipers, F. A. (2014). Opennetmon: Network monitoring in openflow software-defined networks. In *2014 IEEE Network Operations and Management Symposium (NOMS)*, pages 1–8.

Yu, M., Jose, L., and Miao, R. (2013). Software defined traffic measurement with opensketch. In *Proceedings of the 10th USENIX Conference on Networked Systems Design and Implementation*, nsdi'13, pages 29–42, Berkeley, CA, USA. USENIX Association.