BitMatrix: Ferramenta para Geração de Matrizes de Tráfego Ingresso-Egresso usando *Bitmaps*

Brunner K. Nogueira[1], **Fernando A. F. Silveira**[1],
Renato E. N. Moraes[1,2], **Rodolfo S. Villaça**[1,2] *
[1] Núcleo de Estudos em Redes Definidas por Software (NERDS)
[2] Departamento de Tecnologia Industrial – DTI/UFES
{brunnerk,fernandofossi}@gmail.com, {rodolfo.villaca,renato.moraes}@ufes.br

Abstract. *Generating Ingress-Egress traffic matrix is an important function to estimate the number of data flows between nodes in a network. However, generate these matrices from collecting all flows passing through all links is an expensive operation due to the large amount of data that must be analyzed. We can use distributed techniques based on probabilistic data structures like bitmaps to enable a low cost estimation of these traffic matrices. Since it is a probabilistic technique, the estimation accuracy is subject to the parameter settings: size and access limit of the bitmaps. This paper presents a simulation tool to allow: i) the fine tuning of these parameters; ii) generating Ingress-Egress traffic matrices between nodes in the network; and iii) the dynamic display of data flows to support the analysis of the results.*

Resumo. *A geração de matrizes de tráfego Ingresso-Egresso é uma importante função para avaliar a quantidade de fluxos em uma rede. Entretanto, gerar essas matrizes a partir do tráfego medido nos enlaces é uma operação custosa devido à grande quantidade de dados que precisa ser coletada e analisada. Uma técnica distribuída que permite uma estimativa de baixo custo dessas matrizes, consiste na utilização de estruturas de dados probabilísticas do tipo bitmap. Por se tratar de uma técnica probabilística, a precisão da sua estimativa está condicionada ao ajuste dos parâmetros: tamanho e limite de acesso dos bitmaps. Assim, este artigo descreve uma ferramenta computacional que, através de simulação, permite: i) o ajuste fino destes parâmetros; ii) a geração da matriz de tráfego Ingresso-Egresso; e iii) a visualização dinâmica do fluxo de dados na rede para auxiliar a análise dos resultados obtidos.*

1. Introdução

A crescente demanda pelas redes tem como consequência um grande aumento no fornecimento de conteúdos e serviços diferenciados de entretenimento e comunicação. Muitos desses serviços são gerados, por exemplo, por *sites* de *streaming* de vídeos, tais como o Youtube e Netflix, por aplicativos de VoIP como o Skype e por *sites* de jogos *online*. Esses serviços exigem um alto índice de Qualidade de Serviço (*Quality of Service*) pois a perda e o atraso de pacotes tem um grande impacto na experiência dos seus usuários. Essa realidade faz com que o conhecimento e a análise das características de tráfego nas redes seja de extrema importância.

A partir da análise dessas características, os administradores podem tomar decisões como, por exemplo, aumentar a capacidade dos enlaces para remover possíveis

*Agradecimentos à FAPES, CNPq e RNP/FUTEBOL pelo financiamento parcial deste trabalho.

gargalos ou alterar o encaminhamento de tráfego por outros enlaces ou caminhos. Esse conhecimento pode ser construído através de medição de tráfego. Dentre as técnicas existentes para a caracterização do tráfego em redes encontra-se a matriz de tráfego [Paul Tune 2013]. Uma matriz de tráfego é uma representação abstrata do volume de dados trafegado entre conjuntos de pares ingresso (ponto onde os dados entram na rede) e egresso (ponto onde os dados saem da rede). Cada elemento da matriz representa o volume do tráfego de dados entre um par de nós da rede. O volume de dados geralmente é medido em número de *bytes* ou pacotes. Entretando, existe um alto custo associado à construção da matriz de tráfego de uma rede, principalmente em função da necessidade de coleta, armazenamento e análise minuciosa das informações em cada nó da rede para conhecer exatamente o volume de dados que trafegaram por eles.

A diminuição do custo de construção de uma matriz de tráfego pode ser alcançada por meio da substituição dessa coleta minuciosa de informações pela utilização de estruturas de dados probabilísticas do tipo *bitmap* [Zhao et al. 2005]. A estrutura de dados probabilística utilizada nesse trabalho, conhecida como *bitmap*, é, basicamente, um vetor de *bits*. No *bitmap* cada vez que um pacote passa por um nó da rede, um valor inteiro é gerado a partir da aplicação de uma função *hash* simples em alguns campos específicos do cabeçalho do pacote. Um *bit* do *bitmap* é então marcado na posição relativa ao valor inteiro dado pela aplicação do *hash*. Se todos os nós da rede aplicam a mesma função *hash* no pacote, as mesmas posições serão marcadas nos *bitmaps* de cada nó. A partir dessas marcações e dos *bitmaps* de cada nó é possível estimar o volume de tráfego entre pares de nós da rede. A técnica será melhor explicada na Seção 3.

A utilização desta técnica para estimação da matriz de tráfego reduz significativamente o custo da coleta de informações. Entretanto, essa mesma simplificação provoca perdas de precisão na matriz gerada. Essas imprecisões são geradas, por exemplo, quando ocorrem colisões de *hash* nas marcações dos *bitmaps*. Essa precisão pode ser ajustada por dois parâmetros de configuração: (1) tamanho do *bitmap* e (2) limite do número de acessos ao *bitmap*.

Neste contexto, este artigo propõe uma ferramenta de simulação computacional, denominada BitMatrix, que permite: i) o ajuste fino dos parâmetros de configuração dos *bitmaps*, ii) a geração da matriz de tráfego estimada de uma rede e iii) a visualização dinâmica do fluxo de dados nesta rede. A BitMatrix possibilita a experimentação de diversas redes e diferentes parâmetros de configuração dos *bitmaps* a partir da comparação com os as matrizes de tráfego reais, sendo esta a principal contribuição deste trabalho. A ferramenta baseia-se na técnica apresentada em [Zhao et al. 2005], cuja avaliação original foi realizada apenas através de simulação numérica.

A BitMatrix está publicamente disponível na plataforma Github[1] e possui como entradas: um grafo representando o conjunto de nós e enlaces de uma rede; um arquivo de captura de pacotes de rede padrão, do tipo .pcap; e os parâmatros de configuração dos *bitmaps*. Foi avaliada usando *traces* anonimizados de dados de tráfego real da Internet[2] e topologias reais de Sistemas Autônomos (AS) [3], ambos obtidos diretamente do CAIDA (*Center for Applied Internet Data Analysis*).

[1]https://github.com/***/***

[2]http://www.caida.org/data/passive/passive_2012_dataset.xml

[3]https://www.caida.org/research/topology/#Datasets

Os resultados mostraram a eficácia da ferramenta no auxílio ao ajuste fino dos parâmetros de configuração dos *bitmaps* em função da precisão obtida na estimação das matrizes de tráfego das redes simuladas.

O restante deste artigo está organizado na seguinte forma: a Seção 2 faz um levantamento dos trabalhos relacionados na literatura e destaca a principal contribuição deste artigo. A Seção 3 descreve a técnica de estimação das matrizes de tráfego a partir dos *bitmaps*. A Seção 4 apresenta a ferramenta, seus módulos e funcionalidades. Alguns resultados obtidos a partir da ferramenta BitMatrix são apresentados na Seção 5 enquanto a Seção 6 conclui o artigo e propõe trabalhos futuros e em andamento.

2. Trabalhos Relacionados

Segundo [Paul Tune 2013] as estratégias para construir matrizes de tráfego podem ser classificadas em dois tipos: medições indiretas ou diretas. Conceitualmente, medições indiretas estimam a matriz através de modelos de tráfego aplicados nas informações de estado da rede e nos dados disponibilizados pela infraestrutura de gerenciamento da rede. Já as estratégias de medição direta não necessitam de modelos nem de dados externos, pois realizam a medição observando diretamente o tráfego em múltiplos nós. De modo geral, estratégias de medição diretas são mais precisas do que as estratégias indiretas. A estratégia de medição implementada usando *bitmaps* é uma técnica de medição direta. A seguir serão apresentadas algumas técnicas similares.

As informações coletadas diretamente dos nós pelas técnicas de medição direta são observadas através de *packet traces*. Um *packet trace* representa uma coleção de cabeçalhos de pacotes e *timestamps*. Coletar esses *traces* é uma tarefa bastante custosa por dois motivos: primeiro, necessita-se de dispositivos específicos para a coleta de dados e, segundo, a quantidade de informações geradas é muito grande, podendo sobrecarregar os dispositivos de monitoramento. Uma alternativa interessante é a agregação dos pacotes em fluxos [Moshref et al. 2014].

Técnicas de amostragem de pacotes podem ser utilizadas para reduzir a sobrecarga nos elementos da rede. Nelas, os pacotes de ingresso são amostrados baseados em uma regra predeterminada, como no NetFlow e sFlow. Durante a amostragem é mais provável que se escolha um pacote de um fluxo maior, privilegiando assim esses fluxos e, consequentemente, degradando a qualidade dos resultados. Em [Duffield et al. 2003] e [Duffield et al. 2005] são propostos métodos mais eficazes de se amostrar fluxos com menor perda de qualidade nos resultados.

Outro problema da medição direta, além da grande quantidade de dados, é a múltipla contagem de fluxos. Nesse caso, um único fluxo pode ser amostrado mais de uma vez por vários elementos da rede aumentando o erro de amostragem. Em [Duffield and Grossglauser 2001] utiliza-se a pseudo aleatoriedade das funções de *hashing* para rastrear os fluxos na rede e assim resolver a múltipla contagem dos fluxos.

Independente da estratégia de amostragem adotada, deve-se lembrar de que a amostragem é, em sua essência, um processo com perdas. A perda de informações se traduz em erros ou ruídos nos dados. Para uma melhor utilização dos dados, a magnitude desses erros deve ser estimada. Essa dificuldade pode ser superada através do uso de estruturas de dados probabilísticas. Soluções baseadas em estruturas de dados probabilísticas (*sketches*) [Flajolet and Martin 1985] têm sido propostas em diversas áreas

do conhecimento, dentre elas o monitoramento de redes [Yu et al. 2013]. As estruturas de dados probabilísticas permitem estimar diversas métricas e possuem um forte compromisso entre a precisão das medidas obtidas e a quantidade de recursos de memória necessários ao seu armazenamento.

Nesse contexto, este trabalho contribui ao disponibilizar a implementação de um simulador computacional que permite o ajuste fino dos parâmetros de estruturas probabilisticas do tipo *bitmap* na geração de matrizes de tráfego estimadas. As matrizes de tráfego estimadas serão comparadas com matrizes reais, permitindo ao administrador da rede alcançar a maior precisão possível com o menor conjunto de dados possível.

3. Geração de Matrizes de Tráfego usando *bitmaps*

Um *bitmap*, ou mapa de *bits*, é uma estrutura de dados do tipo vetor cujos elementos nas suas respectivas posições podem assumir os valores 0 ou 1 (ou seja, *bits*). Um *bitmap* de tamanho n corresponde a um vetor de n *bits*. Todo *bitmap* está associado a um único dispositivo de rede. As operações permitidas nessa estrutura de dados são: leitura, escrita, armazenamento e instanciação.

O Algoritmo 1 mostra a procedimento de marcação dos *bitmaps* em cada dispositivo de rede monitorado. O algoritmo é executado toda vez que um pacote é recebido em uma interface de rede do dispositivo. O algoritmo recebe como entrada o pacote pkt recém chegado, o *bitmap* atualmente instanciado na memória do dispositivo e o parâmetro $limiteAcessosBitmap$ que define a quantidade máxima de acessos permitida ao *bitmap* instanciado e, consequentemente, o momento que esse deve ser armazenado em memória não-volátil (*flash* ou disco) e substituído por outro *bitmap* em memória volátil (TCAM).

O teste de substituição do *bitmap* é feito logo no início do Algoritmo 1, especificamente na linha 2. Se o número de acessos ao *bitmap* atualmente instanciado na memória do dispositivo atingir o limiar ($limiteAcessosBitmap$), o *bitmap* será armazenado permanentemente em disco na linha 3, caso contrário, mantém-se o *bitmap* atualmente instanciado. Ainda na linha 3, a função `ArmazenaBitmapEmDisco()` também grava permanentemente o instante de tempo (função `SystemTime()`) do armazenamento para que seja possível futuramente gerar a matriz de tráfego da rede durante um intervalo de tempo especificado. Na linha 4, um novo *bitmap* é criado em memória com n *bits*, tendo todas as suas posições zeradas.

Na linha 6, faz-se a seleção de alguns campos fixos do pacote pkt: endereços IP de origem e destino, portas de origem e destino, tipo do protocolo de transporte e o primeiro *Byte* do campo de dados do pacote, se houver. Esse primeiro *Byte* do campo de dados serve para distinguir diferentes pacotes de dados de um mesmo fluxo. Esses campos são atribuídos ao registro $cabecalhoPacote$. Em seguida, na linha 7, aplica-se a função `Hash` neste cabeçalho para gerar uma posição inteira i no intervalo $0 \leq i < n$, que é então utilizada na linha 8 para atribuir à posição i do bitmap bmp o valor 1, correspondendo a algo como "o pacote pkt passou por aqui e eu registrei essa passagem na posição i". Quando a posição i do *bitmap* possui valor 1, diz-se que a posição i está marcada. Na linha 9 o atual *bitmap* instanciado em memória é retornado com a devida marcação.

Os *bitmaps* deverão ser instalados em todos os nós de Ingresso e Egresso da rede que se deseja monitorar. Sobre esse ponto é importante ressaltar as seguintes informações:

i) os *bitmaps* devem possuir o menor tamanho n possível para não ocupar muito espaço na memória volátil (TCAM) dos elementos de rede monitorados [Yu et al. 2013] e; ii) a função de *hashing* utilizada deve ser de fácil implementação em *hardware* para não tornar-se um gargalo no fluxo dos pacotes no elemento de rede. As funções de *hashing* propostas em [Carter and Wegman 1979] satisfazem essa condição.

Algoritmo 1: Gerenciamento dos Bitmaps

Entrada: $pkt \leftarrow$ Pacote recém chegado na interface do dispositivo de rede
Entrada: $bmp \leftarrow$ Bitmap atualmente instanciado na memória do dispositivo de rede
Entrada: $limiteAcessosBitmap \leftarrow$ limiar de número de acessos ao *bitmap*
Saída: Bitmap marcado na posicao correspondente ao pacote pkt

```
1  início
2      se NumeroAcessos(bmp) ≥ limiteAcessosBitmap então
3          ArmazenaBitmapEmDisco(bmp, SystemTime())
4          bmp ← NovoBitmapEmMemoria(n)
5      fim se
6      cabecalhoPacote ← SelecionaCampos(pkt)
7      i ← Hash(cabecalhoPacote)
8      bmp[i] ← 1
9      retorna bmp
10 fim
```

A Figura 1 mostra um exemplo de uma rede com um nó de Ingresso ($R1$) e um nó de Egresso ($R2$), ambos com *bitmaps* já instanciados, e um nó intermediário não monitorado. É condição para o funcionamento da técnica apresentada nesta seção que todos os nós monitorados (com *bitmaps* instanciados) apliquem exatamente a mesma função de *hashing* sob os mesmos campos do cabeçalho dos pacotes de um fluxo. Desta forma, dado um mesmo pacote, uma mesma posição do *bitmap* será marcada nos elementos de rede $R1$ e $R2$. A Figura 1 também mostra que um controlador de rede é utilizado para coletar, periodicamente, todos os *bitmaps* armazenados em memória não volátil de cada dispositivo e gerar a matriz de tráfego da rede a partir de pares de *bitmaps*.

Na técnica implementada seleciona-se dois elementos monitorados distintos de uma mesma rede e um intervalo de tempo arbitrário de monitoramento. De cada elemento é extraído o conjunto de *bitmaps* marcados durante o intervalo de tempo definido. Em seguida, os dois conjuntos de *bitmaps* são comparados. Quanto mais *bits* em comum eles tiverem nas mesmas posições, mais pacotes em comum passaram por esses elementos. Se a interseção de *bits* em uma mesma posição entre dois conjuntos de *bitmaps* for pequena, significa que poucos pacotes em comum trafegaram por esses elementos. Maiores detalhes sobre todo o processo de estimação das matrizes a partir dos conjuntos de *bitmaps* entre todos os nós de uma rede podem ser encontrados em [Zhao et al. 2005].

É importante destacar que as colisões de *hashing* fazem com que dois ou mais pacotes distintos marquem a mesma posição do *bitmap*, prejudicando a precisão da técnica utilizada. Como a função de *hashing* utilizada deve ser simples, esse problema não pode ser negligenciado e precisa ser corretamente equacionado. Duas possíveis soluções podem ser adotadas. Na primeira solução, pode-se aumentar o tamanho dos *bitmaps*. Entretanto, aumentar indefinidamente o tamanho dos *bitmaps* não é uma boa solução pois eles

devem caber na memória dos elementos de rede.

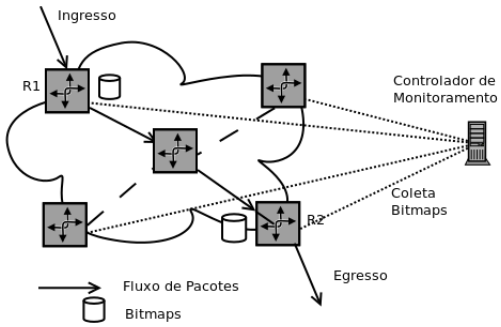

Figura 1. Trecho de rede com nós de Ingresso, Egresso, Fluxo de Pacotes e *Bitmaps*

A segunda solução propõe a geração de novos *bitmaps* sempre que o número de pacotes amostrados na rede atingir um limiar pré-definido ($limiteAcessosBitmap$ no Algoritmo 1), antecipando-se a possíveis colisões. Entretanto, quanto menor for esse limiar, mais *bitmaps* serão gerados em um mesmo intervalo de tempo de monitoramento, o que aumenta a quantidade de *bitmaps* que deverão ser requisitados e processados pelo controlador para geração das matrizes de tráfego entre dois pares de dispositivos.

Como a escolha do tamanho e do limite do número de acessos aos *bitmaps* impacta diretamente na precisão da matriz gerada, o ajuste desses parâmetros torna-se essencial na obtenção dos melhores resultados. A experimentação permite ao administrador encontrar os melhores valores para os parâmetros em função da topologia e da característica de tráfego da rede que se deseja monitorar.

4. Projeto e Implementação da Ferramenta BitMatrix

A BitMatrix é uma ferramenta de simulação de tráfego focada na geração das matrizes de tráfego utilizando *bitmaps*. Conforme mostrado na Figura 2, a ferramenta é dividida em 3 módulos principais: `Entrada`, `Processamento` e `Saída`. A seguir serão descritos cada um dos módulos, suas funcionalidades e o relacionamento entre eles.

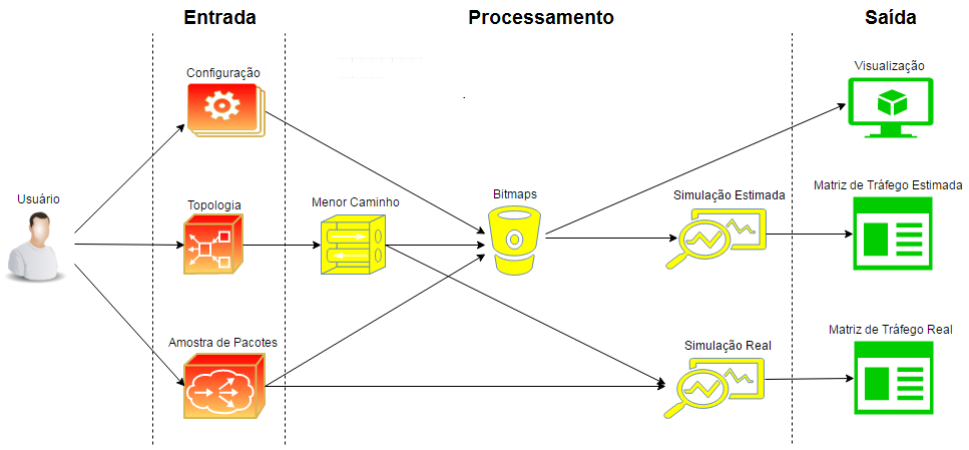

Figura 2. Módulos da Ferramenta BitMatrix

4.1. Módulo de Entrada

O `Módulo de Entrada` permite ao usuário definir o cenário da simulação a ser executada. Três blocos de entrada estão disponíveis:

- `Configuração`: define os parâmetros desejados para os *bitmaps*, ou seja, o tamanho e o limite do número de acessos. Neste módulo define-se também o período das amostras de pacotes para o qual se deseja gerar a matriz.
- `Topologia`: faz a leitura das topologias das redes simuladas, representadas como grafos descritos através dos seus vértices e arestas. Nos testes foram utilizados excertos de sistemas autônomos reais obtidos diretamente no CAIDA. Ao formato original foi adicionada a quantidade de vértices e arestas do grafo para facilitar o processo de leitura e interpretação dos arquivos.
- `Amostras de Pacotes`: faz a leitura dos dados de tráfego (pacotes) no formato .pcap. Cada pacote contém: *timestamp*, endereço IP de origem, endereço IP de destino, porta de origem, porta de destino, tipo de protocolo e o primeiro *Byte* de dados do pacote. Nos testes realizados foram novamente utilizados *traces* de dados reais obtidos no CAIDA.

4.2. Módulo de Processamento

O `Módulo de Processamento` executa a simulação definida pelas entradas e seleções feitas no módulo anterior. No decorrer da simulação é feita a marcação dos *bitmaps*, a geração da matriz de tráfego estimada e a geração da matriz de tráfego real. Em seguida, as duas matrizes, a estimada e a real, podem ser comparadas e a escolha dos parâmetros de simulação pode ser avaliada.

A simulação inicia-se pelo bloco `Menor Caminho`. Nesse bloco é construído o conjunto de caminhos mínimos entre cada par de nós da rede definida em `Topologias`. Os caminhos mínimos são descobertos aplicando-se o algoritmo de *Shortest Path* definido em [Dijkstra 1959].

O segundo passo na execução deste módulo consiste em, dado o conjunto de caminhos mínimos (`Menor Caminho`) e os dados de tráfego anonimizados (`Amostras de Pacotes`), simular o tráfego na rede, ou seja o fluxo de pacotes que atravessa os vértices e as arestas do grafo. Para tanto, associa-se, de modo deterministico, blocos de endereços IP de origem e destino a pares Ingresso-Egresso na rede simulada.

No bloco de `Simulação Real` a simulação do tráfego é feita de maneira que a quantidade de pacotes trafegados por cada um dos vértices é contabilizada sem a utilização dos *bitmaps*. Esse resultado, obtido através da medição direta de tráfego sem perda de precisão, é utilizado no final da execução do bloco para construção da matriz de tráfego real que serve como base na avaliação da precisão das matrizes estimadas.

No bloco de `Simulação Estimada` é feita a geração de matrizes de tráfego utilizando-se a técnica descrita na Seção 3. Esse bloco é executado após a criação dos *bitmaps* em cada nó da rede, de acordo com os parâmetros definidos no bloco `Configurações`. Durante a execução deste bloco a simulação do tráfego é feita de maneira que, cada vez que um pacote passa por um dispositivo, marca-se uma posição no *bitmap* deste dispositivo, conforme Algoritmo 1.

Após o processamento de todos os pacotes pelo simulador a matriz de tráfego estimada pode ser construída a partir de todos os *bitmaps* armazenados ao longo da execução do bloco `Simulação Estimada`.

4.3. Módulo de Saída

O `Módulo de Saída` apresenta as informações geradas no `Módulo de Processamento` facilitando a análise dos parâmetros de configuração.

No bloco `Visualização`, é possível apresentar o caminho de cada um dos pacotes trafegados pela rede. A BitMatrix usa o Graphviz como ferramenta de visualização. Como exemplo, na Figura 3, os arcos azuis mostram o caminho que um pacote específico percorre na rede definida pelo usuário. Neste exemplo, o vértice 1 atua como Ingresso e o vértice 9 representa o nó de Egresso do pacote na rede.

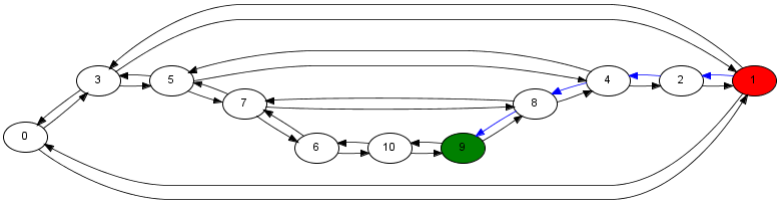

Figura 3. Exemplo de visualização de um pacote trafegando na rede

No bloco `Matriz de Tráfego Real`, a matriz de tráfego real construída ao final da execução do bloco `Simulação Real` e armazenada em um arquivo `.csv`. No bloco `Matriz de Tráfego Estimada` a matriz de tráfego estimada gerada ao final da execução do bloco `Simulação Estimada` também é salva em arquivo `.csv`.

A partir dessas matrizes o administrador da rede, responsável pela configuração dos *bitmaps*, pode fazer uma avaliação das diferentes configurações. Caso não esteja satisfeito com a precisão obtida nas matrizes de tráfego estimadas, o mesmo pode reconfigurar os *bitmaps* e executar o BitMatrix novamente. Ou seja, a ferramenta BitMatrix permite que seja feito o ajuste fino dos parâmetros de configuração dos *bitmaps* até que eles atendam à precisão desejada para a rede simulada.

5. Avaliação e Resultados

De acordo com [Zhao et al. 2005], a estratégia de medição usando estruturas de dados probabilísticas do tipo *bitmap* é mais adequada à detecção de fluxos elefantes (fluxos de maior intensidade) entre pares de dispositivos de rede. Como a ferramenta BitMatrix faz uso dessa estratégia de estimação da matriz de tráfego, a visualização e demonstração dos resultados serão apresentados em função dos 10 (dez) maiores tráfegos entre nós de Ingresso-Egresso na rede simulada, chamados de Top 10. Quanto maior a proximidade da medição entre a análise real e a análise estimada, melhor será o ajuste entre tamanho e limite de acessos aos *bitmaps*

Na avaliação da ferramenta foi utilizada uma amostra com aproximadamente 100.000 pacotes de tráfego real (anonimizado) e a topologia Abilene, obtida diretamente do CAIDA em um intervalo de aproximadamente 2 min. Nestas condições foram testados diversos tipos de configurações, com tamanhos de *bitmaps* variando entre 64 e 64k *bits* e limites de número de acessos ao *bitmap*, variando em 30% e 50% de seu tamanho total.

Tabela 1. Análise de Resultados - Número de Pacotes Amostrados

	Número de Pacotes								
Posição	Real	64 bits		512 bits		16.384 bits		65.536 bits	
Top 10		30%	50%	30%	50%	30%	50%	30%	50%
#1	43328	41.977	41.960	41.509	41.790	41.209	41.318	40.471	40.046
#2	34267	34.628	34.275	33.895	34.051	33.874	33.950	33.115	32.720
#3	28134	28.352	27.924	27.809	28.126	27.636	27.874	27.093	26.792
#4	27079	27.202	27.118	26.989	27.051	26.556	26.490	26.214	25.727
#5	22558	22.680	22.744	22.597	22.246	22.342	22.241	21.477	20.585
#6	22193	22.449	22.477	22.173	21.488	21.880	21.641	21.365	20.413
# 7	21311	21.558	21.476	21.706	21.129	21.400	21.055	20.058	19.072
#8	19971	19.951	19.629	19.891	19.556	19.586	19.343	19.199	18.508
#9	19466	19.306	19.236	19.538	19.538	19.157	18.961	18.709	18.261
#10	19402	19.235	18.963	19.221	18.988	19.036	18.766	18.659	18.244

Tabela 2. Análise de Resultados - Top 10 Pares de Nós

	Pares de Nós								
Posição	Real	64 bits		512 bits		16.384 bits		65.536 bits	
Top 10		30%	50%	30%	50%	30%	50%	30%	50%
#1	(5,9)	(5,9)	(5,9)	(5,9)	(5,9)	(5,9)	(5,9)	(5,9)	(5,9)
#2	(9,10)	(9,10)	(9,10)	(9,10)	(9,10)	(9,10)	(9,10)	(9,10)	(9,10)
#3	(5,10)	(5,10)	(5,10)	(5,10)	(5,10)	(5,10)	(5,10)	(5,10)	(5,10)
#4	(3,5)	(3,5)	(3,5)	(3,5)	(3,5)	(3,5)	(3,5)	(3,5)	(3,5)
#5	(4,6)	(4,6)	(4,6)	(4,6)	(4,6)	(4,6)	(4,6)	(4,6)	(2,4)
#6	(2,4)	(2,4)	(2,4)	(2,4)	(2,4)	(2,4)	(2,4)	(2,4)	(4,6)
#7	(6,8)	(6,8)	(6,8)	(6,8)	(6,8)	(6,8)	(6,8)	(6,8)	(6,8)
#8	(2,3)	(2,3)	(2,3)	(2,3)	(2,3)	(2,3)	(2,3)	(2,3)	(2,5)
#9	(2,5)	(2,5)	(2,5)	(3,9)	(3,9)	(3,9)	(3,9)	(3,9)	(3,9)
#10	(3,9)	(3,9)	(3,9)	(2,5)	(2,5)	(2,5)	(2,5)	(2,5)	(2,3)

Dentre os testes realizados, foram selecionados 8 tipos de configurações para serem exibidas, e seus desempenhos podem ser avaliados pelas Tabelas 1 e 2. A Tabela 1 mostra o valor absoluto de tráfego medido em número de pacotes em cada um dos Top 10 pares de nós das análises estimadas por cada configuração e a sua comparação com o tráfego real medido. Já a Tabela 2, mostra quais são os top 10 pares de nós.

Realizando uma comparação entre os Top 10 pares de nós da Matriz Estimada com os Top 10 pares de nós da Matriz Real é possível avaliar a precisão de cada uma das configurações de *bitmaps* analisadas pela ferramenta proposta.

Como pode ser visto, aumentar o tamanho do *bitmap* não necessariamente é a melhor solução. Quando o *bitmap* cresce demais, a precisão do método cai um pouco, errando algumas posições no Top 10. Também é possível perceber um maior erro de estimativa das configurações de 50% em relação às de 30% do limite do número de acessos, o que é causado pelo aumento das colisões de *hashing*.

6. Conclusões e Trabalhos Futuros

Este artigo apresentou uma ferramanta que implementa a estratégia de construção de matrizes de tráfego através de medição direta usando estruturas de dados probabilísticas do tipo *bitmap*. A BitMatrix possibilita a experimentação dos diferentes parâmetros de configuração dos *bitmaps* que afetam a precisão das matrizes de tráfego estimadas. Essa precisão pode ser avaliada a partir da comparação com as matrizes de tráfego reais, obtidas através da simples contagem de pacotes.

A principal contribuição deste artigo é permitir que a técnica possa ser simulada e ter seus resultados avaliados com uso de dados e redes reais, auxiliando pesquisadores e administradores que trabalham com o uso de estruturas de dados probabilísticas em monitoramento de redes. Essa possibilidade é inédita até onde temos conhecimento.

Como trabalho futuro, a partir dos resultados obtidos em simulação, pretende-se responder ao seguinte questionamento: "É possível estimar a matriz de tráfego de toda a rede sem instalar *bitmaps* em todos os nós? Em caso afirmativo, quais são os pontos estratégicos de medição e qual será o procedimento para estimação dessas matrizes?".

Referências

Carter, J. and Wegman, M. N. (1979). Universal classes of hash functions. *Journal of Computer and System Sciences*, 18(2):143 – 154.

Dijkstra, E. W. (1959). A note on two problems in connexion with graphs. *Numerische mathematik*, 1(1):269–271.

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.

Duffield, N., Lund, C., and Thorup, M. (2005). Learn more, sample less: control of volume and variance in network measurement. *IEEE Transactions on Information Theory*, 51(5):1756–1775.

Duffield, N. G. and Grossglauser, M. (2001). Trajectory Sampling for Direct Traffic Observation. *IEEE/ACM Trans. Netw.*, 9(3):280–292.

Flajolet, P. and Martin, G. N. (1985). Probabilistic Counting Algorithms for Data Base Applications. *J. Comput. Syst. Sci.*, 31(2):182–209.

Moshref, M., Yu, M., Govindan, R., and Vahdat, A. (2014). DREAM: Dynamic Resource Allocation for Software-defined Measurement. *SIGCOMM Comput. Commun. Rev.*, 44(4):419–430.

Paul Tune, M. R. (2013). Internet Traffic Matrices. In Haddadi, H. and Bonaventure, O., editors, *Recent Advances in Networking*, chapter 3. ACM SIGCOMM eBook.

Yu, M., Jose, L., and Miao, R. (2013). Software Defined Traffic Measurement with OpenSketch. In *10th USENIX Symposium on Networked Systems Design and Implementation (NSDI 13)*, pages 29–42, Lombard, IL. USENIX.

Zhao, Q. G., Kumar, A., Wang, J., and Xu, J. J. (2005). Data Streaming Algorithms for Accurate and Efficient Measurement of Traffic and Flow Matrices. *SIGMETRICS Perform. Eval. Rev.*, 33(1):350–361.