EFM: Elephant Flow Manager - Uma Arquitetura para Detecção e Tratamento de Fluxos Elefantes em DCNs

André Luiz B. Rocha[1], **Fábio L. Verdi**[1]

[1]Departamento de Computação (DComp)
Universidade Federal de São Carlos (UFSCar)
Sorocaba, SP, Brasil

debeltrami@gmail.com, verdi@ufscar.br

Abstract. *The elephant flows typically are primarily responsible for network congestion, causing impacts on applications that are intolerant to delays. Although there are several works in the literature that analyze this type of traffic, none of them consider the impact of elephant flow size when perfoming the re-routing of them, influencing the Flow Completion Time (FCT) and throughput. In this way, this article presents EFM (Elephant Flow Manager) as a complete solution for monitoring and re-routing elephant flows in DCNs (Data Center Networks). The tests performed show that the size of the elephant flows influences the throughput and the FCT of the applications and, therefore, deserves to be observed. The evaluation of the solution was performed comparing the joint use of EFM with ECMP. In all tests, our re-routing solution was better when compared to pure routing with only ECMP. Then, this work presents three main contributions: (1) proposes and implements an architecture for the detection and treatment of elephant flows in DCNs (2) highlights the importance of the size of elephant flows in the re-routing of these in DCNs (3) the EFM in conjunction with the ECMP reduces the energy consumption in the DCNs, since the FCT of the flows is decreased.*

Resumo. *Os fluxos elefantes, tipicamente, são os principais responsáveis pelos congestionamentos na rede, causando impactos em aplicações que são intolerantes a atrasos. Muito embora existam vários trabalhos na literatura que analisam este tipo de tráfego, nenhum deles considera o impacto do tamanho do fluxo elefante ao realizar o re-roteamento dos mesmos, influenciando diretamente no Flow Completion Time (FCT) e no throughput dos fluxos. Desta forma, este artigo apresenta o EFM (Elephant Flow Manager) como uma solução completa de monitoramento e re-roteamento de fluxos elefantes em DCNs (Data Center Networks). Os testes realizados mostram que o tamanho dos fluxos elefantes influencia no FCT e no throughput das aplicações e, portanto, merece ser observado. A avaliação da solução foi realizada comparando o uso conjunto do EFM com o mecanismo ECMP. Em todos os testes, nossa solução de re-roteamento foi melhor quando comparada ao roteamento puro com o ECMP. Portanto, este trabalho apresenta três contribuições principais: (1) propõe e implementa uma arquitetura para a detecção e tratamento de fluxos elefantes em DCNs (2) destaca a importância em observar o tamanho dos fluxos elefantes no re-roteamento destes em DCNs (3) o EFM em conjunto com o ECMP reduz o consumo de energia nas DCNs, devido a diminuição no FCT dos fluxos.*

1. Introdução

As DCNs transmitem um grande volume de dados diariamente, demandando que a sua arquitetura seja escalável e o roteamento dos fluxos tenha um baixo custo computacional. Com o intuito de prover tais características, as DCNs são construídas baseadas em topologias *fat-tree* ou VL2 [Al-Fares et al. 2008]. Outra característica das topologias citadas é o fato delas possuírem múltiplas rotas com o mesmo custo entre um par de *hosts*, viabilizando o uso de estratégias de roteamento baseadas no ECMP (*Equal Cost Multipathing*). Estudos realizados em diversos *Data Centers* (DCs) afirmam que apenas 10% dos fluxos trafegam a maior parte do volume de dados, transmitindo mais de 80% da quantidade de bytes de toda a rede. Tais fluxos são denominados fluxos elefantes [Benson et al. 2010] e têm recebido grande atenção da comunidade científica.

O ECMP é uma estratégia de roteamento comumente utilizada em DCNs e possui a seguinte premissa, todos os caminhos da rede devem ter o mesmo custo para que seja realizado o roteamento dos fluxos. As rotas para os fluxos são calculadas a partir da *hash* das 5-tuplas do cabeçalho TCP/IP (IP origem, IP destino, porta origem, porta destino, protocolo). A facilidade de implementação e baixo custo computacional são seus principais pontos positivos. Entretanto, a quantidade de fluxos em DCNs é demasiadamente alta, ocorrendo muitos casos de colisão nas entradas da *hash*. Este problema é agravado em DCNs pois o ECMP não distingue fluxos elefantes dos demais fluxos, denominados camundongos [Xu and Li 2014]. Por este motivo, as colisões entre fluxos elefantes afetam diretamente o desempenho das aplicações, impactando negativamente o FCT (*Flow Completion Time*) e o *throughput* dos fluxos. Tal impacto pode até mesmo aumentar o consumo de energia dos DCs, como foi discutido em [Katta et al. 2016].

Com o intuito de minimizar o impacto dos fluxos elefantes em DCNs as soluções mais comuns na literatura são: efetuar o re-roteamento dos mesmos, como nos trabalhos apresentados em [Zhao et al. 2017, Kanagevlu and Aung 2015] ou subdividir os fluxos elefantes em fluxos camundongos, como pode ser visto em [Xu and Li 2014]. Recentemente, soluções utilizando o MPTCP (*MultiPath TCP*), como por exemplo em [Zhao et al. 2017, Silva and Verdi. 2017], vêm sendo estudadas. Tais soluções se beneficiam dos múltiplos caminhos existentes na rede para encaminhar os fluxos de modo a maximizar os recursos da mesma. Entretanto, nenhuma das soluções atualmente encontradas na literatura consideram o tamanho do fluxo elefante e o impacto disto no momento do re-roteamento. As soluções também não consideram que o re-roteamento de um fluxo elefante nem sempre trará benefícios. É necessário, portanto, verificar se as rotas disponíveis para receber o fluxo elefante irão efetivamente melhorar o desempenho da rede.

O objetivo deste trabalho é apresentar o EFM (*Elephant Flow Manager*) como uma solução para detectar os fluxos elefantes na rede e tomar decisões de re-roteamento. O monitoramento consiste em observar os enlaces da rede a fim de identificar se um congestionamento está prestes a acontecer. Caso a ocupação de um dos enlaces ultrapasse um determinado limiar o EFM verifica se há fluxos elefantes e atua nos mesmos de modo a otimizar a ocupação deste enlace. O EFM diferentemente das outras soluções propostas utiliza a informação do tamanho dos fluxos elefantes presentes na rede e o estado atual da rede em termos de ocupação dos enlaces, para que seja tomada uma decisão de re-roteamento. Para este trabalho, tamanho de um determinado fluxo elefante se refere à taxa de transmissão em bits por segundo (bps) do mesmo.

2. Fundamentação Teórica

O objetivo desta seção é facilitar a compreensão do trabalho, apresentando a fundamentação teórica básica necessária para um melhor entendimento do EFM. Na Sub-Seção 2.1, apresentamos os principais trabalhos relacionados a este e uma breve discussão sobre as principais diferenças deles com relação a nossa proposta. Na Sub-Seção 2.2 será discutida características importantes sobre os fluxos elefantes.

2.1. Trabalhos Relacionados

Os principais trabalhos que tentam minimizar o impacto dos fluxos elefantes em DCNs são apresentados a seguir. As principais diferenças que eles possuem entre si é o modo no qual os fluxos elefantes são detectados e a solução proposta para minimizar o impacto dos mesmos.

O trabalho proposto pelo Hedera [Al-Fares et al. 2010] implementa dois algoritmos para realizar o escalonamento de fluxos em DCNs. O *Global First Fit* e o *Simulated Annealing* foram os algoritmos implementados com o intuito de rotear os fluxos elefantes que se iniciam na rede. É proposto também um estimador de tráfego que calcula o quanto cada fluxo será capaz de ocupar na rede. Tal solução é um dos estados da arte em trabalhos com fluxos elefantes. Entretanto, o custo de rotear todos os fluxos elefantes pode ser um *overhead* na solução, assim como acreditamos que o estimador de tráfego proposto pelos autores não seja muito comum e fácil de se implementar em DCNs.

A proposta TinyFlow [Xu and Li 2014] atua em conjunto com o protocolo de roteamento ECMP. A ideia do TinyFlow é utilizar o controlador OpenFlow para dividir um fluxo elefante em vários fluxos camundongos e então aplicar o ECMP para rotear os fluxos com maior eficiência, pois todos os fluxos da rede seriam considerados camundongos. Em contra partida, como o trabalho de detecção e divisão dos fluxos são feitos no controlador, ocorre o aumento no processamento do mesmo, gerando maior latência nos fluxos de modo a prejudicar o desempenho de determinadas aplicações.

No trabalho Mahout [Curtis et al. 2011], é demonstrada uma arquitetura de gerenciamento de tráfego com baixo *overhead*. Para estabelecer a baixa sobrecarga, a detecção dos fluxos elefantes é feita no *end-host*. Uma camada de monitoramento é adaptada na pilha de rede do sistema operacional do *end-host* permitindo a sinalização de um possível fluxo elefante. Quando o fluxo elefante é detectado, ele é repassado para o controlador OpenFlow que se responsabiliza pela ação a ser tomada. Quando comparado com outros trabalhos, a proposta acaba sendo interessante devido a sua rápida detecção dos fluxos, baixo custo de monitoramento e baixa utilização de recursos nos comutadores. Entretanto, ao levarmos para o cenário de DC onde a maioria dos servidores possuem diversas máquinas virtuais rodando, encontramos um desafio que é alterar o núcleo de cada uma dessas máquinas para suportar tal solução.

As principais diferenças entre os trabalhos descritos acima e o EFM são: (1) o EFM considera o tamanho dos fluxos e o estado atual da rede para tomar decisões de re-roteamento; (2) um fluxo apenas será re-roteado pelo EFM caso haja pelo menos um enlace com a ocupação superior a um limiar, de modo a minimizar o processamento e evitar possíveis congestionamentos; (3) o EFM re-roteia apenas um fluxo elefante por iteração e não re-roteia o mesmo fluxo mais de uma vez, minimizando assim re-roteamentos desnecessários; (4) Por fim, o EFM é capaz de decidir se é benéfico ou não efetuar o

re-roteamento de um determinado fluxo elefante. Ou seja, um fluxo elefante só será re-roteado caso haja alguma rota que comporte o mesmo sem que exceda um limiar definido, com isso evitamos casos em que o re-roteamento piore o desempenho da rede como um todo.

2.2. Fluxos Elefantes

Na literatura, a definição do que são fluxos elefantes varia de acordo com o trabalho estudado. Sendo assim, identificamos com base nos estudos realizados, três definições que são comumente utilizadas para caracterizar os fluxos elefantes. A primeira delas define o fluxo elefante como aquele no qual foi transmitido uma grande quantidade de *bytes*, como por exemplo no Mahout [Curtis et al. 2011]. Outra abordagem encontrada caracteriza como fluxo elefante aquele no qual teve um longo tempo de vida na rede, ou seja, os que tiveram uma duração significativa na rede, como no TinyFlow [Xu and Li 2014]. Por último, no trabalho Hedera [Al-Fares et al. 2010], o fluxo elefante é definido como o fluxo que está transmitindo a uma taxa elevada. Neste último caso, a literatura define como elefante o fluxo que esteja ocupando mais de 10% da capacidade do enlace. Portanto, encontramos diferentes definições do que são fluxos elefantes, mas de modo geral, são os fluxos que prejudicam a rede de alguma forma afetando o desempenho da DCN prejudicando a execução de aplicações intolerantes a atraso.

3. EFM - Elephant Flow Manager

Nesta seção iremos apresentar o EFM, uma arquitetura para a detecção e re-roteamento de fluxos elefantes em DCNs. Na Sub-Seção 3.1, descreveremos a arquitetura proposta, assim como as principais funcionalidades dos componentes presentes na mesma. Na Sub-Seção 3.2 será detalhado o funcionamento da arquitetura evidenciando as suas funções mais importantes. Por fim, na Sub-Seção 3.3 os detalhes de implementação serão apresentados.

3.1. Arquitetura

A proposta deste trabalho denominada de EFM, define e implementa uma arquitetura completa para o tratamento de fluxos elefantes em DCNs. Ela contempla desde o monitoramento da rede para detecção de tais fluxos e possíveis pontos de congestionamento, até a atuação na DCN por meio do re-roteamento dos fluxos elefantes. Portanto, este trabalho visa otimizar o FCT e o *throughput* dos fluxos em DCNs através do re-roteamento dos fluxos elefantes. Tal re-roteamento é realizado observando a taxa de transmissão de cada fluxo presente nos pontos de congestionamento, além do estado atual da rede com relação a ocupação de seus enlaces.

A arquitetura do EFM é composta por três módulos, o *Monitoring Module*, o *Elephant Detector Module* e o *Actuator Module*. Cada módulo é responsável por funcionalidades específicas que contemplam desde o monitoramento da rede até a atuação, caso haja pelo menos um ponto de congestionamento na DCN. Além dos módulos principais o EFM necessita de uma base de dados e do controlador OpenFlow. Na Figura 1, ilustramos os módulos que compõem nossa arquitetura, assim como as interações entre eles.

Os módulos propostos e suas responsabilidades são descritos a seguir:

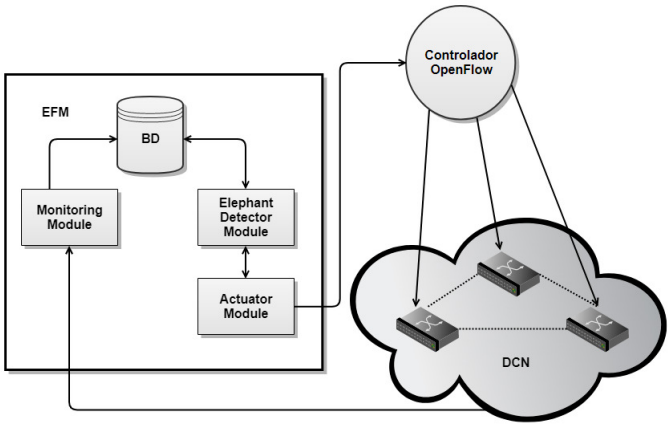

Figura 1. Arquitetura proposta do EFM.

- **Monitoring Module**: este módulo é responsável pela coleta dos dados de monitoramento de cada elemento de rede e armazená-los em uma base de dados. Através dos dados coletados, o *Monitoring Module* verifica pontos de congestionamento, ou seja, observa se há enlaces com taxas de ocupação acima do limiar definido. É importante destacar que nossa solução atua apenas caso encontre pontos de congestionamento que atinjam o limiar definido. Caso contrário, nenhum re-roteamento é realizado;
- **Elephant Detector Module**: este módulo é responsável por detectar fluxos elefantes presentes nos pontos de congestionamento (filtrados pelo *Monitoring Module*). Os fluxos elefantes serão então ordenados crescentemente conforme a taxa de transmissão dos mesmos. A partir disso, escolhe-se qual fluxo re-rotear, o maior ou o menor, notificando o *Actuator Module*;
- **Actuator Module**: este módulo é responsável por encontrar rotas para os fluxos elefantes detectados pelo *Elephant Detector Module*. Uma rota é considerada válida se ela é capaz de receber o fluxo elefante sem ultrapassar o limiar definido como ponto de congestionamento. Por fim, caso encontre uma rota, este módulo cria as regras OpenFlow e as envia para a *API REST* do controlador.

A base de dados armazena a topologia da DCN assim como, a taxa de ocupação atual dos enlaces obtidas através do monitoramento. O controlador OpenFlow será essencial para realizar o re-roteamento dos fluxos de forma dinâmica e eficiente.

3.2. Funcionamento do EFM

Esta sub-seção detalha o funcionamento da arquitetura assim como a interação dos componentes descritos acima. Na Figura 2, apresentamos o Fluxograma que contém os passos necessários para detecção e re-roteamento de fluxos elefantes neste trabalho. Antes de descrevê-lo, é importante destacar que o EFM obtém as informações da topologia da rede consultando o controlador OpenFlow através de uma *API* padrão encontrada em todos os controladores. Com isso, o EFM monta a topologia armazenando-a na BD externa.

O EFM, mais especificamente o *Monitoring Module*, verifica constantemente a ocupação de cada enlace da topologia e, caso detecte um ponto de congestionamento, o *Elephant Detector Module* é notificado. Tal módulo então verifica se há fluxos elefantes

nesses pontos. Através do monitoramento utilizado neste trabalho, é possível mensurar quantos fluxos estão presentes no ponto de congestionamento e qual a taxa de transmissão dos mesmos. Com tais informações, o módulo ordena apenas os fluxos elefantes utilizando como parâmetro a taxa de transmissão dos mesmos. A ordenação dos fluxos elefantes é de suma importância para este trabalho pois uma das nossas contribuições é justamente analisar se a informação da taxa de transmissão dos fluxos influencia ao efetuar o re-roteamento. Por fim, o *Elephant Detector Module* seleciona o maior ou o menor fluxo elefante e notifica o *Actuator Module* para que uma rota seja encontrada para receber o fluxo. Caso não haja rota para este fluxo, o fluxo é removido da tabela e o passo é repetido para o próximo fluxo até que o re-roteamento seja realizado ou não haja mais fluxos na tabela de fluxos elefantes ordenados.

O *Actuator Module* é responsável por encontrar a melhor rota para cada fluxo elefante. O conceito de melhor rota neste trabalho se refere a uma rota em que a soma da ocupação dos seus enlaces fim-a-fim mais a colocação do fluxo elefante nesta rota não ultrapasse o limiar definido para o congestionamento em nenhum enlace. Na Figura 3, apresentamos de forma mais detalhada o pseudocódigo da função que busca pela melhor rota.

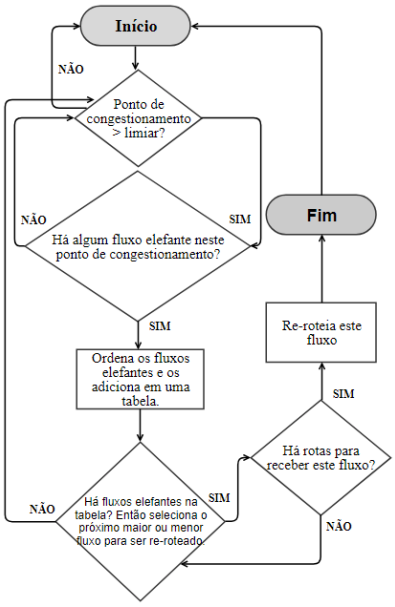

Figura 2. Fluxograma de tratamento de fluxos elefantes.

```
1:  procedure FINDROUTES(Flow f)
2:      while true do
3:          routes[] ← getRoutes(f)
4:          computeAvgUsage(routes)
5:          for i ← 1, routes[].length do
6:              if routes[i].Usage ≥ threshold then
7:                  delete(routes[i])
8:              end if
9:          end for
10:         routes[] ←sortRoutes(routes)
11:         for i ← 1, routes[].length do
12:             flagReroute = true
13:             for j ← 1, routes[i].links[].length do
14:                 if routes[i].links[j] + f.rate ≥ threshold then
15:                     flagReroute = false
16:                     break
17:                 end if
18:             end for
19:             if flagReroute then
20:                 reroute(f, routes[i])
21:                 return 0
22:             end if
23:         end for
24:         f ← chooseNewElephant()
25:     end while
26: end procedure
```

Figura 3. Pseudocódigo do módulo Actuator para encontrar a melhor rota.

A função *FindRoutes* possui como entrada o fluxo f que será re-roteado. Na linha 3, todas as rotas fim-a-fim são obtidas considerando a origem e destino do fluxo f e armazenadas no vetor $routes$. Posteriormente, a função $computeAvgUsage$ calcula a média fim-a-fim com base na ocupação de cada enlace considerando todos os enlaces da rota. O laço das linhas 5 a 9 verifica e remove as rotas cujas médias de ocupação excederam o limiar. Na linha 10, a função descrita ordena crescentemente o restante das rotas utilizando como parâmetro a taxa de ocupação média da rota. Os laços encadeados das linhas 11 a 23 realizam suas iterações de modo a verificar se a taxa de transmissão do fluxo a ser re-roteado somada a ocupação de cada enlace da rota não ultrapassa o

limiar definido. Caso esta soma exceda em um dos enlaces da rota, uma nova rota será verificada, caso haja rotas disponíveis. No fim da execução, se nenhuma rota que satisfaça as condições descritas for encontrada um novo fluxo elefante é escolhido. Caso uma rota seja encontrada, a função presente na linha 20 tem como objetivo passar as informações necessárias para que a nova rota para o fluxo elefante a ser re-roteado seja instanciada pelo controlador OpenFlow nos elementos de rede.

3.3. Implementação

Os três módulos descritos anteriormente foram implementados na linguagem de programação Java e projetados para executarem paralelamente utilizando *threads*. O monitoramento e a atualização da base de dados ocorrem paralelamente à detecção dos pontos de congestionamento na rede e a um possível re-roteamento dos fluxos elefantes.

A base de dados utilizada foi o Neo4J [Neo4J 2016], que implementa a linguagem de consulta CYPHER focada em bases relacionais para representação de grafos. O controlador OpenFlow selecionado foi o Floodlight [Floodlight 2008], implementado na linguagem Java e já consolidado no meio acadêmico como um dos controladores mais utilizados. As regras criadas nos elementos de rede para a realização do re-roteamento são comunicadas para o controlador por meio de uma *API REST* denominada *Static Entry Pusher* disponível pelo Floodlight.

O monitoramento realizado pelo EFM utiliza o protocolo sFlow [Phaal 2004]. Tal protocolo utiliza a metodologia de coleta das estatísticas por meio do método *Pushing* e efetua a amostragem de pacotes, ou seja 1 a cada N pacotes é analisado. Foram estudadas diversas ferramentas diferentes que utilizam o protocolo sFlow para a coleta de estatísticas, entretanto a que possuía maior afinidade com este projeto foi o sFlow Real Time (sFlow-RT) [Corporation 2004]. É através dela que coletamos e sumarizamos os dados que são relevantes para a nossa solução. Os agentes sFlow dos elementos de rede se comunicam com um coletor central do sFlow-RT que armazena em tempo real as informações que serão consultadas e utilizadas pelo EFM. Estes agentes possuem alguns parâmetros que devem ser configurados, por exemplo, a razão de amostragem que utilizamos 1 a cada 10 pacotes e o intervalo de envio das estatísticas para o coletor central que foi de 1 segundo.

4. Ambiente e Metodologia dos Testes

O ambiente de testes utilizado é composto por uma máquina física HP com as seguintes configurações: processador Intel i7 2600 da 2^a geração com 3.4Ghz, 16 GB de RAM e 1 TB de disco rígido. Nesta máquina física está sendo executado o EFM, o controlador Floodlight e a topologia do DC. Utilizamos o Mininet [Team 2007] para instanciar uma topologia *fat-tree* com 20 *switches* Open vSwitch (OVS) e 6 *hosts*. A topologia *fat-tree* utilizada possui três níveis contendo 20 *switches* (4 core, 8 de agregação e 8 de borda) e pode ser observada de modo minimalista na Figura 4. Implementamos o ECMP como um módulo do controlador Floodlight.

Os enlaces da topologia foram configurados para atuarem a uma taxa máxima de 10 Mb/s. Portanto, neste trabalho um fluxo é considerado elefante se sua taxa de transmissão exceder 10% da capacidade total do link ou seja, 1 Mb/s. Definimos o limiar de congestionamento em 70% de uso de um determinado enlace, ou seja, caso exceda 7

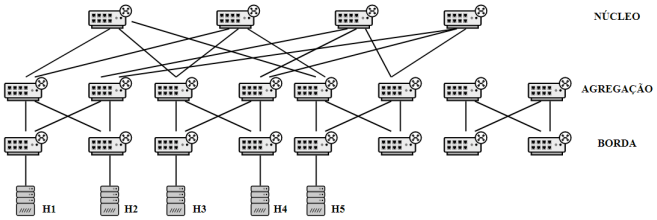

Figura 4. Topologia fat-tree com os *hosts*.

Mb/s de uso o EFM deverá atuar. A geração de tráfego foi realizadao através do software Iperf [NLANR/DAST 2007].

Como elencado em trabalhos como o Hedera [Al-Fares et al. 2010], as colisões que podem ser minimizadas em DCNs com topologias *fat-tree* são as colisões *upstream* e *downstream* (geradas pelo *hashing* do ECMP). Os cenários apresentados neste trabalho reproduzem estas colisões que ocorrem nos *switches* núcleo e, portanto, merecem um gerenciamento mais fino.

Nossos testes compararam duas abordagens: a primeira utilizou o ECMP puro para efetuar o roteamento dos fluxos. A segunda consiste no uso do EFM em conjunto com o ECMP. Neste último caso, o EFM foi exercitado de duas formas: re-roteando o maior fluxo elefante e re-roteando o menor fluxo elefante. Realizamos quatro testes contemplando colisões *upstream* e *downstream* e, para cada teste, fizemos quatro variações de banda para cada fluxo. Calculamos então a média dos valores obtidos que serão exibidos na próxima Seção 5. As métricas analisadas nestes testes foram: FCT, *throughput* e a análise preliminar sobre o consumo de energia.

5. Resultados

Esta seção apresenta e discorre sobre os resultados obtidos na realização dos testes. A Sub-Seção 5.1 apresenta os parâmetros utilizados na geração dos fluxos elefantes para os testes com colisões *upstream* assim como, os gráficos que ilustram o FCT e o *throughput*. Análogo a Sub-Seção 5.1 apresentamos os resultados obtidos para os testes com colisões *downstream* na Sub-Seção 5.2. Por fim, na Sub-Seção 5.3, dois testes preliminares (sendo um com colisões *upstream* e o outro com colisões *downstream*) contendo a análise do consumo de energia nos *hosts*.

5.1. Testes de Colisões Upstream

Este teste foi dividido em dois testes principais. A primeira bateria de testes foi realizada com três fluxos elefantes e então fizemos quatro variações nas bandas dos fluxos deste mesmo teste. Na segunda rodada de testes adicionamos um quarto fluxo com o intuito de gerar tráfego de fundo na rota que antes estava relativamente ociosa. Realizamos também quatro variações para a segunda rodada de testes.

Na Tabela 1, apresentamos um resumo das taxas de transmissão utilizadas em cada um dos fluxos presentes nos testes. As colunas F1 a F4 representam os fluxos e suas taxas de transmissão em Mb/s (megabits por segundo). Os fluxos são TCP e todos transmitem 20 Megabytes no total, utilizando uma janela de congestionamento fixa em

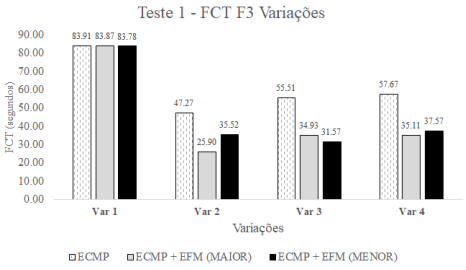

Figura 5. Variações do F3 no Teste 1.

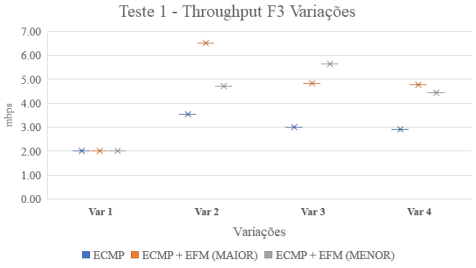

Figura 6. Variações do F3 no Teste 1.

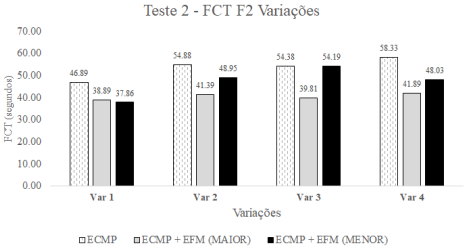

Figura 7. Variações do F2 no Teste 2.

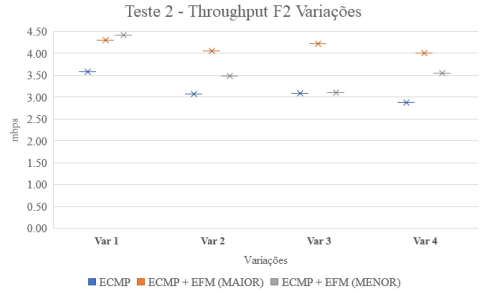

Figura 8. Variações do F2 no Teste 2.

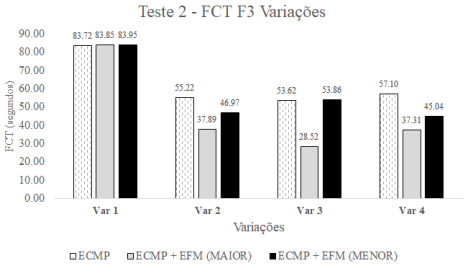

Figura 9. Variações do F3 no Teste 2.

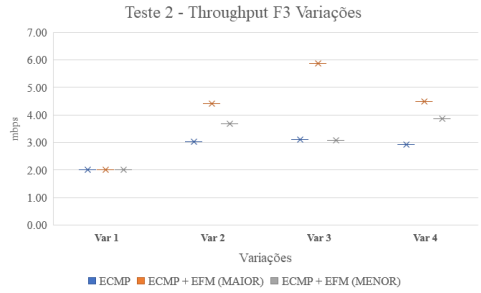

Figura 10. Variações do F3 no Teste 2.

64 KB. As marcações com X na Tabela ilustram que não houve tal fluxo naquele teste. A única diferença entre as variações destes testes é a taxa de transmissão dos fluxos.

Tabela 1. Parâmetros Testes 1 e 2 em Mb/s.

		F1	F2	F3	F4
Teste 1	Variação 1	4	4	8	X
	Variação 2	4	2	8	X
	Variação 3	3	5	8	X
	Variação 4	8	8	8	X
Teste 2	Variação 1	4	4	8	2
	Variação 2	4	8	6	2
	Variação 3	8	8	6	2
	Variação 4	6	4	5	2

Tabela 2. Parâmetros Testes 3 e 4 em Mb/s.

		F1	F2	F3	F4
Teste 3	Variação 1	4	4	8	X
	Variação 2	6	6	10	X
	Variação 3	8	8	5	X
	Variação 4	9	4	7	X
Teste 4	Variação 1	4	4	8	2
	Variação 2	8	8	2	3
	Variação 3	8	4	4	2
	Variação 4	6	8	3	2

Nas Figs. 5 e 6 apresentamos os resultados das diferentes variações para o FCT e *throughput* do Fluxo 3 (F3). Observamos que na Variação 2 obtivemos um melhor FCT re-roteando o maior fluxo elefante e na Variação 3 houve uma melhora no FCT atuando no menor fluxo elefante. Nas Figs. 7 a 10 os resultados dos Fluxos 2 e 3 (F2 e F3) para o Teste 2 são apresentados. Nestes gráficos houve uma predominância do melhor resultado ser obtido re-roteando o maior fluxo elefante, com exceção da Variação 1 que para o F2 foi um pouco melhor re-rotear o menor fluxo elefante e para o F3 o resultado permaneceu

o mesmo para ambas as abordagens do EFM.

O *throughput* é inversamente proporcional ao FCT, ou seja, se o FCT diminuiu consequentemente o *throughput* aumentou. A análise dos resultados é semelhante a dos gráficos que apresentamos o FCT. De modo geral, na maioria dos resultados superamos o ECMP, ou no pior dos casos igualamos os nossos resultados ao ECMP.

Sumarizando os resultados obtidos nos testes com colisões *upstream* realizados nesta Sub-Seção, observamos que predominantemente a melhor opção de re-roteamento foi atuar no maior fluxo elefante. Entretanto, houveram casos em que atuar no menor fluxo elefante resultou em melhores resultados tanto no FCT quanto no *throughput*. No Teste 1, a média de ganho no FCT re-roteando o maior fluxo elefante foi de 36.53%, enquanto que re-roteando o menor fluxo elefante obtivemos um ganho médio de 32.23%. No Teste 2, a média de ganho re-roteando o maior fluxo elefante foi de 36.78%, enquanto que re-roteando o menor fluxo elefante obtivemos um ganho médio de 16.71%. Estes valores consideraram todos os fluxos presentes no Teste 1 e 2.

5.2. Testes de Colisões Downstream

A metodologia utilizada nos testes desta Sub-Seção são iguais as descritas na Sub-Seção anterior. Na Tabela 2, apresentamos um resumo das taxas de transmissão utilizadas em cada um dos fluxos presentes nos testes. As colunas F1 a F4 representam os fluxos e suas taxas em Mb/s. Os fluxos são TCP e todos transmitem 20 Megabytes no total, utilizando uma janela de congestionamento fixa em 64 KB. As marcações com X na Tabela ilustram que não houve tal fluxo naquele teste.

Os resultados para os Testes 3 e 4 são apresentados nas Figuras 11 a 16. Observamos nas Figs. 11 e 12 que o FCT para o F1 no Teste 3 foi menor quando re-roteamos o menor fluxo elefante. No caso do F3, mostrado nas Figs. 13 e 14, temos que na maioria dos casos é melhor re-rotear o menor fluxo elefante, destaque para a Variação 2 onde o *throughput* dobrou quando comparado ao ECMP. Nas Figs. 15 e 16 relacionadas ao F2 do Teste 4, observamos que nas Variações 3 e 4 os resultados foram semelhantes, enquanto que nas Variações 1 e 2 foi melhor re-rotear o maior fluxo elefante. Como dito na Sub-Seção anterior, o *throughput* é inversamente proporcional ao FCT. Portanto, os ganhos obtidos no *throughput* refletem na diminuição do FCT dos fluxos.

Sumarizando os resultados obtidos nos Testes 3 e 4, podemos concluir que não houve tanta diferença entre re-rotear o maior ou o menor fluxo elefante já que, ambas as opções apresentaram resultados semelhantes. No Teste 3, a média de ganho re-roteando o maior fluxo elefante foi de 52.44%, enquanto que re-roteando o menor fluxo elefante obtivemos um ganho médio de 54.43%. No Teste 4, a média de ganho re-roteando o maior fluxo elefante foi de 29.1%, enquanto que re-roteando o menor fluxo elefante obtivemos um ganho médio de 24.13%.

A principio gostaríamos de concluir os resultados dos quatro testes expostos neste artigo apresentando qual abordagem é mais benéfica. Entretanto analisando os resultados vimos que os ganhos entre as diferentes abordagens são semelhantes. Em contra partida, constatamos que na maioria dos cenários apresentados o valor médio de retransmissões ocasionados ao re-rotear o menor fluxo-elefante é inferior do que ao re-rotear o maior fluxo elefante, como podemos observar nas Figuras 17 e 18 que representam o valor médio de retransmissões dos Testes 2 (*upstream*) e 4 (*downstream* respectivamente. Nos

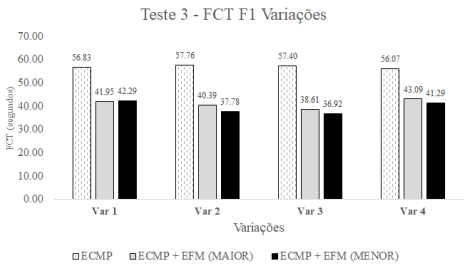

Figura 11. Variações do F1 no Teste 3.

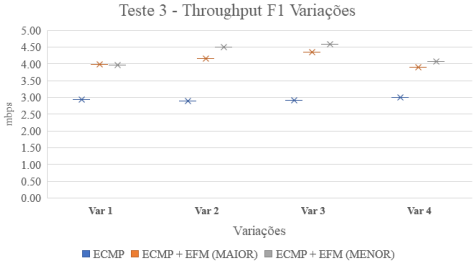

Figura 12. Variações do F1 no Teste 3.

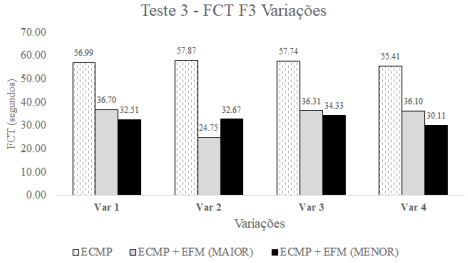

Figura 13. Variações do F3 no Teste 3.

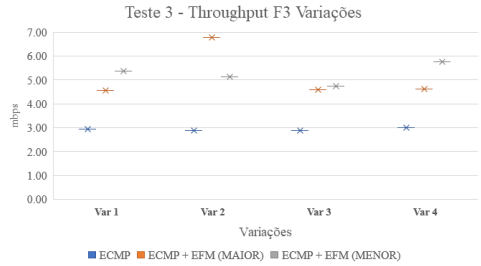

Figura 14. Variações do F3 no Teste 3.

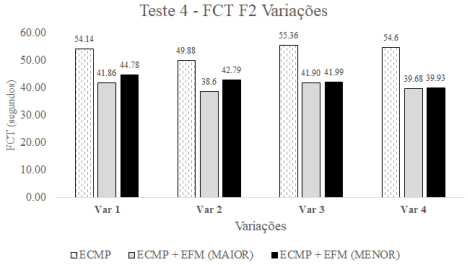

Figura 15. Variações do F2 no Teste 4.

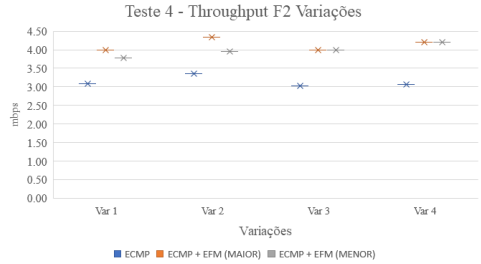

Figura 16. Variações do F2 no Teste 4.

casos onde o oposto é observado a diferença de tamanho entre o maior e o menor fluxo elefante é pequena, por este motivo ocorrem casos onde o número de retransmissões re-roteando o maior fluxo elefante é inferior ao número de retransmissões obtido ao re-rotear o menor fluxo elefante.

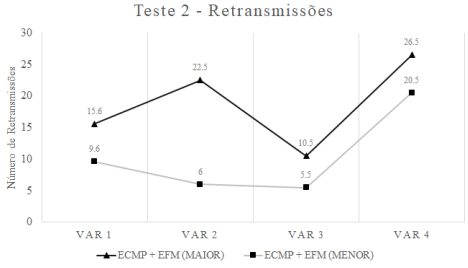

Figura 17. Teste 2 colisões *upstream* valor médio de retransmissões.

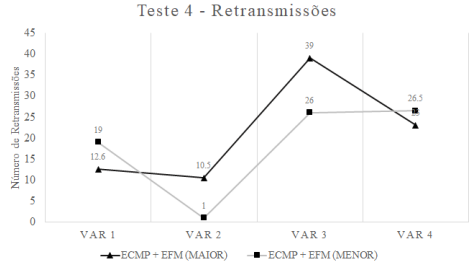

Figura 18. Teste 4 colisões *downstream* valor médio de retransmissões.

5.3. Testes Preliminares sobre o Consumo de Energia

Realizamos dois testes preliminares para medirmos o consumo de energia nos *hosts*, comparando o ECMP puro com o EFM + ECMP. O ambiente de testes foi semelhante ao utilizado nos testes descritos anteriormente, a única alteração foi trocar os *hosts* que antes eram máquinas virtuais por máquinas físicas, pois a ferramenta utilizada para a medição do consumo de energia não apresentou bons resultados ao medir os valores das máquinas virtuais instanciadas pelo Mininet. Para coletarmos as informações do consumo de energia dos *hosts*, utilizamos a ferramenta RAPL [Desrochers et al. 2016]. Tal ferramenta traz diversas informações referentes ao consumo de energia, entretanto neste trabalho apenas duas foram estudadas o *package* e os *cores*. O *package* é o consumo de energia da placa mãe como um todo e os *cores* o consumo de energia dos núcleos do processador dos *hosts*, denominados nos gráficos como MB (*motherboard*) e CPU respectivamente.

Os dois testes realizados contemplam colisões *upstream* (Teste 1) e *downstream* (Teste 2). Foram criados três fluxos para cada um dos testes. Os parâmetros utilizados na criação dos fluxos podem ser vistos na Tabela 3. Optamos por coletar as informações do consumo de energia de apenas um dos *hosts* em cada teste. No Teste 1 medimos o consumo de energia do *host* 2 enquanto no Teste 2 obtivemos o consumo de energia do *host* 5. Cada teste foi executado 5 vezes para o ECMP, ECMP + EFM re-roteando o maior e o menor fluxo elefante e então calculamos a média do consumo de energia da MB e da CPU para cada alternativa de roteamento acima.

Tabela 3. Parâmetros dos Testes sobre o consumo de energia.

		Taxa de Transmissão	Bytes a Transferir	Host de Origem	Host de Destino
Teste 1	F1	4 Mb/s	100 MB	H1	H3
	F2	4 Mb/s	100 MB	H1	H3
	F3	8 Mb/s	100 MB	H2	H5
Teste 2	F1	4 Mb/s	100 MB	H5	H3
	F2	4 Mb/s	100 MB	H5	H3
	F3	8 Mb/s	100 MB	H1	H4

Os resultados obtidos para o Teste 1 e 2 estão expostos nas Figs. 19 a 22. No eixo y dos gráficos observamos o consumo de energia em *Watts*, enquanto que no eixo x visualizamos uma linha temporal que ilustra desde o início do tráfego no DC até o término do mesmo. Nos quatro gráficos podemos analisar que em boa parte do tempo o consumo de energia executando apenas o ECMP é próxima a ambas as soluções do EFM. Entretanto, nas Figs. 19 e 20 a partir do instante 241 o ECMP se mantém enquanto que o consumo com o EFM diminui durante um intervalo de tempo. Para as Figs. 21 e 22 a análise é a mesma a partir do instante 196. Isto acontece pois como os fluxos com o EFM terminam antes devido à diminuição em seus FCTs, o consumo de energia nos *hosts* também diminui assim que os fluxos terminam. Sendo assim, a diferença de tempo entre o término de um determinado fluxo com ECMP e com ECMP + EFM é também o tempo em que o consumo de energia da nossa solução será inferior ao consumo de energia utilizando o ECMP puro.

Concluímos que devido à diminuição do FCT através da atuação do EFM, pode-se também diminuir o consumo de energia nos servidores dos DCs. A Tabela 4 ilustra o consumo médio de energia (em *Watts*) dos dois testes apresentados. Observamos na tabela que re-rotear o menor fluxo elefante obteve melhores resultados em relação ao ECMP puro. Constatamos também que a economia de energia será sempre proporcional

Tabela 4. Consumo médio de energia dos testes em Watts.

	ECMP	EFM (Maior)	EFM (Menor)
Teste 1 - CPU	0.04718	0.04837	0.04062
Teste 1 - MB	2.86575	2.87505	2.85102
Teste 2 - CPU	0.08279	0.07823	0.07961
Teste 2 - MB	2.97591	2.95849	2.97840

ao ganho obtido com relação ao FCT, comparando o EFM + ECMP com apenas o ECMP puro. Se o ganho no FCT não for significativo o consumo de energia permanecerá o mesmo, ou seja, não haverá grandes mudanças. Entretanto, caso o FCT apresente ganhos interessantes, a tendência na diminuição do consumo de energia é vantajosa para o DC.

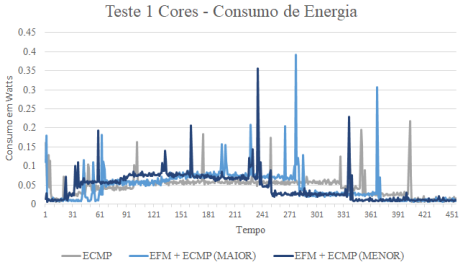

Figura 19. Teste 1 - Consumo da CPU.

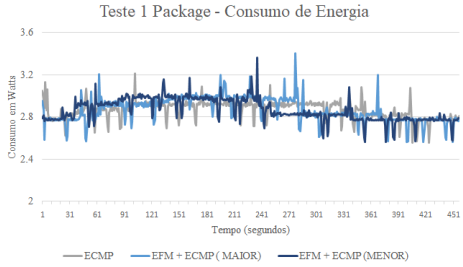

Figura 20. Teste 1 - Consumo da MB.

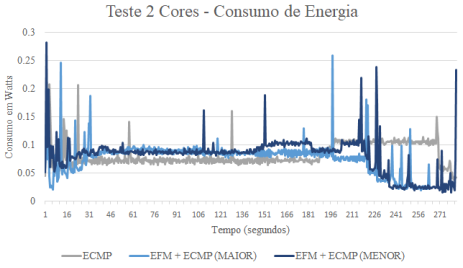

Figura 21. Teste 2 - Consumo da CPU.

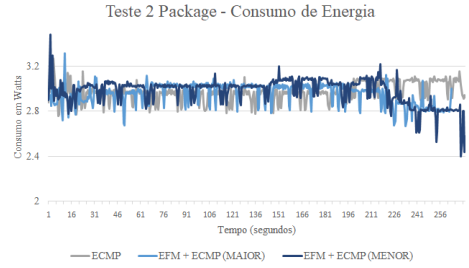

Figura 22. Teste 2 - Consumo da MB.

6. Conclusão

Neste trabalho, apresentamos o EFM cujo objetivo é otimizar o FCT e o *throughput* dos fluxos em DCNs. O EFM atua em conjunto com o ECMP e utiliza as informações da taxa de transmissão dos fluxos elefantes detectados e o estado atual da rede para decidir qual fluxo re-rotear (o maior ou menor) e qual a melhor rota para o mesmo.

Observamos que é viável conhecer a taxa de transmissão atual dos fluxos elefantes presentes na rede, de modo a maximizar o *throughput* e diminuir o FCT dos fluxos através do re-roteamento dos mesmos. Nossa arquitetura se mostrou eficiente quando comparada ao ECMP puro, utilizando ambas as opções de re-roteamento, atuando no maior ou no menor fluxo elefante. Resultados preliminares mostraram que o EFM pode trazer benefícios para a economia de energia nos servidores dos DCs.

Alguns pontos podem ser elencados como continuação deste trabalho. O primeiro deles é utilizar técnicas de aprendizado de máquina para auxiliar na escolha de qual abordagem do EFM utilizar de acordo com o tráfego atual na DCN (re-rotear o maior ou o menor fluxo elefante). Outra contribuição futura é realizar testes em ambientes reais para

validar melhor o EFM, testando em cenários com diferentes padrões de tráfego e uma carga maior de fluxos com equipamentos mais robustos.

Agradecimentos

Este trabalho foi fomentado pela CAPES e FAPESP cujo processo é 15/19766-9.

Referências

Al-Fares, M., Loukissas, A., and Vahdat, A. (2008). A scalable, commodity data center network architecture. *SIGCOMM Comput. Commun. Rev.*, 38(4):63–74.

Al-Fares, M., Radhakrishnan, S., Raghavan, B., Huang, N., and Vahdat, A. (2010). Hedera: Dynamic flow scheduling for data center networks. In *NSDI*, volume 10, pages 19–19.

Benson, T., Akella, A., and Maltz, D. A. (2010). Network traffic characteristics of data centers in the wild. In *Proceedings of the 10th ACM SIGCOMM Conference on Internet Measurement*, IMC '10, pages 267–280, New York, NY, USA. ACM.

Corporation, I. (2004). sflow-rt page.

Curtis, A. R., Kim, W., and Yalagandula, P. (2011). Mahout: Low-overhead datacenter traffic management using end-host-based elephant detection. In *2011 Proceedings IEEE INFOCOM*, pages 1629–1637.

Desrochers, S., Paradis, C., and Weaver, V. M. (2016). A validation of dram rapl power measurements. In *Proceedings of the Second International Symposium on Memory Systems*, MEMSYS '16, pages 455–470, New York, NY, USA. ACM.

Floodlight, P. (2008). Documentation of Floodlight.

Kanagevlu, R. and Aung, K. M. M. (2015). Sdn controlled local re-routing to reduce congestion in cloud data center. In *2015 International Conference on Cloud Computing Research and Innovation (ICCCRI)*, pages 80–88.

Katta, N., Hira, M., Ghag, A., Kim, C., Keslassy, I., and Rexford, J. (2016). Clove: How i learned to stop worrying about the core and love the edge. In *Proceedings of the 15th ACM Workshop on Hot Topics in Networks*, HotNets '16, pages 155–161, New York, NY, USA. ACM.

Neo4J (2016). Documentation of neo4j.

NLANR/DAST (2007). Iperf - the TCP/UDP bandwidth measurement tool.

Phaal, P. (2004). sflow version 5 specification.

Silva, A. and Verdi., F. L. (2017). Empowering applications with rfc 6897 to manage elephant flows in datacenter networks. In *IEEE International Conference on Cloud Networking (Cloudnet)*, pages 1–6.

Team, M. (2007). Documentation of Mininet.

Xu, H. and Li, B. (2014). Tinyflow: Breaking elephants down into mice in data center networks. In *LANMAN*, pages 1–6. IEEE.

Zhao, J., Liu, J., Wang, H., and Xu, C. (2017). Multipath tcp for datacenters: From energy efficiency perspective. In *IEEE INFOCOM 2017 - IEEE Conference on Computer Communications*, pages 1–9.