André Luiz Beltrami Rocha

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

Sorocaba, SP

19 de Dezembro de 2017

André Luiz Beltrami Rocha

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

Dissertação de mestrado apresentada ao Programa de Pós-Graduação em Ciência da Computação (PPGCC-So) da Universidade Federal de São Carlos como parte dos requisitos exigidos para a obtenção do título de Mestre em Ciência da Computação. Linha de pesquisa: Engenharia de Software e Redes de Computadores.

Universidade Federal de São Carlos – UFSCar

Centro de Ciências em Gestão e Tecnologia – CCGT

Programa de Pós-Graduação em Ciência da Computação – PPGCC-So

Orientador: Prof. Dr. Fábio Luciano Verdi

Sorocaba, SP

19 de Dezembro de 2017

«Rocha», «André Luiz Beltrami»
EFM - Elephant Flow Manager: Uma Arquitetura para Detecção e Tratamento de Fluxos Elefantes em DCNs/ André Luiz Beltrami Rocha. – 2018.
78 f. : 30 cm.

Dissertação (Mestrado) – Universidade Federal de São Carlos – UFSCar
Centro de Ciências em Gestão e Tecnologia – CCGT
Programa de Pós-Graduação em Ciência da Computação – PPGCC-So.
Orientador: Prof. Dr. Fábio Luciano Verdi
Banca examinadora: Prof. Dr. Fábio Luciano Verdi, Prof. Dr. Rafael Pasquini, $Prof^a.Dr^a$. Yeda R. Venturini
Bibliografia

1. Fluxos Elefantes. 2. Data Centers. 3. Software Defined Networks. I. Prof. Dr. Fábio Luciano Verdi. II. Universidade Federal de São Carlos. III. EFM - Elephant Flow Manager: Uma Arquitetura para Detecção e Tratamento de Fluxos Elefantes em DCNs

UNIVERSIDADE FEDERAL DE SÃO CARLOS

Centro de Ciências em Gestão e Tecnologia
Programa de Pós-Graduação em Ciência da Computação

Folha de Aprovação

Assinaturas dos membros da comissão examinadora que avaliou e aprovou a Defesa de Dissertação de Mestrado do candidato André Luiz Beltrami Rocha, realizada em 19/12/2017:

Prof. Dr. Fabio Luciano Verdi
UFSCar

Prof. Dr. Rafael Pasquini
UFU

Profa. Dra. Yeda Regina Venturini
UFSCar

Certifico que a defesa realizou-se com a participação à distância do(s) membro(s) Rafael Pasquini e, depois das arguições e deliberações realizadas, o(s) participante(s) à distância está(ao) de acordo com o conteúdo do parecer da banca examinadora redigido neste relatório de defesa.

Prof. Dr. Fabio Luciano Verdi

Aos meus pais Diogenes e Marcia,
e toda a minha Família.

Agradecimentos

Agradeço,

a Deus por nunca me abandonar.

aos meus pais, Diogenes e Marcia, por todo o amor e cuidado que sempre demonstraram por mim.

a minha família, irmãos, por sempre estarem ao meu lado.

ao meu orientador, Professor Doutor Fábio, por tudo o que ele me proporcionou, inúmeras oportunidades e ensinamentos que serão levados por toda a minha vida.

a minha namorada, Thais, por me aguentar e me ajudar a passar por todas as dificuldades.

aos meus amigos, Rodrigo, Vinícius, Caio, Matheus, Vanessa, Marcelo, Eduardo e Jimi por participarem da minha vida neste período especial e contribuírem cada um de modo singular.

a todos os professores do PPGCCS da UFSCar-Sorocaba, pelas excelentes aulas e ensinamentos no decorrer do curso.

aos técnicos administrativos da UFSCar-Sorocaba, por sempre estarem dispostos e disponíveis a ajudar, em especial ao secretário Roberto.

à UFSCar pela infraestrutura oferecida, em especial ao laboratório de pesquisa LERIS/LASID.

à CAPES e FAPESP pelo auxílio financeiro.

a todos os demais envolvidos para que este trabalho fosse concluído, muito obrigado.

"Laços são apenas criados quando dificuldades são superadas em conjunto."(Hagomoro)

Resumo

Um dos principais problemas em Data Center Networks (DCNs) está relacionado aos fluxos elefantes. Estes fluxos, 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 trabalho apresenta o EFM (*Elephant Flow Manager*) como uma solução completa de monitoramento e re-roteamento de fluxos elefantes em DCNs observando o tamanho destes fluxos na rede. 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.

Palavras-chaves: Redes de Data Center, Fluxos Elefantes, Re-roteamento, Monitoramento, SDN, OpenFlow, Consumo de Energia.

Abstract

One of the major issues in Data Center Networks (DCNs) is related to elephant flows. These 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 work presents EFM (Elephant Flow Manager) as a complete solution for monitoring and re-routing elephant flows in DCNs by observing the size of these flows in the network. 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 flows is reduced.

Key-words: Data Center Networks, Elephant Flows, Re-routing, Monitoring, SDN, OpenFlow, Energy Consumption.

Lista de ilustrações

Lista de tabelas

Lista de abreviaturas e siglas

bps	bits por segundo
DC	*Data Center*
DCN	*Data Center Network*
EFM	*Elephant Flow Manager*
ECMP	*Equal Cost Multipathing*
FCT	*Flow Completion Time*
Mbps	Megabits por segundo
OVS	OpenV-switch
ToR	Top of Rack

Sumário

1 Introdução

As *Data Center Networks* (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; LOUKISSAS; VAHDAT, 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; AKELLA; MALTZ, 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; 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).

O monitoramento de redes é parte fundamental nos principais DCs. Saber o que está sendo trafegado na rede e o impacto disto nela é primordial para tomar decisões de modo a minimizar possíveis problemas. Visando trazer maior flexibilidade de gerenciamento para os administradores de redes surgiu o paradigma de redes definidas por software (*SDNs - Software Defined Networks*). Assim como nas redes tradicionais, nas SDNs o monitoramento dos elementos de rede é necessário para que haja a detecção de possíveis pontos de congestionamento na DCN. Entretanto, o monitoramento feito de forma isolada não é suficiente para minimizar os problemas que ocorrem nos DCs. Sendo assim, se faz necessário monitorar e atuar sobre as DCNs de modo a minimizar problemas típicos de DCs, como por exemplo, o congestionamento nos elementos de rede.

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; AUNG, 2015) ou subdividir os fluxos elefantes em fluxos camundongos, como pode ser visto em (XU; LI, 2014). Recentemente, soluções utilizando o MPTCP (*MultiPath TCP*), como por exemplo em (ZHAO et al., 2017; SILVA; 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.

Objetivo

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 pelo menos um ponto de congestionamento seja detectado, o EFM irá atuar nos fluxos elefantes de modo a minimizar os impactos negativos ocasionados pelos mesmos, minimizando o consequente descarte de pacotes pelos elementos de rede. Neste sentido, nossa abordagem apenas atuará caso um determinado limiar de ocupação do enlace seja alcançado. Quando tal limiar for atingido, nossa solução observará se há fluxos elefantes passando pelo(s) enlace(s) e, caso haja, identificará o maior e o menor fluxo para tomar as decisões de re-roteamento. Neste trabalho, o conceito de tamanho (maior e menor fluxo) está relacionado à taxa de transmissão em bits por segundo (bps) dos fluxos elefantes.

As principais contribuições que o EFM disponibiliza para a comunidade são:

- Propõe e implementa uma arquitetura completa para detecção e tratamento de fluxos elefantes nas DCNs;
- Promove a diminuição do FCT e o aumento do *throughput* dos fluxos elefantes quando comparado à estratégia de roteamento mais utilizada nas DCNs, o ECMP;
- Devido à diminuição do FCT de alguns fluxos, testes preliminares realizados demonstram a diminuição do consumo de energia na DCN.

Estrutura do Trabalho

Visando facilitar a leitura deste trabalho, apresentamos a seguir um breve resumo de cada um dos Capítulos desta Dissertação.

- Capítulo 2: Neste Capítulo, apresentamos a fundamentação teórica para este projeto de pesquisa. Discutimos os trabalhos relacionados assim como a contextualização dos pontos mais importantes que permeiam este trabalho.

- Capítulo 3: Neste Capítulo descrevemos o EFM, apresentando a arquitetura proposta assim como as funcionalidades de cada um dos componentes presentes na mesma. Uma descrição em alto nível do seu funcionamento e alguns detalhes de implementação também são discutidos neste Capítulo.

- Capítulo 4: Neste Capítulo, os testes realizados e seus respectivos resultados são discorridos. Informamos também detalhes do ambiente de testes e a metodologia utilizada para a obtenção dos resultados.

- Conclusão: Na Conclusão, relembramos brevemente o que foi apresentado neste projeto e discorremos sobre possíveis novas linhas de pesquisa que podem ser herdadas deste trabalho.

2 Fundamentação Teórica

O objetivo deste capítulo é facilitar a compreensão do trabalho, apresentando toda a fundamentação teórica necessária para um melhor entendimento do EFM. Na Seção 2.1, apresentamos os principais trabalhos relacionados a este e uma breve discussão sobre as principais diferenças deles com relação a nossa proposta. O ECMP será o *baseline* para este trabalho, portanto, iremos descrevê-lo com maiores detalhes na Seção 2.2. Na Seção 2.3 abordaremos resumidamente alguns conceitos relacionados às SDNs. Nas Seções 2.4 e 2.5 serão discutidas características importantes sobre o monitoramento de redes e sobre os fluxos elefantes, assuntos estes que permeiam o trabalho proposto.

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.

O DevoFlow (CURTIS et al., 2011) sugere algumas modificações no comportamento do protocolo OpenFlow (MCKEOWN et al., 2008). A ideia central da proposta é manter no controlador apenas os fluxos significativos. Tais fluxos são determinados de acordo com um levantamento estatístico feito pelo controlador. O controle dos fluxos menos significativos é devolvido aos elementos de rede. O principal problema dessa abordagem é que os elementos precisam ser robustos, pois eles são essenciais no funcionamento da arquitetura proposta. Como esse é um fator determinante, ele pode gerar um aumento no custo de manutenção da solução.

A proposta TinyFlow (XU; 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; KIM; YALAGANDULA, 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.

A proposta apresentada por *R. Kanagevlu* em (KANAGEVLU; AUNG, 2015) se assemelha à nossa, pois eles utilizam as estatísticas obtidas através do monitoramento como importante informação para decidir qual a melhor rota para o fluxo elefante detectado. Em resumo, eles monitoram o tráfego da rede e a detecção do fluxo elefante é dada através das estatísticas coletadas dos elementos de rede ToR (*Top of Rack*). Caso um fluxo ultrapasse o limiar que eles definiram, ele é considerado elefante. Para auxiliar na tomada de decisões eles mantêm *snapshots* com os valores da banda disponível por enlace na memória. Posteriormente, essas *snapshots* são utilizadas para decidir qual a melhor rota para realocar o fluxo elefante.

As principais diferenças entre os trabalhos descritos acima e o nosso são: (1) a análise das diferenças entre re-rotear o maior ou o menor fluxo elefante utilizando a sua taxa de transmissão para ordená-los crescentemente; (2) um fluxo será re-roteado caso haja pelo menos um ponto de congestionamento na DCN; (3) será efetuado o re-roteamento de apenas um fluxo elefante por iteração do EFM, escolhendo-se então o maior ou o menor elefante detectado. O mesmo fluxo não pode ser re-roteado mais de uma vez. (4) o fluxo só será re-roteado caso haja algum enlace que comporte o mesmo sem exceder o limiar definido.

2.2 Equal Cost Multipathing (ECMP)

O ECMP é uma estratégia madura de roteamento de fluxos comumente utilizada em DCs que utilizam as topologias *fat-tree*, CLOS, entre outras. Tais topologias possuem múltiplas rotas de mesmo custo, pré-requisito para que o ECMP possa ser bem utilizado.

O ECMP possui dois modos de operação. O mais comum é realizar o balanceamento de carga baseado nos fluxos, efetuando a *hash* dos 5 campos do cabeçalho TCP (IP origem, IP destino, porta origem, porta destino e protocolo) para definir a rota que o fluxo deverá seguir. Ou seja, todos os pacotes do mesmo fluxo TCP/IP serão encaminhados pela mesma interface. O segundo modo de operação é realizar o balanceamento baseado em pacote, utilizando um simples *round robin* por pacote de modo a encaminhar pacotes do mesmo fluxo por rotas diferentes.

Embora o protocolo ECMP seja simples de ser implementado, alguns problemas são encontrados ao utilizá-lo em DCNs. O principal deles é o congestionamento dos fluxos considerados elefantes. Isto acontece pois o ECMP não distingue fluxos camundongos de fluxos elefantes, ou seja, os fluxos elefantes podem concorrer entre si devido a problemas de colisão nas entradas da *hash*, prejudicando a vazão da rede como um todo. O resultado disso é um aumento no FCT e diminuição do *throughput* dos fluxos.

2.3 Redes Definidas por Software

Redes definidas por software (SDN) é um paradigma introduzido há menos de 10 anos, que revolucionou a área de redes como um todo. Sua principal motivação foi a flexibilidade e facilidade no gerenciamento que o software provê aos administradores de redes. Uma visão centralizada de toda a rede e a possibilidade de configurá-la dinamicamente pelo administrador da rede são as principais contribuições deste novo paradigma.

O protocolo OpenFlow (MCKEOWN et al., 2008) é um dos exemplos mais comuns de redes definidas por software. Uma de suas principais características é a separação do plano de controle e do plano de dados. O plano de controle é o responsável por permitir a comunicação de um controlador centralizado com os elementos presentes na rede. Sendo que todas as regras criadas pelo administrador para controlar a rede são trafegadas por este plano. O plano de dados é a estrutura por onde os pacotes de dados irão percorrer até chegar ao seu destino. Esta distinção entre os planos é fundamental para uma gerência transparente da rede. Funções que antes eram feitas por um hardware específico ou eram difíceis de serem realizadas, hoje podem ser substituídas por software. O encaminhamento de pacotes e o re-roteamento de forma dinâmica dos fluxos presentes na rede são exemplos destas funções.

Os softwares que gerenciam as redes OpenFlow são denominados controladores OpenFlow e, atualmente, existem diversos controladores em diferentes linguagens de programação. O crescimento da computação em nuvem impulsionou o desenvolvimento deste protocolo devido a sua visão centralizada da rede e configuração de regras nos elementos de rede através do controlador OpenFlow. Tal protocolo vem sendo amplamente utilizado em DCNs devido à sua flexibilidade e escalabilidade para o gerenciamento da rede como

um todo, seja ela física ou virtualizada.

2.4 Monitoramento dos Elementos de Rede

O monitoramento dos elementos de rede é parte fundamental na detecção dos fluxos elefantes assim como para o re-roteamento dos mesmos no nosso trabalho. Sendo assim, exploramos na literatura os aspectos mais importantes de cada protocolo, analisando seus pontos fortes e fracos. Foram pesquisados os principais protocolos de monitoramento, como por exemplo, o sFlow (PHAAL, 2004), o NetFlow/IPFIX (CLAISE; ED, 2004; CLAISE; ED, 2008), o SNMP (PRESUHN, 2002) e o próprio OpenFlow (MCKEOWN et al., 2008), que também implementa a funcionalidade de coletar estatísticas dos elementos de rede.

Na Tabela 1, baseada no estudo realizado em (RASLEY et al., 2014), comparamos três características importantes referentes aos cinco protocolos apresentados. Segue abaixo uma breve explicação de cada uma delas.

Tabela 1 – Principais características dos protocolos de monitoramento.

	Metodologia	**Port Counters**	**Packet Sampling**
IPFIX	Pull	Não	Sim
NetFlow	Pull	Não	Sim
OpenFlow	Pull	Sim	Não
sFlow	Push	Sim	Sim
SNMP	Pull	Sim	Não

Metodologia é a forma com que as métricas dos elementos são coletadas. Dois exemplos tipicamente utilizados são o *Pushing* e o *Polling*. Ambos possuem um coletor central responsável por obter e armazenar as informações dos elementos de rede. No método *Pushing*, as métricas são enviadas para o coletor central periodicamente a partir da rede. Por outro lado, no método *Polling* o coletor requisita periodicamente as métricas aos elementos de rede.

Port Counters são contadores em que os elementos de rede armazenam o número de *bytes* e de pacotes enviados e recebidos por porta de cada elemento monitorado. Tal informação pode ser facilmente recuperada pelos protocolos que suportam tal tecnologia.

Packet Sampling é o modo em que os elementos de rede encaminham os pacotes até o coletor central. A amostragem de pacotes (*Packet Sampling*) envia apenas o cabeçalho dos pacotes e utiliza a razão de 1/N, ou seja, um em cada N pacotes recebidos têm seus cabeçalhos encaminhados para o coletor. Então, é estimado o tráfego da

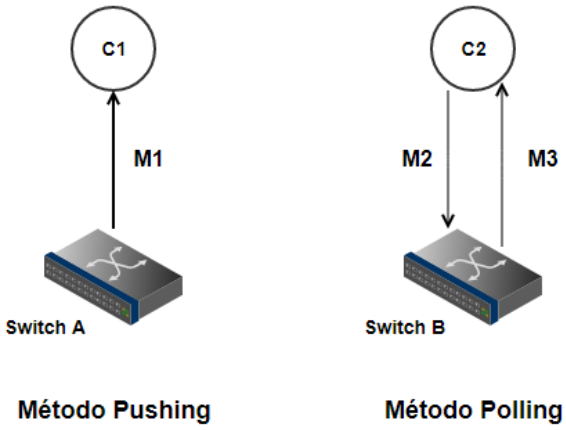
rede multiplicando o contador de pacotes e *bytes* de cada amostra por N (PHAAL; PANCHEN, 2004). Tais razões de 1/N variam de acordo com a capacidade de transmissão da rede. Na Tabela 2 extraída de (PETER, 2013), podemos visualizar quais são as razões utilizadas para as taxas de transmissão mais comuns.

Essencialmente, os protocolos NetFlow e IPFIX não fazem a coleta por amostragem, eles efetuam o espelhamento das portas dos elementos de rede e obtêm as informações pacote por pacote. Entretanto, eles podem ser configurados para espelharem as portas a cada N pacotes transmitidos.

Tabela 2 – Razão de amostragem.

Capacidade do Enlace	**Razão de Amostragem**
10 Mbit/s	1-em-10
100 Mbit/s	1-em-100
1 Gbit/s	1-em-1000
10 Gbit/s	1-em-10000
40 Gbit/s	1-em-40000
100 Gbit/s	1-em-100000

Figura 1 – Metodologias de monitoramento.



Na Figura 1, ilustramos os dois métodos de monitoramento apresentados. À esquerda, temos o método *Pushing*, onde C1 corresponde ao controlador central e M1 a mensagem que o elemento de rede A envia ao controlador periodicamente. À direita, temos o método *Polling*, onde C2 corresponde ao controlador central, M2 é a mensagem que requisita as estatísticas aos elementos de rede e M3 é a mensagem contendo as estatísticas dos fluxos.

Grande parte dos trabalhos relacionados à área de monitoramento possui o objetivo de otimizá-lo, ou seja, torná-lo cada vez mais preciso e eficiente. Em (RASLEY et

al., 2014) é proposto um método de monitoramento baseado no *timestamp* dos pacotes e no espelhamento das portas dos elementos de rede. Este trabalho também compara a solução proposta com as já conhecidas na literatura. Em estudos como o FlowSense (YU et al., 2013) e o Payless (CHOWDHURY et al., 2014) encontramos exemplos de monitoramento que utilizam o protocolo OpenFlow para obter informações, como por exemplo, a utilização de cada link. Nosso foco neste projeto não é otimizar o monitoramento em si, mas sim propor e implementar uma arquitetura que minimize os impactos dos fluxos elefantes em DCNs.

2.5 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; KIM; YALAGANDULA, 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; 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.

Assim como existem diversas definições de fluxos elefantes, há também diferentes formas de detectá-los. Basicamente, a detecção pode ser efetuada monitorando a rede ou diretamente nos *ends-host*, como aponta o estudo no artigo de Virtmone (BASHIR; AHMED, 2015). A maioria dos trabalhos definem um limiar para detecção de acordo com a definição de fluxos elefantes adotada. Na Tabela 3, temos um resumo das diferentes formas de detecção encontradas na literatura, as quais são explicadas abaixo.

- **Detecção através do monitoramento da rede:** pode ser subdividida no monitoramento da rede usando métodos de amostragem ou no monitoramento por fluxo nos *switches* ToR (*switches* ToR são tipicamente encontrados em DC e representam os *switches* topo de *rack*).

  - **Método de amostragem:** é o mesmo descrito na Sub-Seção 2.4, ou seja, são coletadas as métricas de todas as portas dos elementos de rede para posteriormente serem enviadas via *Pushing* para um coletor centralizado responsável por

armazenar e analisar as informações. No coletor, é possível fazer uma análise dos fluxos e detectar se eles são ou não elefantes com base na definição de fluxo elefante adotada. Trabalhos como (AFAQ; REHMAN; SONG, 2015; XU; LI, 2014; PHAAL; PANCHEN; MCKEE, 2001) utilizam métodos de amostragem;

  - **Monitoramento por fluxo nos *switches* ToR:** detecta os fluxos elefantes nos *switches* ToR, monitorando cada fluxo passante e periodicamente enviando estatísticas via *Polling* para um coletor centralizado quando requisitado. O fluxo é considerado elefante se seu consumo exceder um limiar definido. Tal limiar varia de acordo com o trabalho. Alguns exemplos de trabalhos que utilizam este método são: o Hedera (AL-FARES et al., 2010), o Helios (FARRINGTON et al., 2010) e o DevoFlow (CURTIS et al., 2011).

- **Detecção nos *end-hosts*:** pode ser subdividida em três métodos: monitoramento dos *buffers* dos sockets dos *end-hosts*, agregação de fluxos e detecção usando o Open vSwitch (OVS (TEAM, 2014)).

  - **Monitoramento do *buffer* do socket TCP:** coleta as informações direto dos *buffers* das interfaces dos *end-hosts*. Um exemplo de trabalho é o Mahout (CURTIS; KIM; YALAGANDULA, 2011) que define um limiar de 100 KB acumulados para classificar um fluxo como elefante;
  - **Agregação de fluxos:** introduzido no trabalho MicroTE (BENSON et al., 2011). Basicamente é feita uma modificação no núcleo de cada *host* da rede. Tal modificação permite coletar informações dos fluxos em micro intervalos, posteriormente elas são enviadas para um servidor que agrega os fluxos baseados em informações tais como, qual é o servidor e *rack* correspondente a cada fluxo;
  - **Monitoramento com *switches* virtuais:** introduzido no EMC2 (MANN; VISHNOI; BIDKAR, 2013), onde foi implementado um *web-service* capaz de obter as informações dos *switches* virtuais através de dois diferentes métodos de monitoramento: o sFlow e o NetFlow. A partir destas informações, em cada *hypervisor* é possível verificar os fluxos e identificar se são considerados elefantes.

A Tabela 3, baseada no trabalho (BASHIR; AHMED, 2015), ilustra as principais características dos diferentes métodos encontrados na literatura, assim como, apresenta os diferentes limiares utilizados. A primeira coluna corresponde aos principais trabalhos encontrados na literatura, a segunda coluna é o local que ocorre a detecção dos fluxos elefantes, a terceira coluna caracteriza qual definição de fluxo elefante foi adotada (Total de Bytes, Tempo de Vida ou Bytes/s) e a quarta coluna apresenta os limiares definidos em cada trabalho.

Tabela 3 – Comparação dos métodos para detecção de fluxos elefantes.

	Local de Detecção	**Definição Utilizada**	**Limiar para Detecção**
DevoFlow (CURTIS et al., 2011)	Na Rede	Total de Bytes	1-10 MB
Hedera (AL-FARES et al., 2010)	Na Rede	Bytes/s	100 Mb/s
Helios (FARRINGTON et al., 2010)	Na Rede	Bytes/s	15 Mb/s
M. Afaq em (AFAQ; REHMAN; SONG, 2015)	Na Rede	Bytes/s	500 KB/s
Mahout (CURTIS; KIM; YALAGANDULA, 2011)	No Host	Total de Bytes	100 KB
TinyFlow (XU; LI, 2014)	Na Rede	Tempo de Vida	0,5 ms

Por fim, é importante destacar, mesmo que resumidamente, as principais técnicas encontradas na literatura que possuem como objetivo minimizar o impacto dos fluxos elefantes na rede (CASADO, 2013). A primeira delas utiliza filas distintas para separar os fluxos camundongos dos fluxos elefantes. Outra técnica consiste em utilizar caminhos diferentes para separar os fluxos camundongos dos fluxos elefantes. Outros trabalhos se propõem a quebrar os fluxos elefantes em fluxos camundongos espalhando esses fluxos através dos múltiplos caminhos existentes na rede (SILVA; VERDI., 2017). Finalmente, algumas propostas atuam no re-roteamento dos fluxos elefantes com base nas informações do tráfego da rede. Neste trabalho, a definição de fluxo elefante utilizada será aquela relacionada à taxa de transmissão como visto em (AL-FARES et al., 2010), sendo que um fluxo será considerado elefante caso ele ultrapasse os 10% de ocupação do enlace. A técnica de atuação utilizada pelo EFM é a de re-rotear os fluxos elefantes com base nas informações de ocupação da rede e taxa de transmissão dos fluxos elefantes.

3 EFM - Elephant Flow Manager

Neste Capítulo iremos apresentar o EFM, uma arquitetura para a detecção e re-roteamento de fluxos elefantes em DCNs. Na Seção 3.1, descreveremos a arquitetura proposta, assim como as principais funcionalidades dos componentes presentes na mesma. Na Seção 3.2 será detalhado o funcionamento da arquitetura evidenciando as suas funções mais importantes. As informações relevantes sobre o monitoramento e detecção dos fluxos elefantes também serão descritas nesta seção. Por fim, na Seção 3.3 os detalhes de implementação e as decisões de projeto serão apresentadas.

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 consiste em definir e implementar uma arquitetura que 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 2, ilustramos os módulos que compõem nossa arquitetura, assim como as interações entre eles.

Os módulos propostos e suas responsabilidades são descritos a seguir:

- **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[1]. 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;

[1] A frequência da coleta é um parâmetro configurável e está fora do escopo deste trabalho discutir sobre o desempenho desta frequência.

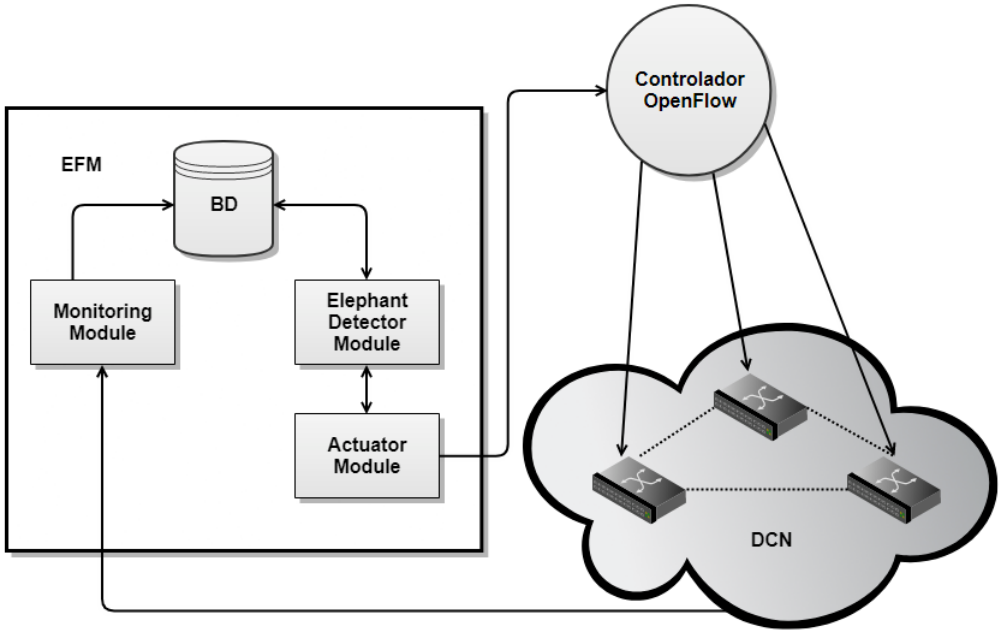


Figura 2 – Arquitetura proposta do EFM.

- **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 irá armazenar a topologia da DCN assim como, a taxa de ocupação atual dos enlaces obtidas através do monitoramento. O controlador OpenFlow será primordial para permitir o re-roteamento dos fluxos de forma dinâmica e eficiente.

3.2 Funcionamento da Arquitetura

Esta Seção irá descrever os pontos mais importantes apresentados em cada um dos módulos apresentados anteriormente, assim como pseudocódigos que ilustram em alto nível todo o processo de funcionamento do EFM. Na Sub-Seção 3.2.1 explanaremos detalhes de como foi realizado o monitoramento e mostraremos quais as informações de monitoramento são relevantes para o EFM detectar os pontos de congestionamentos e

os fluxos elefantes. Na Sub-Seção 3.2.2, o funcionamento do EFM é descrito utilizando pseudocódigos que sumarizam minimamente as funções do EFM.

3.2.1 Monitoramento e Detecção dos Fluxos Elefantes

Neste trabalho, o monitoramento é realizado periodicamente sendo que os principais objetivos são: (1) manter o estado atual do nível de ocupação de todos enlaces da topologia e; (2) obter informações detalhadas dos fluxos presentes em determinados enlaces.

Para manter o estado atual da rede, o EFM utiliza basicamente duas métricas que são coletadas dos elementos de rede: os *in octetos* e *out octetos*. Os *in octetos* e *out octetos* correspondem a quantidade de octetos que ingressam ou egressam, respectivamente, por uma determinada porta do elemento de rede. Esta é a primeira informação relevante coletada pelo EFM. Após a obtenção destas métricas, o *Monitoring Module* é responsável por calcular a porcentagem de ocupação dos enlaces e atualizar tanto a nossa base de dados (BD) quanto as estruturas de dados internas para consultas posteriores. Na BD, os valores de *in e out octetos* são calculados como a porcentagem de ocupação de um determinado enlace. Através da consulta periódica das estruturas de dados internas, é possível verificar se há enlaces cuja ocupação ultrapasse o limiar definido e, caso haja, estes enlaces são apontados como possíveis pontos de congestionamento. A partir deste momento, o EFM começa a atuar de modo a minimizar os congestionamentos.

O segundo objetivo do monitoramento é obter informações detalhadas sobre os fluxos presentes nos enlaces congestionados. As informações obtidas nesta segunda etapa do monitoramento são: os cinco campos do cabeçalho TCP/IP e a quantidade de *bytes* passantes nos enlaces por fluxo. Tais informações são de suma importância para a detecção dos fluxos elefantes e a ordenação dos mesmos utilizando como parâmetro a suas respectivas taxas de transmissão. As características detalhadas dos fluxos são obtidas a partir de um filtro encontrado no protocolo de monitoramento utilizado, este filtro retorna as informações mencionadas acima que são armazenadas e consultadas pelo EFM para que seja realizada a detecção dos fluxos elefantes presentes nos enlaces congestionados.

A ordenação dos fluxos elefantes, utilizando suas taxas de transmissão média, ocorre durante a detecção dos mesmos, levando em consideração a quantidade de *bytes* média por fluxo coletada nos enlaces congestionados. Na Sub-Seção 3.2.2 explanaremos melhor sobre como são realizados tais cálculos.

3.2.2 Descrição em Alto Nível do Funcionamento do EFM

Esta Sub-Seção detalhará o funcionamento da arquitetura assim como a interação dos componentes descritos acima. Na Figura 3, apresentamos o Fluxograma que contém os

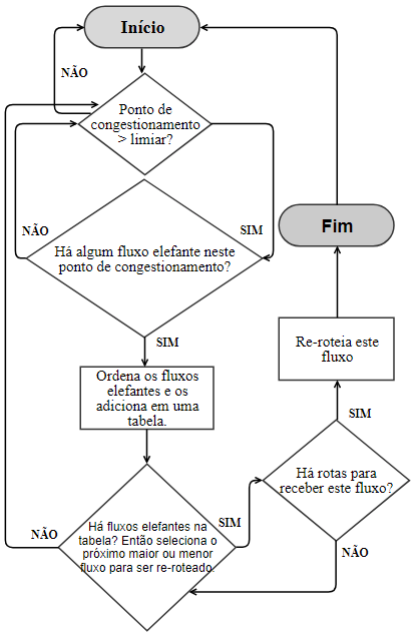


Figura 3 – Fluxograma de tratamento de fluxos elefantes.

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.

Na Figura 4, apresentamos um pseudocódigo sumarizado do funcionamento dos módulos *Monitoring e Elephant Detector*. A linha 3 tem como objetivo verificar a ocupação de cada um dos enlaces na rede. Caso algum enlace exceda o limiar definido, o seu identificador é armazenado no vetor *linkCongest*. A função *zoomLink* na linha 8, recebe como entrada o vetor anteriormente descrito e então calcula a taxa de transmissão média de cada um dos fluxos presentes nos enlaces que foram detectados como congestionados.

```
1: procedure DetectCongestion(MonitoringM)
2:     while true do
3:         for alllinks do
4:             if link.Usage ≥ threshold then
5:                 linkCongest[] ← link.ID
6:             end if
7:         end for
8:         flows[] ← zoomLink(linkCongest[])
9:         elephantsOrdered[] ← detectElephants(flows[])
10:        if smallest == TRUE then
11:            findRoutes(smallestElephant())
12:        else
13:            findRoutes(biggestElephant())
14:        end if
15:    end while
16: end procedure
```

Figura 4 – Pseudocódigo dos módulos Monitoring e Elephant Detector para detecção dos fluxos elefantes.

A taxa de transmissão média é calculada somando os valores obtidos individualmente da quantidade de *bytes* passantes naquele intervalo de tempo em cada enlace congestionado. Caso haja repetição de fluxos (ou seja, os 5 campos do cabeçalho TCP/IP são iguais) os valores são somados e então é obtida a média. Por exemplo, suponha que temos dois enlaces congestionados, o Enlace 1 e o Enlace 2. No Enlace 1 foram encontrados os fluxos A e B enquanto no Enlace 2 foram detectados os fluxos B e C. Como o fluxo B apareceu em dois enlaces distintos nossa solução calcula a média dos dois valores para obter a taxa de transmissão média do fluxo B.

Após calculada a taxa de transmissão dos fluxos presentes nos enlaces congestionados a função *detectElephants* os recebe como entrada e verifica se a taxa de transmissão dos fluxos ultrapassa o limiar definido. Caso ultrapasse, o fluxo é considerado elefante e é adicionado ao vetor *elephantOrdered*, que é ordenado crescentemente usando como parâmetro a taxa de transmissão dos fluxos. Por fim, a escolha de qual fluxo será re-roteado é definido de modo fixo, ou seja a solução irá re-rotear o maior ou o menor fluxo elefante a cada iteração. Após um fluxo elefante ser re-roteado ou o EFM não encontrar rota para o determinado fluxo ele é marcado como um fluxo elefante já computado e não estará nas próximas iterações.

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 5, apresentamos de forma mais detalhada o pseudocódigo da função que busca pela melhor

```
1: procedure FindRoutes(Flowf)
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 5 – Pseudocódigo do módulo Actuator para encontrar a melhor rota.

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* irá calcular 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 ordenará crescentemente o restante das rotas utilizando como parâmetro a taxa de ocupação média da rota. Os laços encadeados das linhas 11 a 16 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 deverá ser 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 17 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

3.3.1 Decisões de Projeto

Os três módulos descritos anteriormente foram implementados na linguagem de programação Java e projetados para executarem paralelamente utilizando *threads* possuindo cerca de 1700 linhas de código fonte. 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, como mencionado na Seção 2.4, 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 e o intervalo de envio das estatísticas para o coletor central. Neste trabalho a razão de amostragem utilizada foi de 1 a cada 10 pacotes, e o intervalo de envio foi de 1 segundo.

3.3.2 Detalhes de Implementação

Nesta Sub-seção, detalharemos a implementação das principais funções do EFM. Inicialmente, apresentamos as funções de coleta e armazenamento das informações tanto internamente (em memória) quanto na BD. A partir da Fig. 4, abordaremos as funções $zoomLink(linkCongest[])$ e $detectElephants(flows[])$ das linhas 8 e 9, respectivamente. A partir da Fig. 5 detalharemos as funções $getRoutes(f)$, $computeAvgUsage(routes)$ e

$reroute(f, routes[i])$ das linhas 3, 4 e 17, respectivamente. Ambas as figuras estão presentes na Sub-Seção 3.2.2.

Como descrito na Sub-Seção 3.2.1, as informações obtidas do monitoramento são os *in octetos* e os *out octetos*. Para armazená-los internamente, utilizamos uma *hashtable* que possui como chave o identificador da porta do elemento de rede e como valores os *in e out octetos*, o *datapath* ID do *switch* e a porta OpenFlow do *switch*. Na BD, calculamos para cada enlace a porcentagem de ocupação do mesmo e armazenamos tanto a ocupação baseada nos *in octetos* quanto nos *out octetos*. Para calcular a taxa de ocupação, utilizamos a seguinte fórmula:

$$taxaOcupacao = (inOctets * 8)/capacidadeEnlace * 100$$

.

onde o valor fixo 8 representa a transformação dos octetos em bits e a variável *capacidadeEnlace* representa a capacidade do enlace e deverá estar em bps (bits por segundo).

A função $zoomLink(linkCongest[])$ recebe como parâmetro um vetor contendo os enlaces que a sua taxa de ocupação ultrapassou o limiar. Esta função realiza uma consulta mais detalhada nestes enlaces, obtendo todos os fluxos presentes nos enlaces congestionados. As informações recuperadas dos fluxos são: o IP de origem, o IP de destino, a porta de origem, a porta destino, o protocolo e a quantidade de *bytes* passantes naquele enlace por fluxo e é uma funcionalidade do sFlow, mais precisamente do sFlow-RT. Esta função também é responsável por agregar os fluxos repetidos nos diferentes enlaces congestionados, como comentado na Sub-Seção 3.2.2. Armazenamos todas estas informações dos fluxos em um vetor denominado $flows$ que será parâmetro para a próxima função, a $detectElephants(flows[])$.

A quantidade de *bytes* passantes num determinado enlace por fluxo é também a taxa de transmissão do fluxo naquele enlace. A função $detectElephants(flows[])$ irá percorrer todos os fluxos do vetor $flows$ de modo a verificar a taxa de transmissão dos mesmos. Caso ela seja superior a 10% da capacidade do enlace, o fluxo é considerado elefante. Os fluxos detectados como elefantes são ordenados crescentemente no vetor $elephantsOrdered[]$, que possui como valores os mesmos do vetor $flows$. A partir deste momento o EFM pode escolher entre o maior ou o menor fluxo elefante detectado para encontrar uma nova rota para ele.

A função $getRoutes(f)$ recebe um fluxo como parâmetro (o maior ou o menor do vetor $elephantsOrdered[]$) e então utiliza a topologia instanciada na BD na forma de grafo para buscar por todas as rotas com base no IP origem e IP destino do fluxo f. Todas as informações das rotas possíveis são armazenadas no vetor $routes[]$, contendo todos os enlaces e *switches*. As informações referentes aos *switches* são o *datapath ID* e

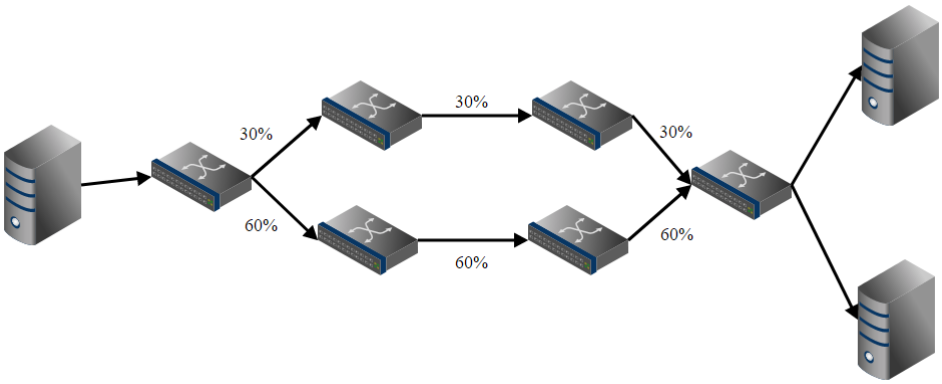


Figura 6 – Exemplo de cálculo de melhor rota.

o IP. As informações dos enlaces são os *switches* de origem e destino, portas de origem e destino (portas dos *switches*) e a ocupação de entrada e saída do enlace (*in* ou *out octetos*). Com tais informações, é possível calcular a porcentagem de ocupação média de cada rota, realizada pela função $computeAvgUsage(routes)$. Se a taxa de ocupação média de um determinado enlace é superior ao limiar definido, ela é removida do vetor *routes* por não ser uma rota viável para o re-roteamento do fluxo.

Caso haja pelo menos uma rota com a ocupação média inferior ao limiar, é verificado qual seria a melhor rota para o fluxo a ser re-roteado[2]. O EFM percorre os enlaces das rotas restantes[3], somando a taxa de transmissão do fluxo a ser re-roteado com a ocupação de cada enlace. Se esta soma não ultrapassar o limiar no decorrer de toda a rota, ela é escolhida como a melhor rota e então o re-roteamento pode ser realizado. Na Fig. 6 apresentamos um exemplo simples de como o cálculo de melhor rota é realizado. Suponha que o fluxo a ser re-roteado esteja com uma taxa de ocupação média na rede equivalente a 25%, a rota superior está com ocupação média de 30% enquanto que a rota inferior possui 60% de ocupação. Se nosso limiar fosse definido em 60%, a rota inferior seria excluída e restaria apenas a rota superior. Sendo assim, para cada um dos enlaces da rota superior é somado a sua ocupação média de 30% com a ocupação média do fluxo a ser re-roteado, que é de 25%, totalizando nos três enlaces 55% de ocupação. Se esta soma excedesse 60% (valor definido como limiar) em algum enlace, o fluxo não seria re-roteado. Se não houver pelo menos uma rota que comporte o fluxo sem ultrapassar o limiar, um novo fluxo elefante é escolhido.

A função $reroute(f, routes[i])$ têm como parâmetros as informações do fluxo a ser re-roteado e a rota na qual ele será re-roteado. A função definirá as regras OpenFlow para cada um dos *switches* pertencentes a nova rota do fluxo. As informações utilizadas para a criação das rotas via API REST do controlador OpenFlow são: *datapath ID* do *switch*,

[2] O conceito de melhor rota foi definido na Sub-Seção 3.2.2.
[3] As rotas estão ordenadas pela ocupação média de seus enlaces.

portas OpenFlow de origem e destino[4], IPs de origem e destino, portas TCP de origem e destino. Após enviadas as regras para o controlador OpenFlow, elas são instanciadas nos *switches* dinamicamente sem interromper a transmissão dos dados e uma nova rodada do EFM se inicia.

[4] As portas OpenFlow são utilizadas exclusivamente pelo protocolo OpenFlow e estão contidas nas informações do fluxo f.

4 Testes e Resultados

Neste Capítulo iremos detalhar os testes realizados assim como discutir os resultados obtidos a partir da comparação do ECMP com o EFM + ECMP re-roteando o maior e o menor fluxo elefante. Na Seção 4.1 descreveremos o ambiente de testes utilizado. Na Seção 4.2 a metodologia proposta para a coleta dos resultados nos diferentes testes realizados é descrita. Os resultados obtidos foram subdivididos em 4 Sub-Seções: A Subseção 4.3.1 apresenta os resultados obtidos simulando colisões *upstream*. A Subseção 4.3.2 apresenta os resultados obtidos simulando colisões *downstream*. A Subseção 4.3.3 apresenta a análise do tempo de execução das principais funcionalidades do EFM. A Subseção 4.3.4 contém os resultados preliminares que demonstram a diminuição no consumo de energia no DC após o re-roteamento do maior fluxo elefante ou do menor fluxo elefante.

4.1 Ambiente de 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. No Mininet, instanciamos uma topologia *fat-tree* com 20 *switches* OVS e 6 hosts. A topologia *fat-tree* utilizada possui três níveis contendo 20 *switches* (4 core, 8 de agregação e 8 de borda). A Figura 7 apresenta a topologia dos testes realizados. Alguns links foram omitidos para uma melhor visualização da imagem. Implementamos o ECMP como um módulo do controlador Floodlight.

Validamos nossa proposta em um ambiente virtualizado de forma emulada. Para isso, utilizamos o emulador de redes Mininet (TEAM, 2007) na versão 2.2.3. Os elementos de rede foram virtualizados utilizando o Open vSwitch (OVS) na versão 2.4. Os OVSs foram habilitados para suportar o agente sFlow para que ocorra a coleta das estatísticas nos elementos de rede.

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

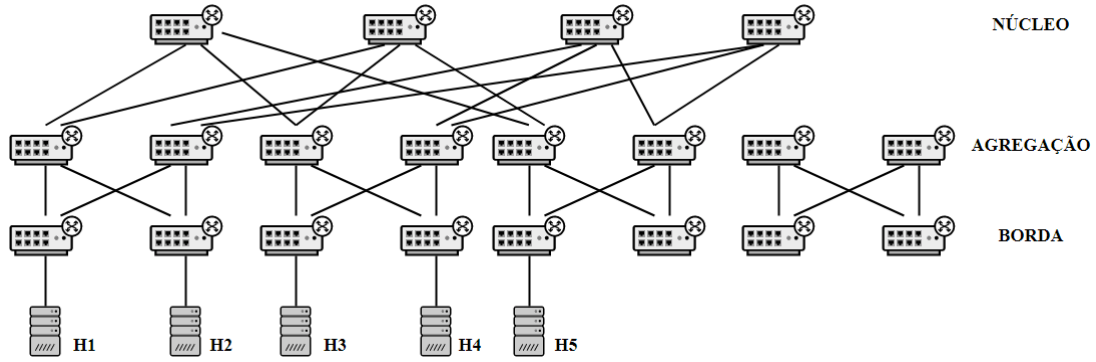


Figura 7 – Topologia fat-tree.

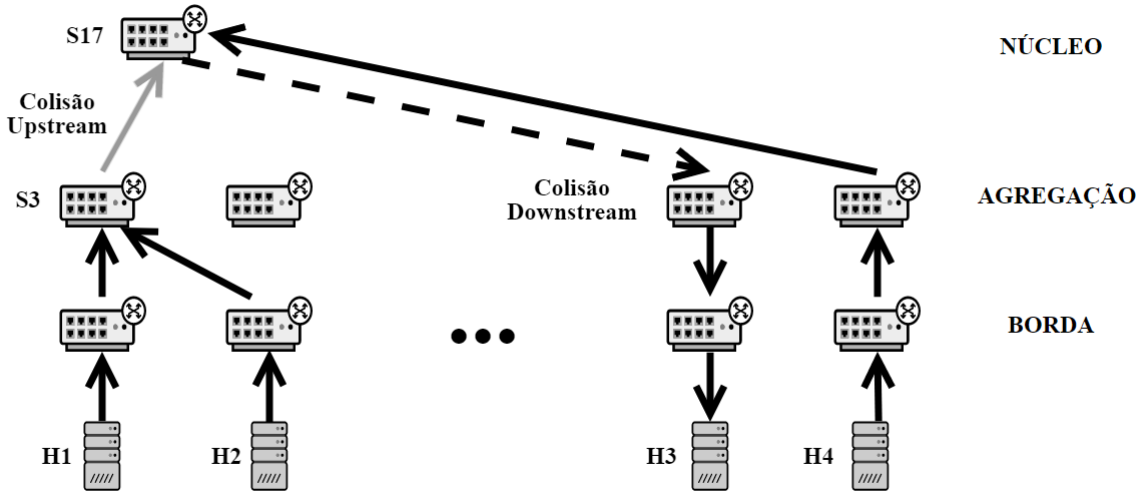


Figura 8 – Colisões upstream e downstream.

4.2 Metodologia dos Testes

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. Na Figura 8 ilustramos tais colisões. A colisão *upstream* ilustrada ocorre no S3 com a agregação dos fluxos que possuem como origem o H1 e H2 e destino o H3. A colisão *downstream* pode ser observada na linha tracejada e ocorre no S17 com os fluxos da linha cinza (que possuem como origem o H1 e H2 e destino o H3) e preta (que possui como origem o H4 e destino o H3) sendo encaminhados para a mesma interface de saída.

O tráfego foi gerado de modo controlado através do software Iperf (NLANR/DAST, 2007), com o objetivo de representar os fluxos elefantes na rede. Nos nossos testes, não

contemplamos os fluxos camundongos, pois nosso foco é analisar o impacto de utilizar a taxa de transmissão dos fluxos elefantes visando melhorar o re-roteamento.

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 onde cada variação foi executada cinco vezes. Calculamos então a média dos valores obtidos que serão exibidos na próxima Seção 4.3. As métricas analisadas nestes testes foram: FCT, *throughput*, número de retransmissões, tempo de execução das principais funções do EFM e a análise preliminar sobre o consumo de energia comparando o uso do ECMP com o uso do EFM juntamente com o ECMP.

4.3 Resultados

Esta Seção irá apresentar e discorrer sobre os resultados obtidos na realização dos testes. A Sub-Seção 4.3.1 apresenta os parâmetros utilizados na geração dos fluxos elefantes para os testes com colisões *upstream* assim como, os gráficos que ilustram o FCT, o *throughput* e o número de retransmissões. Análogo a Sub-Seção 4.3.1 apresentamos os resultados obtidos para os testes com colisões *downstream* na Sub-Seção 4.3.2. A Sub-Seção 4.3.3 analisa o tempo de execução das principais funções do EFM. Por fim, na Sub-Seção 4.3.4, dois testes preliminares (sendo um com colisões *upstream* e o outro com colisões *downstream*) contendo a análise do consumo de energia nos *hosts*.

4.3.1 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 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 4, 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 Mbps (megabits por segundo). 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. A única diferença entre as variações destes testes é a taxa de transmissão dos fluxos. Na Tabela 5, ilustramos a origem e o destino de cada um dos fluxos criados para cada teste realizado. O posicionamento dos *hosts* na topologia utilizada pode ser visto na Fig. 7.

Tabela 4 – Parâmetros em Mbps para os Testes 1 e 2 - Colisões upstream.

		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 5 – Origem e Destino dos fluxos para os Testes 1 e 2 - Colisões upstream.

		Origem	Destino
Teste 1	F1	H1	H3
	F2	H1	H3
	F3	H2	H5
Teste 2	F1	H1	H3
	F2	H1	H3
	F3	H2	H5
	F4	H2	H4

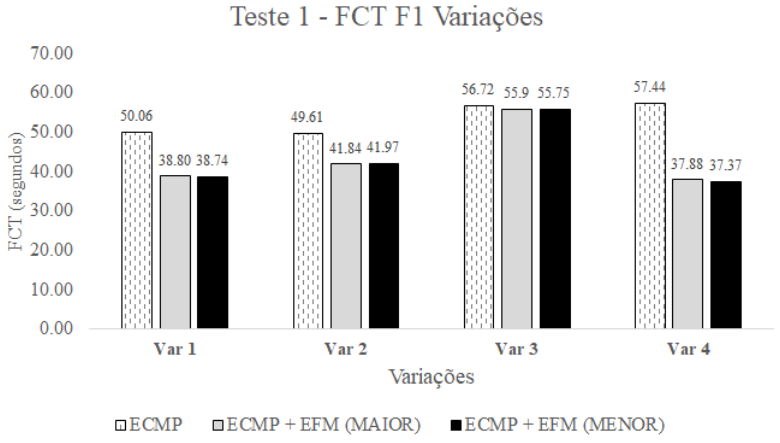


Figura 9 – FCT: Variações do F1 no Teste 1.

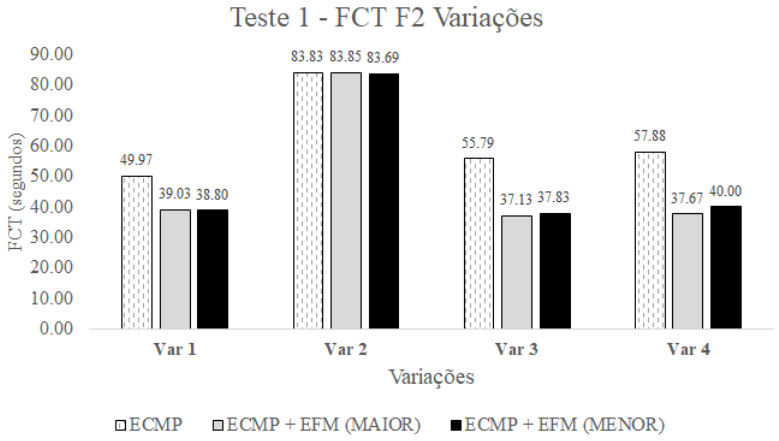


Figura 10 – FCT: Variações do F2 no Teste 1.

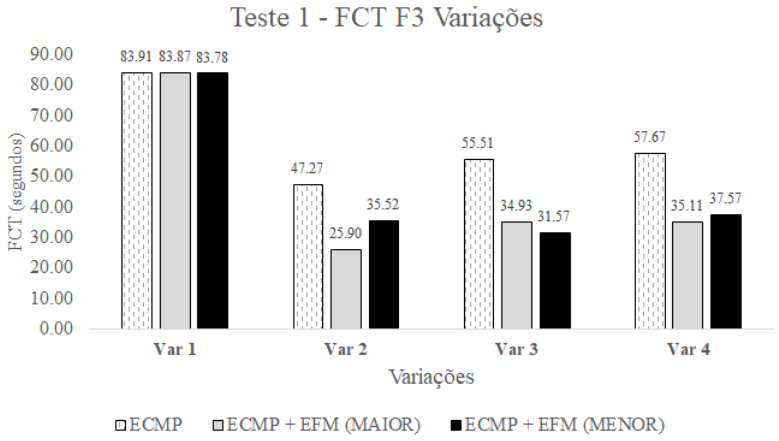


Figura 11 – FCT: Variações do F3 no Teste 1.

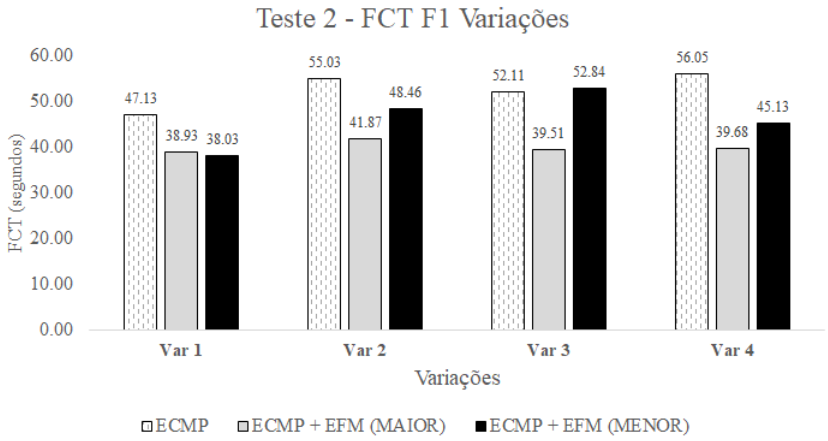


Figura 12 – FCT: Variações do F1 no Teste 2.

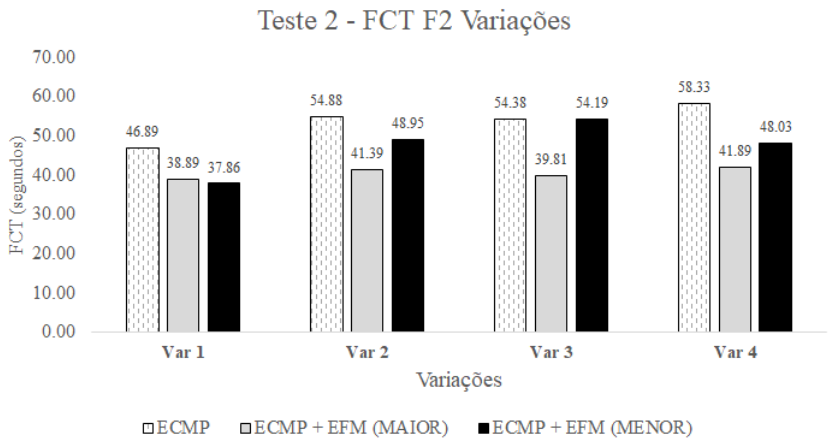


Figura 13 – FCT: Variações do F2 no Teste 2.

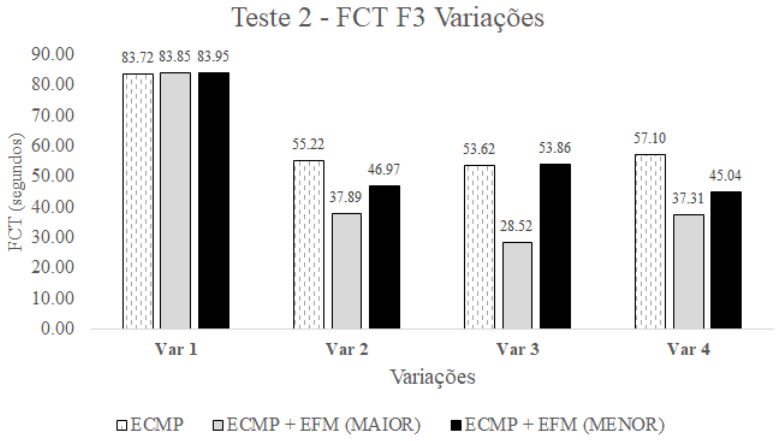


Figura 14 – FCT: Variações do F3 no Teste 2.

Nas Figuras de 9 a 14 apresentamos os principais resultados obtidos nas variações dos Testes 1 e 2 (colisões *upstream*). Para uma melhor compreensão, plotamos um gráfico para cada fluxo. A primeira conclusão dos nossos resultados é que o uso do EFM em conjunto com o ECMP na maioria dos testes obtém resultados superiores quando comparado ao uso do ECMP sem o EFM. Nos piores casos, o uso do EFM em conjunto com o ECMP apresenta resultados iguais ao uso do ECMP sem o EFM.

Nas Figs. 9 e 10 apresentamos os resultados para os Fluxos F1 e F2. Observamos que os ganhos no FCT são muito semelhantes independente da escolha do fluxo elefante (maior ou menor). Entretanto, na Fig. 11 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. Para o Teste 2 (usando o F4 como tráfego na rota ociosa), os resultados apresentados nas Figs. 12, 13 e 14 mostram que os Fluxos F1, F2 e F3 obtiveram um melhor resultado re-roteando o maior fluxo elefante em todos os casos. Em todas as variações não houveram diferenças nos resultados do F4, já que tal fluxo continuou a ser transmitido na sua taxa máxima, mesmo após o re-roteamento dos outros

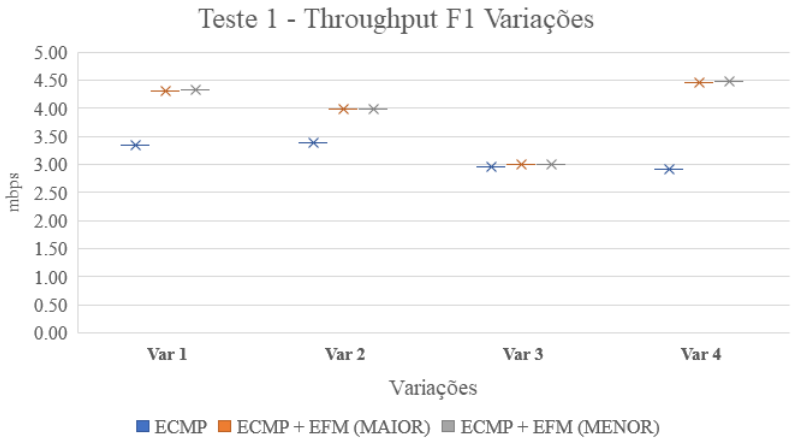


Figura 15 – Throughput: Variações do F1 no Teste 1.

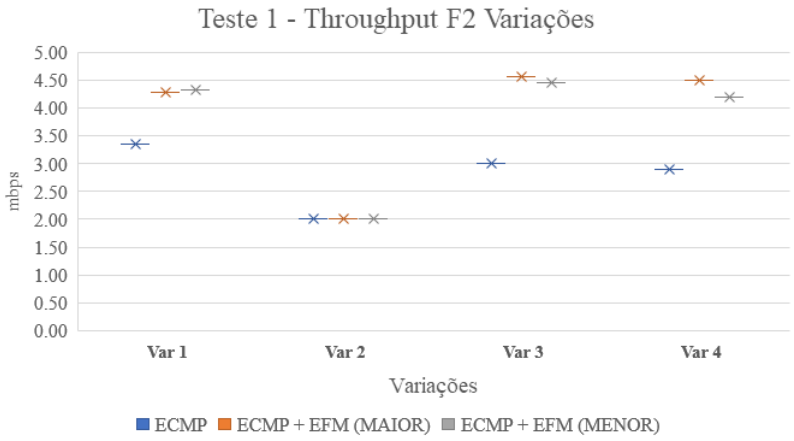


Figura 16 – Throughput: Variações do F2 no Teste 1.

fluxos.

Nas Figuras de 15 a 20 apresentamos os valores obtidos para o *throughput* dos Testes 1 e 2 em cada variação realizada. 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. Destaques para os ganhos das Figs. 16 e 17 na Variação 3 onde houve um aumento considerável para o *throughput* em relação ao ECMP.

Além das análises do FCT e *throughput* verificamos o número de retransmissões geradas pelo re-roteamento. As retransmissões já eram esperadas pois estamos alterando a rota de um fluxo enquanto ele ainda está trafegando pela rede, porém, é interessante analisar as diferenças entre re-rotear o maior e menor fluxo elefante. Por este motivo, decidimos coletar tal métrica para verificar se há impacto negativo no número de retransmissões com os resultados do FCT e *throughput*. Calculamos a média das retransmissões considerando todas as variações dos Testes 1 e 2.

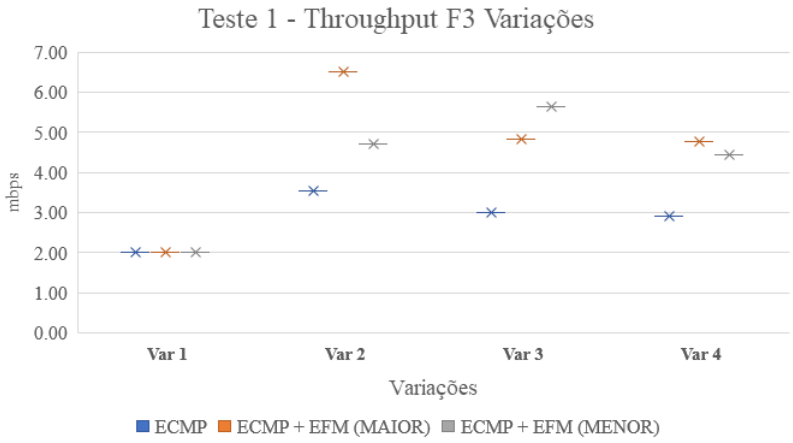


Figura 17 – Throughput: Variações do F3 no Teste 1.

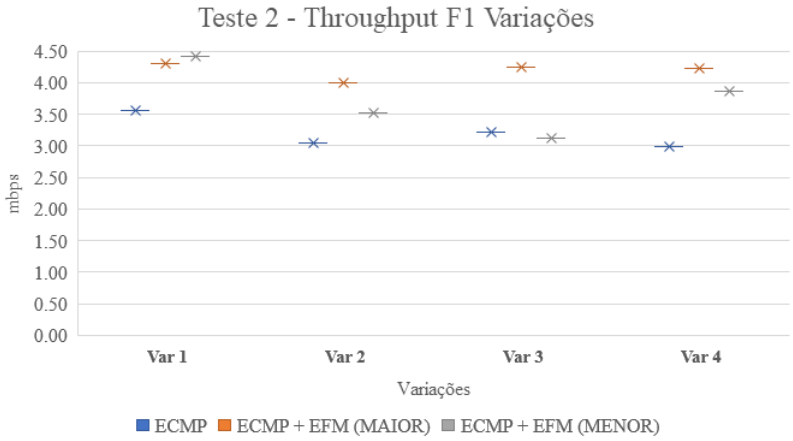


Figura 18 – Throughput: Variações do F1 no Teste 2.

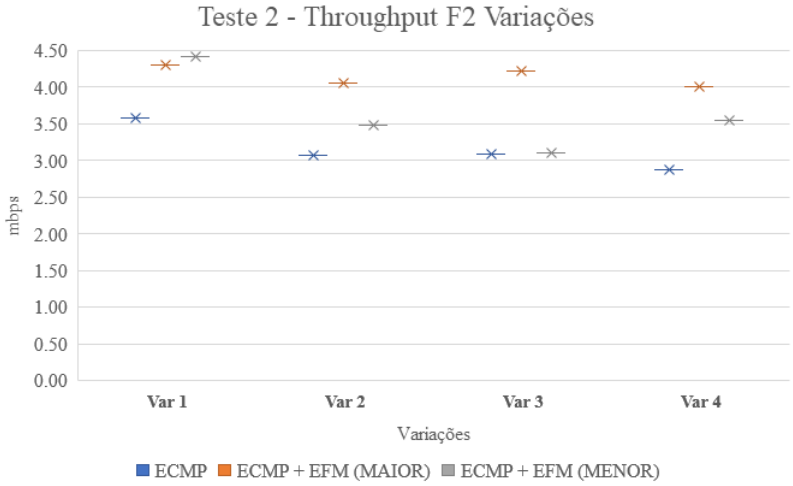


Figura 19 – Throughput: Variações do F2 no Teste 2.

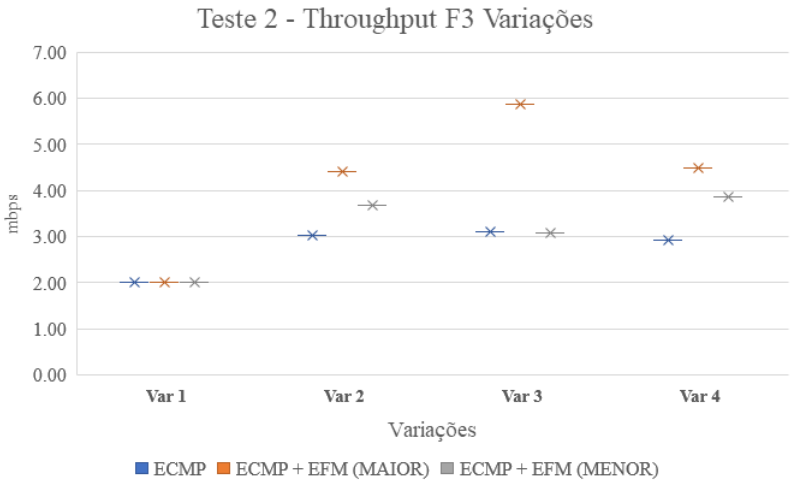


Figura 20 – Throughput: Variações do F3 no Teste 2.

Nas Figuras 21 e 22, observamos uma diferença grande no número de retransmissões quando re-roteamos o menor fluxo elefante ao invés do maior fluxo elefante. Em poucos casos como nas Variações 2 e 4 da Fig. 21 o contrário é constatado. Entretanto, predominantemente nos Testes 1 e 2, re-rotear o menor fluxo elefante gerou menos retransmissões.

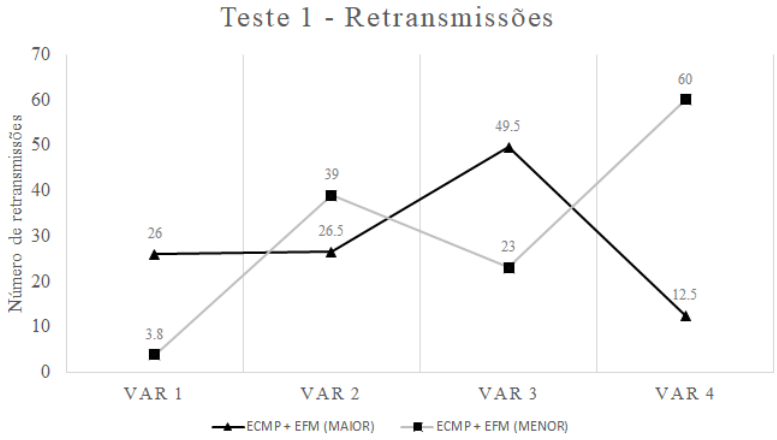


Figura 21 – Número de retransmissões do Teste 1.

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%.

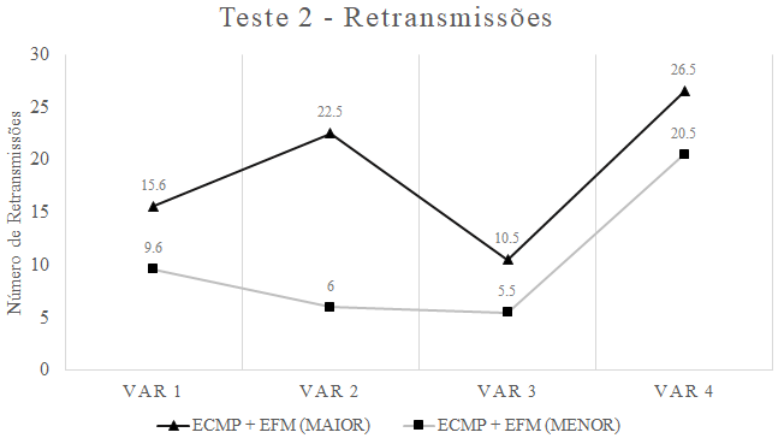


Figura 22 – Número de retransmissões do Teste 2.

No Apêndice A, apresentamos as tabelas com a porcentagem dos ganhos obtidos no FCT e *throughput* do EFM em relação ao ECMP.

4.3.2 Testes de Colisões Downstream

A metodologia utilizada nos testes desta Sub-Seção são iguais as descritas na Sub-Seção anterior. Dividimos o teste em dois e executamos quatro variações para cada teste. No Teste 4 foi acrescentado um quarto fluxo (F4) que irá trafegar nos enlaces que anteriormente no Teste 3 estavam ociosos. De modo semelhante, analisamos o FCT, o *throughput* e o número de retransmissões comparando os resultados obtidos do EFM + ECMP com o ECMP puro.

Na Tabela 6, 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 Mbps. 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. Na Tabela 7, ilustramos a origem e o destino de cada um dos fluxos criados para cada teste realizado. O posicionamento dos *hosts* na topologia utilizada pode ser visto na Fig. 7.

Tabela 6 – Parâmetros em Mbps para os Testes 3 e 4 - Colisões downstream.

		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

Tabela 7 – Origem e Destino dos fluxos para os Testes 3 e 4 - Colisões downstream.

		Origem	Destino
Teste 3	F1	H5	H3
	F2	H5	H3
	F3	H1	H4
Teste 4	F1	H5	H3
	F2	H5	H3
	F3	H1	H4
	F4	H1	H5

Os resultados para os Testes 3 e 4 são apresentados nas Figuras 23 a 28. Observamos na Fig. 23 que o FCT para o F1 foi menor quando re-roteamos o menor fluxo elefante. No caso do F2, mostrado na Fig. 24, temos que no geral é melhor re-rotear o menor fluxo elefante. Na Fig. 25, observamos que com exceção da Variação 2, é sempre melhor re-rotear o menor fluxo elefante. No caso do Teste 4 (usando o F4 como tráfego de fundo), observando os resultados apresentados nas Figs. 26, 27 and 28, em termos gerais neste teste é melhor re-rotearmos o maior fluxo elefante.

Os gráficos das Figs. 29 a 34 apresentam o *throughput* dos fluxos dos Testes 3 e 4. 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. Destaques para as Figs. 29, 30 e 31 onde em todas as variações possíveis o re-roteamento realizado pelo EFM foi superior ao ECMP, apresentando casos como na variação 2 da Fig. 31 em que o *throughput* do F3 dobrou. Nas Figs. 32, 33 e 34 a maioria dos resultados foi superior ao ECMP. Entretanto, nos piores casos os resultados foram iguais aos obtidos pelo ECMP.

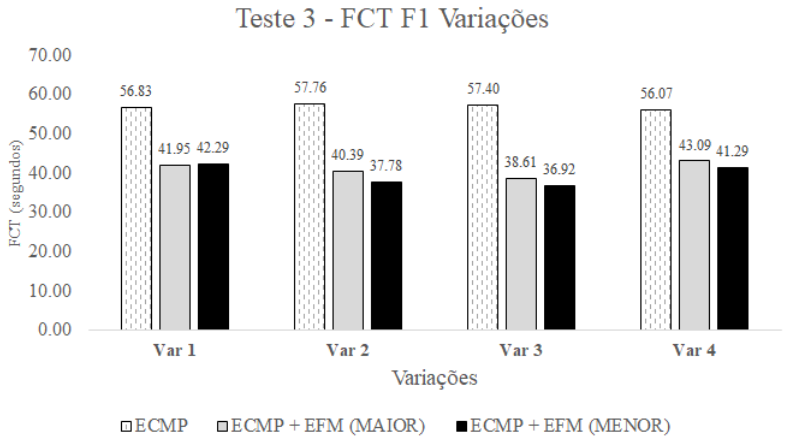


Figura 23 – FCT: Variações do F1 no Teste 3.

O número de retransmissões geradas nos Testes 3 e 4 são observadas nas Figs. 35 e 36. Assim como na análise do número de retransmissões dos Testes 1 e 2, houve uma leve predominância dos resultados obtidos re-roteando o menor fluxo elefante serem inferiores

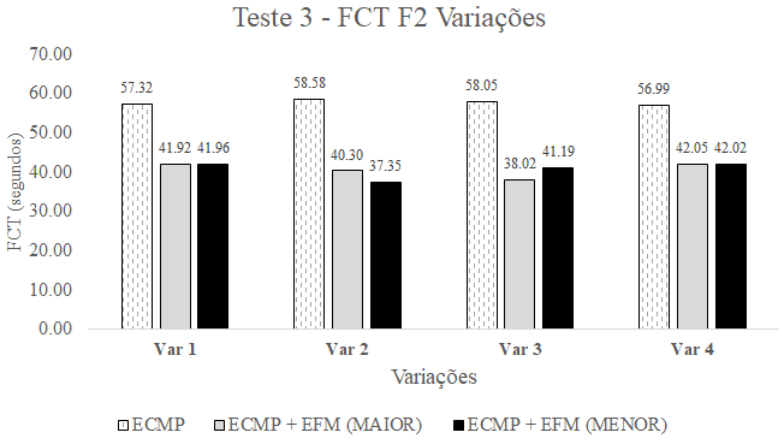


Figura 24 – FCT: Variações do F2 no Teste 3.

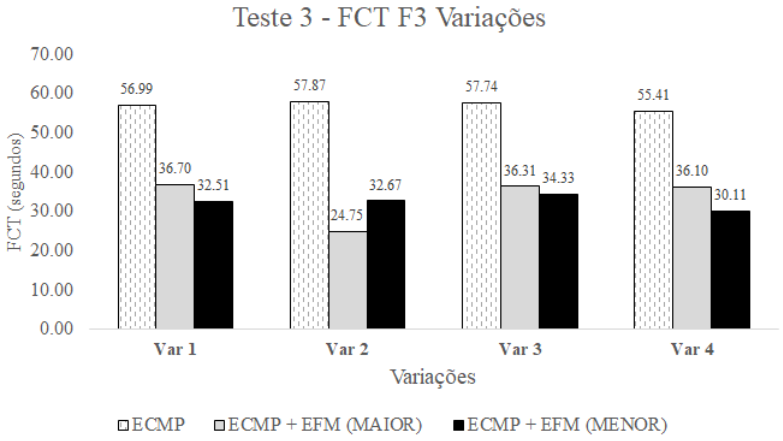


Figura 25 – FCT: Variações do F3 no Teste 3.

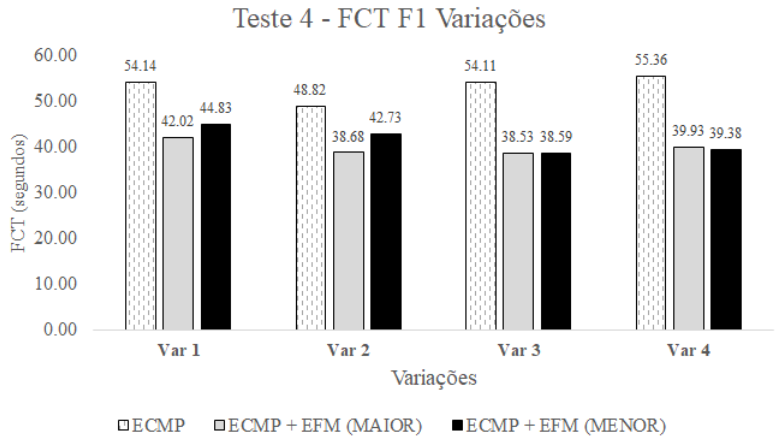


Figura 26 – FCT: Variações do F1 no Teste 4.

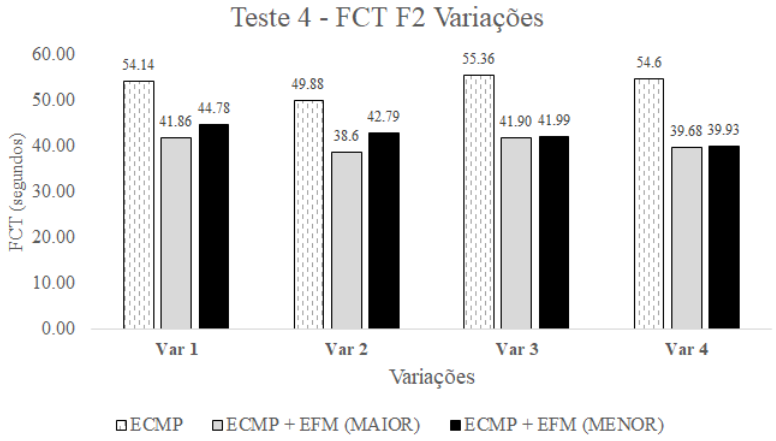


Figura 27 – FCT: Variações do F2 no Teste 4.

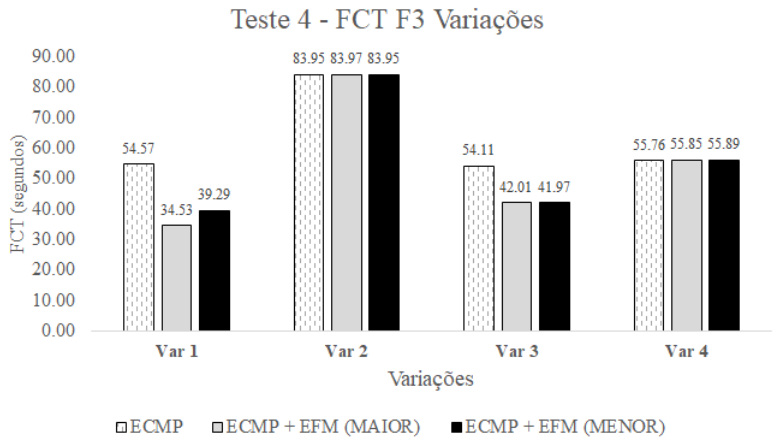


Figura 28 – FCT: Variações do F3 no Teste 4.

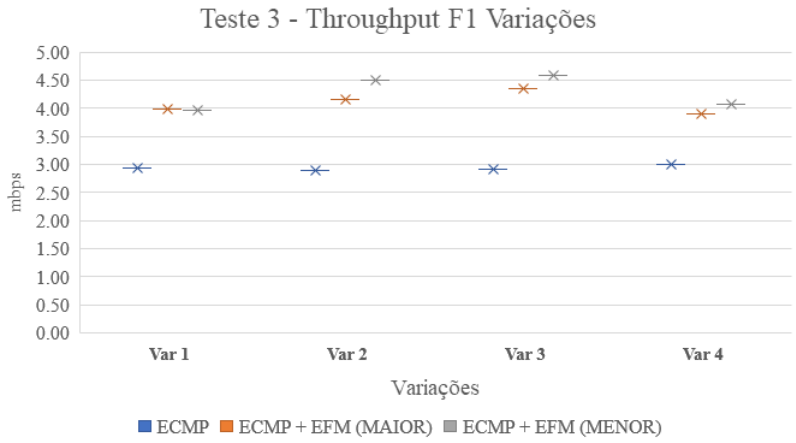


Figura 29 – Throughput: Variações do F1 no Teste 3.

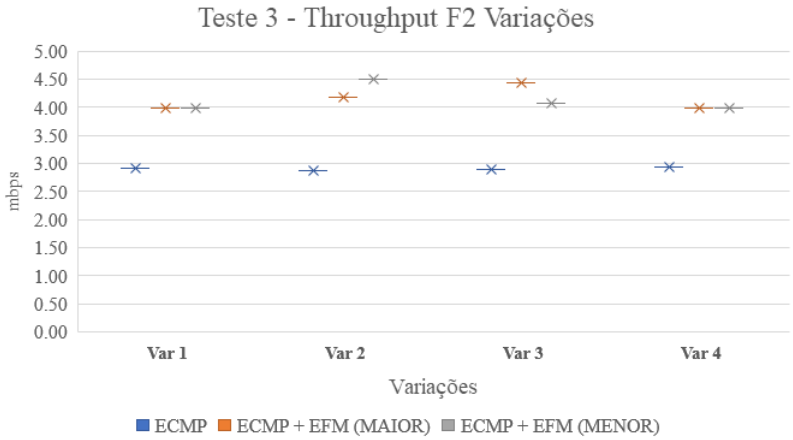


Figura 30 – Throughput: Variações do F2 no Teste 3.

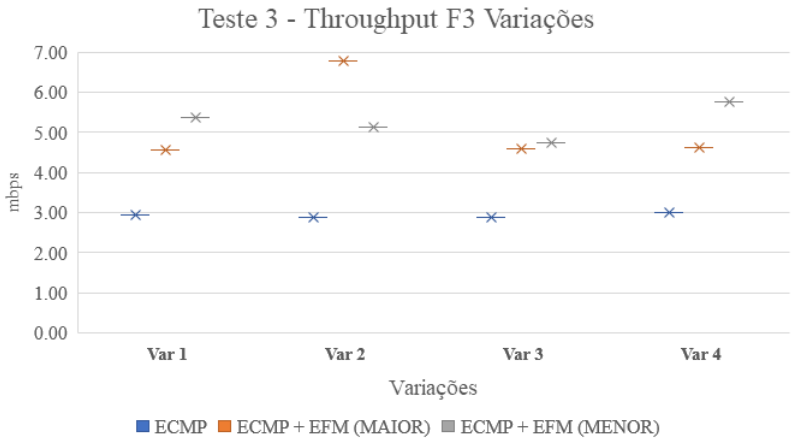


Figura 31 – Throughput: Variações do F3 no Teste 3.

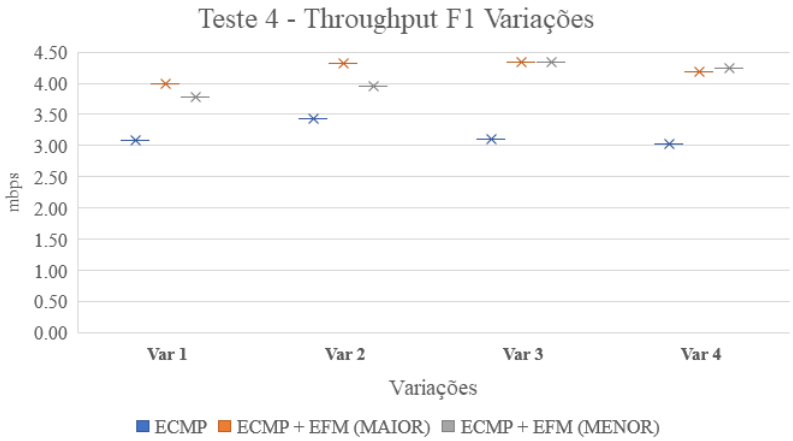


Figura 32 – Throughput: Variações do F1 no Teste 4.

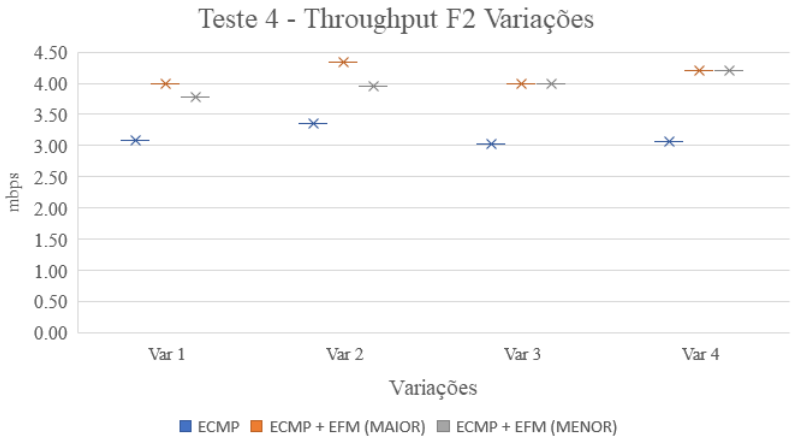


Figura 33 – Throughput: Variações do F2 no Teste 4.

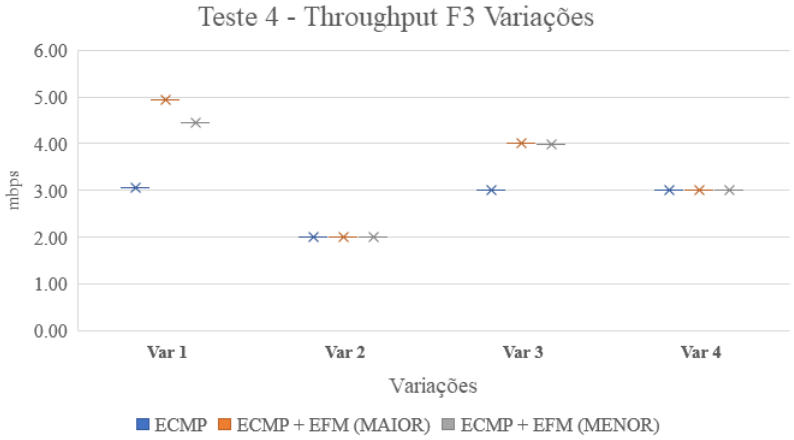


Figura 34 – Throughput: Variações do F3 no Teste 4.

ao re-rotear o maior fluxo elefante. Na maioria dos casos em que o re-roteamento do maior fluxo elefante foi superior, a diferença é muito pequena com relação ao menor fluxo elefante, como podemos observar nas Variações 1 e 3 da Fig. 35 e na Variação 4 da Fig. 36.

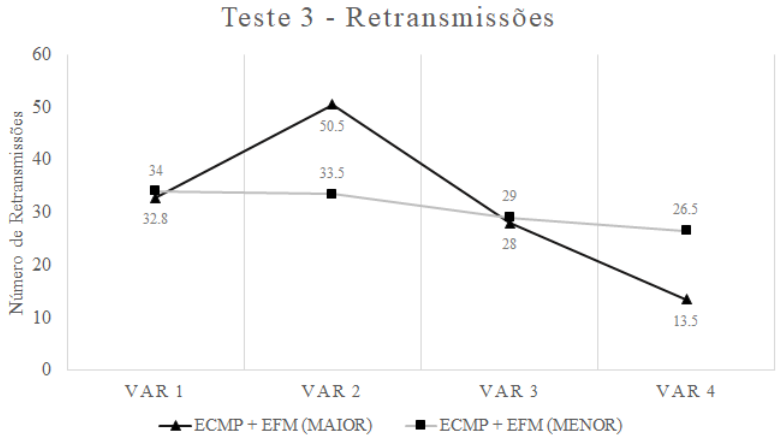


Figura 35 – Número de retransmissões do Teste 3.

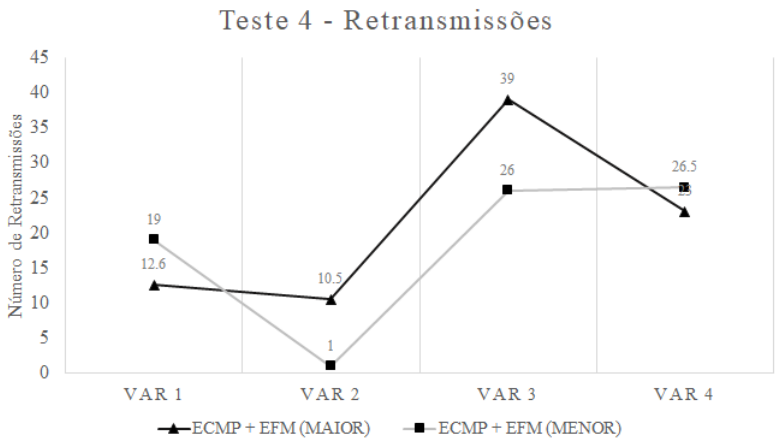


Figura 36 – Número de retransmissões do Teste 4.

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. Se considerarmos o número de retransmissões gerados, re-rotear o menor fluxo elefante possui uma ligeira vantagem. 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%.

No Apêndice B, apresentamos as tabelas com a porcentagem dos ganhos obtidos no FCT e *throughput* do EFM em relação ao ECMP.

4.3.3 Análise do Tempo de Execução

Realizamos a análise do tempo de execução das principais funções executadas pelo EFM, que são descritas a seguir: detecção e ordenação dos fluxos elefantes (*detectElephants*), cálculo de uma nova rota para os fluxos elefantes (*getRoutes*) e Atuação (*reroute*). Tais funções foram descritas em detalhes na Sub-Seção 3.3.2.

Os resultados estão em segundos e divididos entre re-rotear o maior fluxo elefante e o menor fluxo elefante. Calculamos o tempo de execução de cada uma das funções nas quatro variações realizadas por teste. A função *reroute* do EFM se mostrou mais custosa em termos de tempo de execução devido à requisição HTTP que ela executa para passar as informações da nova rota para o controlador OpenFlow. A função *getRoutes* possui um custo computacional intermediário quando comparado às outras duas funções e este custo está intimamente relacionado à quantidade de caminhos disponíveis entre um par de *hosts*. A função *detectElephants* possui o menor custo computacional devido à quantidade de fluxos elefantes presentes na rede, número este que costuma ser pequeno.

Nas Figs. 37 a 40, ilustramos os resultados do tempo de execução dos testes re-

alizados anteriormente. Analisando os gráficos, observamos que re-rotear o menor fluxo elefante exige menor tempo computacional do EFM quando comparado ao re-roteamento do maior fluxo elefante, na maioria dos casos.

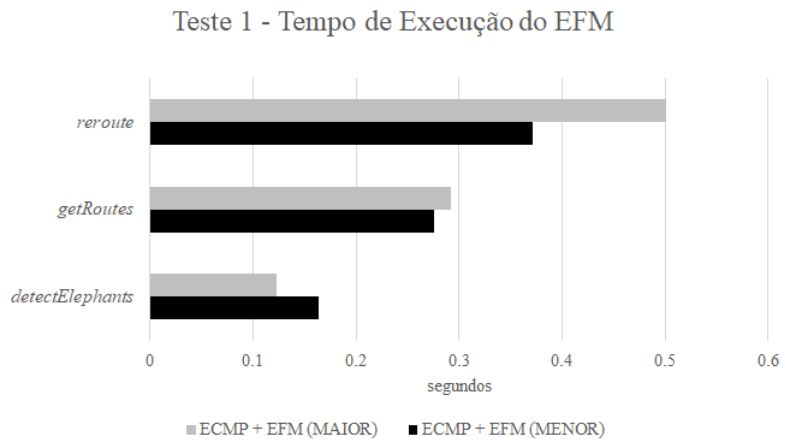


Figura 37 – Tempo de execução das funções do EFM no Teste 1.

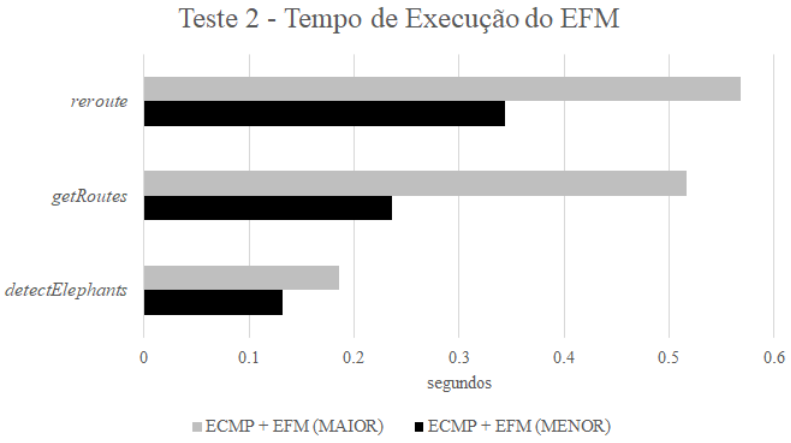


Figura 38 – Tempo de execução das funções do EFM no Teste 2.

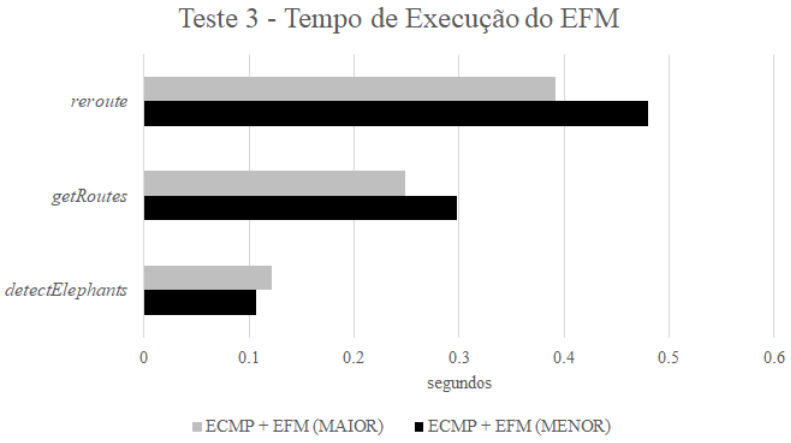


Figura 39 – Tempo de execução das funções do EFM no Teste 3.

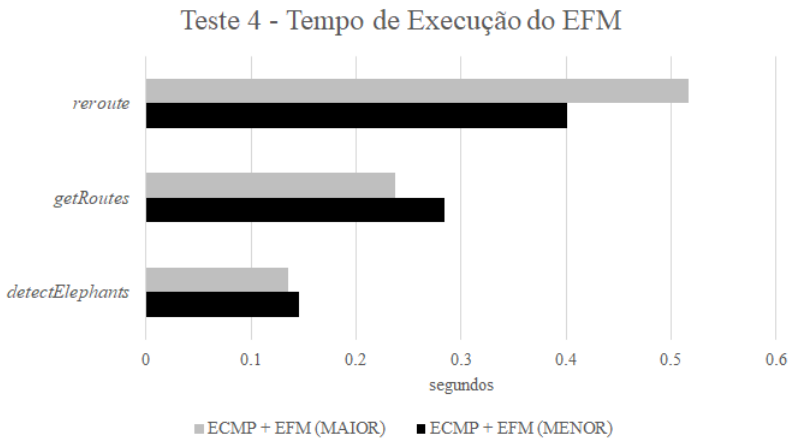


Figura 40 – Tempo de execução das funções do EFM no Teste 4.

4.3.4 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 diferença é que os *hosts* foram trocados por máquinas físicas para podermos coletar o consumo efetivo de energia. Para coletarmos as informações do consumo de energia dos *hosts*, utilizamos a ferramenta RAPL (DESROCHERS; PARADIS; WEAVER, 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*.

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 8. 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 do *package* e dos *cores* da CPU para cada alternativa de roteamento acima.

Tabela 8 – Parâmetros dos Testes sobre o consumo de energia.

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

Tabela 9 – Consumo médio de energia dos testes em Watts.

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

Os resultados obtidos para o Teste 1 estão expostos nas Figs. 41 e 42. 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 dois gráficos podemos analisar que em boa parte do tempo o consumo de energia executando apenas o ECMP é inferior a ambas as soluções do EFM. Entretanto, a partir do instante 241 o ECMP se mantém enquanto que o consumo com o EFM diminui durante um intervalo de tempo. 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.

Os resultados obtidos para o Teste 2 estão ilustrados nas Figs. 43 e 44. Este Teste apresentou conclusões semelhantes às do Teste 1. Ou seja, na maior parte do tempo o ECMP consumiu menos ou a mesma quantidade de energia que o EFM. Entretanto, o comportamento de diminuição do consumo de energia com o EFM também pôde ser observado de modo semelhante à analise feita no Teste 1, a partir do instante 196. Como os fluxos do *host* coletado encerraram antecipadamente devido à atuação do EFM, o consumo foi diminuindo gradativamente enquanto o consumo com apenas o ECMP permaneceu o mesmo até que os fluxos terminassem.

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 9 ilustra o consumo médio de energia (em *Watts*) dos dois testes apresentados. Observamos na tabela casos onde o EFM foi mais econômico que o ECMP e casos onde ocorreu o contrário. A economia de energia será sempre proporcional 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. Entretanto, caso o FCT apresente ganhos interessantes, a tendência na diminuição do consumo de energia pode ser vantajosa.

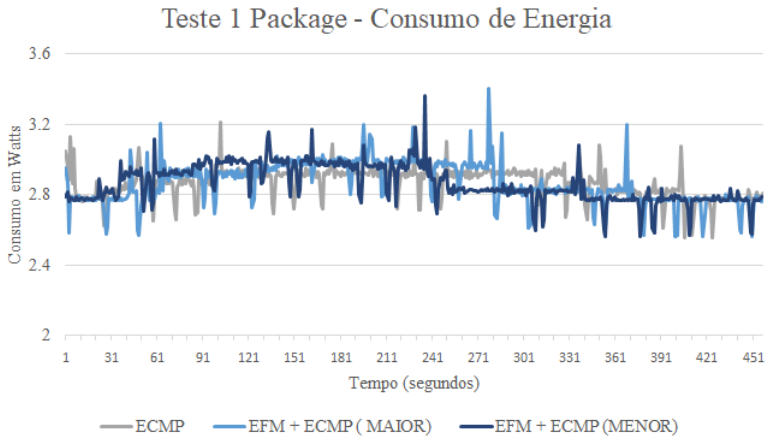


Figura 41 – Consumo de energia colisão upstream - Package.

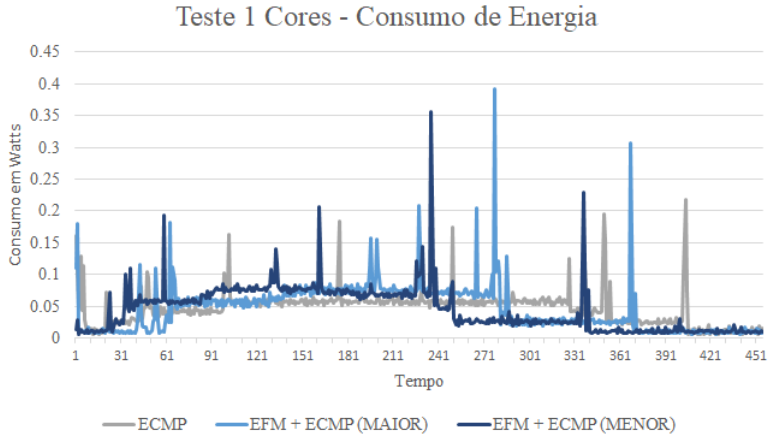


Figura 42 – Consumo de energia colisão upstream - Cores.

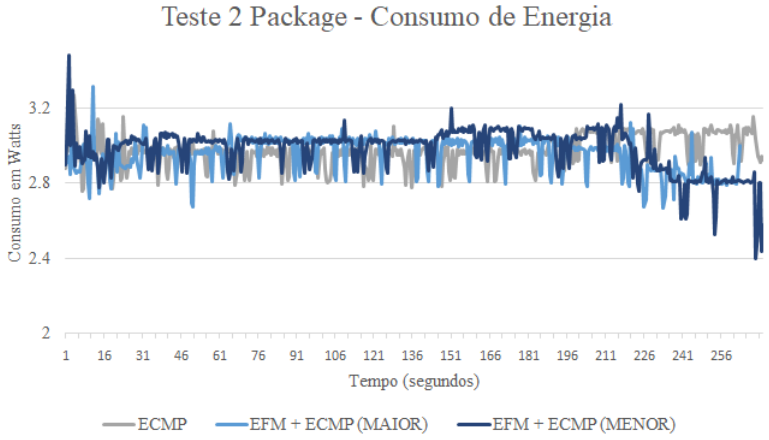


Figura 43 – Consumo de energia colisão downstream - Package.

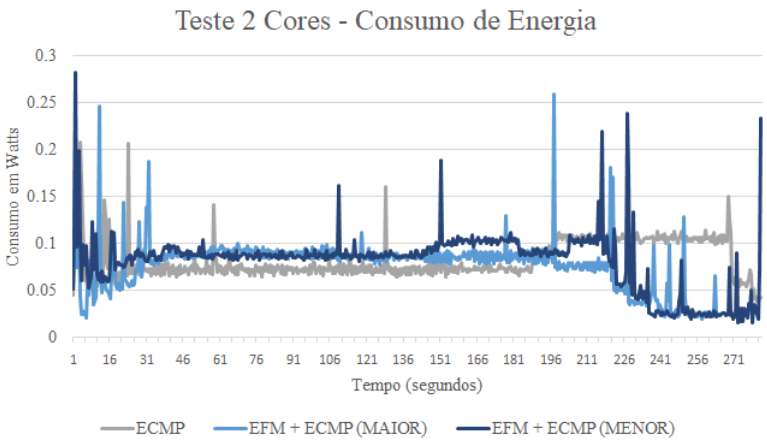


Figura 44 – Consumo de energia colisão downstream - Cores.

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.

A diferença entre re-rotear o maior e o menor fluxo elefante em alguns casos tende a ser alta. Desta forma, mostramos que conhecer a rede e seu estado (utilizando algoritmos de aprendizado de máquina, por exemplo) é extremamente útil para que a nossa solução possa aprender a tomar a melhor decisão de roteamento, dada as circunstâncias do tráfego ou das colisões detectadas na rede.

Por fim, alguns resultados preliminares mostraram que o EFM pode trazer benefícios para a economia de energia nos servidores dos DCs. Acreditamos que a ideia deste trabalho de mestrado possa ser amplamente utilizada em pesquisas futuras para que de fato beneficie grandes DCs, provendo uma melhoria na vazão dos fluxos assim como na diminuição do consumo de energia dos servidores.

Trabalhos Futuros

Este projeto tem a possibilidade de ser expandido e continuado em diversos pontos. Alguns deles são:

- Realização de testes do EFM em um cenário real, não emulado, com o intuito de coletar os resultados do re-roteamento do maior fluxo elefante e do menor fluxo elefante a longo prazo;
- Uma análise mais aprofundada sobre a economia de energia proporcionada pelo EFM. Seria interessante realizar este estudo em um ambiente real de modo a ser o mais fiel possível às DCNs;
- Utilizar algum *dataset* de DC para efetuar o treinamento supervisionado de qual a melhor escolha (re-rotear o maior ou o menor fluxo elefante) dado o estado atual

da rede, a ocupação dos enlaces, o tipo de congestionamento detectado e algumas características dos fluxos, dentre outras informações;

- Construir um ambiente WEB para o monitoramento e gerenciamento dos fluxos elefantes presentes em DCNs já que todas as informações necessárias para isto já estão sendo coletadas pelo EFM.

Submissões de artigos

1. EFM - Elephant Flow Manager: Uma Arquitetura para Detecção e Tratamento de Fluxos Elefantes em DCNs. SBRC 2018, Simpósio Brasileiro de Redes de Computadores. (Em processo de revisão).

2. EFM: Improving DCNs throughput using the transmission rates of elephant flows. IEEE ISCC 2018, IEEE Symposium on Computers and Communications. (Em processo de revisão).

Referências

AFAQ, M.; REHMAN, S. U.; SONG, W. C. Visualization of elephant flows and qos provisioning in sdn-based networks. In: *2015 17th Asia-Pacific Network Operations and Management Symposium (APNOMS)*. [S.l.: s.n.], 2015. p. 444–447. Citado 2 vezes nas páginas 35 e 36.

AL-FARES, M.; LOUKISSAS, A.; VAHDAT, A. A scalable, commodity data center network architecture. *SIGCOMM Comput. Commun. Rev.*, ACM, New York, NY, USA, v. 38, n. 4, p. 63–74, ago. 2008. ISSN 0146-4833. Disponível em: <http://doi.acm.org/10.1145/1402946.1402967>. Citado na página 25.

AL-FARES, M. et al. Hedera: Dynamic flow scheduling for data center networks. In: *NSDI*. [S.l.: s.n.], 2010. v. 10, p. 19–19. Citado 5 vezes nas páginas 29, 34, 35, 36 e 48.

BASHIR, S.; AHMED, N. Virtmone: Efficient detection of elephant flows in virtualized data centers. In: *2015 International Telecommunication Networks and Applications Conference (ITNAC)*. [S.l.: s.n.], 2015. p. 280–285. Citado 2 vezes nas páginas 34 e 35.

BENSON, T.; AKELLA, A.; MALTZ, D. A. Network traffic characteristics of data centers in the wild. In: *Proceedings of the 10th ACM SIGCOMM Conference on Internet Measurement*. New York, NY, USA: ACM, 2010. (IMC '10), p. 267–280. ISBN 978-1-4503-0483-2. Disponível em: <http://doi.acm.org/10.1145/1879141.1879175>. Citado na página 25.

BENSON, T. et al. Microte: Fine grained traffic engineering for data centers. In: *Proceedings of the Seventh COnference on Emerging Networking EXperiments and Technologies*. New York, NY, USA: ACM, 2011. (CoNEXT '11), p. 8:1–8:12. ISBN 978-1-4503-1041-3. Disponível em: <http://doi.acm.org/10.1145/2079296.2079304>. Citado na página 35.

CASADO, M. *Of Mice and Elephants*. 2013. Disponível em: <http://http://networkheresy.com/2013/11/01/of-mice-and-elephants/>. Acesso em: 5 mar. 2017. Citado na página 36.

CHOWDHURY, S. R. et al. Payless: A low cost network monitoring framework for software defined networks. In: *2014 IEEE Network Operations and Management Symposium (NOMS)*. [S.l.: s.n.], 2014. p. 1–9. ISSN 1542-1201. Citado na página 34.

CLAISE, B.; ED. *RFC 3954*. 2004. Disponível em: <http://www.ietf.org/rfc/rfc3954.txt>. Acesso em: 21 set. 2017. Citado na página 32.

CLAISE, B.; ED. *RFC 5101*. 2008. Disponível em: <https://tools.ietf.org/rfc/rfc5101.txt>. Acesso em: 30 jan. 2017. Citado na página 32.

CORPORATION, I. *sFlow-RT Page*. 2004. Disponível em: <http://www.inmon.com/products/sFlow-RT.php>. Acesso em: 20 fev. 2017. Citado na página 43.

CURTIS, A. R.; KIM, W.; YALAGANDULA, P. Mahout: Low-overhead datacenter traffic management using end-host-based elephant detection. In: *2011 Proceedings IEEE*

INFOCOM. [S.l.: s.n.], 2011. p. 1629–1637. ISSN 0743-166X. Citado 4 vezes nas páginas 30, 34, 35 e 36.

CURTIS, A. R. et al. Devoflow: Scaling flow management for high-performance networks. *SIGCOMM Comput. Commun. Rev.*, ACM, New York, NY, USA, v. 41, n. 4, p. 254–265, ago. 2011. ISSN 0146-4833. Disponível em: <http://doi.acm.org/10.1145/2043164.2018466>. Citado 3 vezes nas páginas 29, 35 e 36.

DESROCHERS, S.; PARADIS, C.; WEAVER, V. M. A validation of dram rapl power measurements. In: *Proceedings of the Second International Symposium on Memory Systems*. New York, NY, USA: ACM, 2016. (MEMSYS '16), p. 455–470. ISBN 978-1-4503-4305-3. Disponível em: <http://doi.acm.org/10.1145/2989081.2989088>. Citado na página 64.

FARRINGTON, N. et al. Helios: A hybrid electrical/optical switch architecture for modular data centers. In: *Proceedings of the ACM SIGCOMM 2010 Conference*. New York, NY, USA: ACM, 2010. (SIGCOMM '10), p. 339–350. ISBN 978-1-4503-0201-2. Disponível em: <http://doi.acm.org/10.1145/1851182.1851223>. Citado 2 vezes nas páginas 35 e 36.

FLOODLIGHT, P. *Documentation of Floodlight*. 2008. Disponível em: <http://www.projectfloodlight.org/documentation/>. Acesso em: 10 fev. 2017. Citado na página 43.

KANAGEVLU, R.; AUNG, K. M. M. Sdn controlled local re-routing to reduce congestion in cloud data center. In: *2015 International Conference on Cloud Computing Research and Innovation (ICCCRI)*. [S.l.: s.n.], 2015. p. 80–88. Citado 2 vezes nas páginas 26 e 30.

KATTA, N. et al. 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*. New York, NY, USA: ACM, 2016. (HotNets '16), p. 155–161. ISBN 978-1-4503-4661-0. Disponível em: <http://doi.acm.org/10.1145/3005745.3005751>. Citado na página 25.

MANN, V.; VISHNOI, A.; BIDKAR, S. Living on the edge: Monitoring network flows at the edge in cloud data centers. In: *2013 Fifth International Conference on Communication Systems and Networks (COMSNETS)*. [S.l.: s.n.], 2013. p. 1–9. ISSN 2155-2487. Citado na página 35.

MCKEOWN, N. et al. Openflow: Enabling innovation in campus networks. *SIGCOMM Comput. Commun. Rev.*, ACM, New York, NY, USA, v. 38, n. 2, p. 69–74, mar. 2008. ISSN 0146-4833. Disponível em: <http://doi.acm.org/10.1145/1355734.1355746>. Citado 3 vezes nas páginas 29, 31 e 32.

NEO4J. *Documentation of Neo4J*. 2016. Disponível em: <http://neo4j.com/>. Acesso em: 20 mar. 2017. Citado na página 43.

NLANR/DAST. *Iperf - The TCP/UDP Bandwidth Measurement Tool*. 2007. Disponível em: <http://dast.nlanr.net/Projects/Iperf/>. Acesso em: 16 fev. 2017. Citado na página 48.

PETER. *Sampling Rates*. 2013. Disponível em: <http://blog.sflow.com/2013/06/large-flow-detection.html>. Acesso em: 15 fev. 2017. Citado na página 33.

PHAAL, P. *sFlow Version 5 Specification*. 2004. Disponível em: <http://www.sflow.org/sflow_version_5.txt>. Acesso em: 26 ago. 2017. Citado 2 vezes nas páginas 32 e 43.

PHAAL, P.; PANCHEN, S. *Packet Sampling Basics*. 2004. Disponível em: <http://www.sflow.org/packetSamplingBasics/index.html>. Acesso em: 6 fev. 2017. Citado na página 33.

PHAAL, P.; PANCHEN, S.; MCKEE, N. *InMon Corporation's sFlow: A Method for Monitoring Traffic in Switched and Routed Networks*. United States: RFC Editor, 2001. Citado na página 35.

PRESUHN, R. *Version 2 of the Protocol Operations for the Simple Network Management Protocol (SNMP)*. 2002. Disponível em: <http://tools.ietf.org/html/rfc3416>. Acesso em: 19 set. 2017. Citado na página 32.

RASLEY, J. et al. Planck: Millisecond-scale monitoring and control for commodity networks. *SIGCOMM Comput. Commun. Rev.*, ACM, New York, NY, USA, v. 44, n. 4, p. 407–418, ago. 2014. ISSN 0146-4833. Disponível em: <http://doi.acm.org/10.1145/2740070.2626310>. Citado 2 vezes nas páginas 32 e 34.

SILVA, A.; VERDI., F. L. Empowering applications with rfc 6897 to manage elephant flows in datacenter networks. In: *IEEE International Conference on Cloud Networking (Cloudnet)*. [S.l.: s.n.], 2017. p. 1–6. Citado 2 vezes nas páginas 26 e 36.

TEAM, M. *Documentation of Mininet*. 2007. Disponível em: <https://github.com/mininet/mininet/wiki/Documentation>. Acesso em: 15 fev. 2017. Citado na página 47.

TEAM, O. vSwitch. *Open vSwitch*. 2014. Disponível em: <http://openvswitch.org>. Acesso em: 10 fev. 2017. Citado na página 35.

XU, H.; LI, B. Tinyflow: Breaking elephants down into mice in data center networks. In: *LANMAN*. IEEE, 2014. p. 1–6. ISBN 978-1-4799-4894-9. Disponível em: <http://dblp.uni-trier.de/db/conf/lanman/lanman2014.html#XuL14>. Citado 6 vezes nas páginas 25, 26, 29, 34, 35 e 36.

YU, C. et al. Flowsense: Monitoring network utilization with zero measurement cost. In: *Proceedings of the 14th International Conference on Passive and Active Measurement*. Berlin, Heidelberg: Springer-Verlag, 2013. (PAM'13), p. 31–41. ISBN 978-3-642-36515-7. Disponível em: <http://dx.doi.org/10.1007/978-3-642-36516-4_4>. Citado na página 34.

ZHAO, J. et al. Multipath tcp for datacenters: From energy efficiency perspective. In: *IEEE INFOCOM 2017 - IEEE Conference on Computer Communications*. [S.l.: s.n.], 2017. p. 1–9. Citado na página 26.

APÊNDICE A – Porcentagem de Ganho do EFM em relação ao ECMP - Testes 1 e 2

Tabela 10 – Porcentagem de ganho do EFM em relação ao ECMP - FCT Teste 1.

	Maior	Menor
F1 - Variação 1	29.01 %	29.23 %
F1 - Variação 2	18.57 %	18.22 %
F1 - Variação 3	1.47 %	1.74 %
F1 - Variação 4	51.64 %	53.73 %
F2 - Variação 1	28.03 %	28.81 %
F2 - Variação 2	-0.02 %	0.17 %
F2 - Variação 3	50.28 %	47.48 %
F2 - Variação 4	53.65 %	44.72 %
F3 - Variação 1	0.05 %	0.16 %
F3 - Variação 2	82.51 %	33.08 %
F3 - Variação 3	58.94 %	75.86 %
F3 - Variação 4	64.28 %	53.50 %

Tabela 11 – Porcentagem de ganho do EFM em relação ao ECMP - Throughput Teste 1.

	Maior	Menor
F1 - Variação 1	28.66 %	28.96 %
F1 - Variação 2	18.05 %	18.05 %
F1 - Variação 3	1.35 %	1.18 %
F1 - Variação 4	52.74 %	53.60 %
F2 - Variação 1	28.06 %	28.96 %
F2 - Variação 2	0.00 %	0.00 %
F2 - Variação 3	51.5 %	61.46 %
F2 - Variação 4	55.36 %	44.81 %
F3 - Variação 1	0.00 %	0.00 %
F3 - Variação 2	83.90 %	33.05 %
F3 - Variação 3	60.47 %	131.89 %
F3 - Variação 4	64.14 %	52.76 %

Tabela 12 – Porcentagem de ganho do EFM em relação ao ECMP - FCT Teste 2.

	Maior	Menor
F1 - Variação 1	21.04 %	23.93 %
F1 - Variação 2	31.43 %	13.56 %
F1 - Variação 3	31.91 %	9.04 %
F1 - Variação 4	41.27 %	24.21 %
F2 - Variação 1	20.59 %	23.85 %
F2 - Variação 2	32.59 %	12.11 %
F2 - Variação 3	36.60 %	12.34 %
F2 - Variação 4	39.26 %	21.44 %
F3 - Variação 1	-0.15 %	-0.28 %
F3 - Variação 2	45.74 %	17.56 %
F3 - Variação 3	88.04 %	16.00 %
F3 - Variação 4	53.04 %	26.79 %

Tabela 13 – Porcentagem de ganho do EFM em relação ao ECMP - Throughput Teste 2.

	Maior	Menor
F1 - Variação 1	20.79 %	23.88 %
F1 - Variação 2	31.58 %	15.79 %
F1 - Variação 3	32.24 %	-2.65 %
F1 - Variação 4	41.61 %	29.87 %
F2 - Variação 1	20.45 %	23.81 %
F2 - Variação 2	32.35 %	13.56 %
F2 - Variação 3	36.85 %	0.65 %
F2 - Variação 4	39.37 %	23.52 %
F3 - Variação 1	0.00 %	0.00 %
F3 - Variação 2	46.03 %	21.85 %
F3 - Variação 3	88.42 %	-1.13 %
F3 - Variação 4	53.42 %	31.85 %

APÊNDICE B – Porcentagem de Ganho do EFM em relação ao ECMP - Testes 3 e 4

Tabela 14 – Porcentagem de ganho do EFM em relação ao ECMP - FCT Teste 3.

	Maior	Menor
F1 - Variação 1	55.26 %	75.27 %
F1 - Variação 2	133.87 %	77.13 %
F1 - Variação 3	59.02 %	68.22 %
F1 - Variação 4	53.49 %	84.03 %
F2 - Variação 1	35.45 %	34.36 %
F2 - Variação 2	43.02 %	52.91 %
F2 - Variação 3	48.69 %	55.49 %
F2 - Variação 4	30.12 %	35.81 %
F3 - Variação 1	36.72 %	36.59 %
F3 - Variação 2	45.38 %	56.84 %
F3 - Variação 3	52.68 %	40.93 %
F3 - Variação 4	35.53 %	35.63 %

Tabela 15 – Porcentagem de ganho do EFM em relação ao ECMP - Throughput Teste 3.

	Maior	Menor
F1 - Variação 1	35.37 %	34.69 %
F1 - Variação 2	43.10 %	54.83 %
F1 - Variação 3	48.97 %	56.68 %
F1 - Variação 4	30.10 %	35.95 %
F2 - Variação 1	36.64 %	36.64 %
F2 - Variação 2	45.80 %	57.34 %
F2 - Variação 3	53.29 %	40.66 %
F2 - Variação 4	35.71 %	35.54 %
F3 - Variação 1	54.76 %	82.65 %
F3 - Variação 2	135.07 %	77.60 %
F3 - Variação 3	59.17 %	64.01 %
F3 - Variação 4	53.82 %	91.36 %

Tabela 16 – Porcentagem de ganho do EFM em relação ao ECMP - FCT Teste 4.

	Maior	Menor
F1 - Variação 1	58.05 %	38.89 %
F1 - Variação 2	-0.02 %	0.01 %
F1 - Variação 3	28.82 %	28.94 %
F1 - Variação 4	-0.16 %	-0.22 %
F2 - Variação 1	29.34 %	20.90 %
F2 - Variação 2	29.22 %	16.58 %
F2 - Variação 3	32.14 %	31.84 %
F2 - Variação 4	37.62 %	36.74 %
F3 - Variação 1	28.86 %	20.76 %
F3 - Variação 2	26.23 %	14.25 %
F3 - Variação 3	40.44 %	40.24 %
F3 - Variação 4	38.64 %	40.60 %

Tabela 17 – Porcentagem de ganho do EFM em relação ao ECMP - Throughput Teste 4.

	Maior	Menor
F1 - Variação 1	25.24 %	0.00 %
F1 - Variação 2	25.58 %	14.68 %
F1 - Variação 3	40.00 %	39.84 %
F1 - Variação 4	38.74 %	40.73 %
F2 - Variação 1	14.89 %	0.00 %
F2 - Variação 2	29.17 %	17.56 %
F2 - Variação 3	31.68 %	31.85 %
F2 - Variação 4	37.13 %	36.81 %
F3 - Variação 1	25.82 %	0.00 %
F3 - Variação 2	0.00 %	0.00 %
F3 - Variação 3	32.45 %	32.28 %
F3 - Variação 4	0.00 %	0.00 %