Universidade Federal de Juiz de Fora

Instituto de Ciências Exatas

Programa de Pós-Graduação em Ciência da Computação

Wallace Nascimento Paraizo

Gerenciamento de Tráfego Dinâmico com Ciência de Energia para Redes Definidas por Software

Juiz de Fora

2018

Wallace Nascimento Paraizo

Gerenciamento de Tráfego Dinâmico com Ciência de Energia para Redes Definidas por Software

Dissertação apresentada ao Programa de Pós-Graduação em Ciência da Computação da Universidade Federal de Juiz de Fora, como requisito parcial para obtenção do título de Mestre em Ciência da Computação.

Orientador: Alex Borges Vieira

Juiz de Fora

2018

Ficha catalográfica elaborada através do programa de geração
automática da Biblioteca Universitária da UFJF,
com os dados fornecidos pelo(a) autor(a)

Paraizo, Wallace Nascimento.
Gerenciamento de tráfego dinâmico com ciência de energia para Redes Definidas por Software / Wallace Nascimento Paraizo. -- 2018.
93 p.

Orientador: Alex Borges Vieira
Dissertação (mestrado acadêmico) - Universidade Federal de Juiz de Fora, Instituto de Ciências Exatas. Programa de Pós Graduação em Ciência da Computação, 2018.

1. Eficiência energética. 2. Redes Definidas por Software. 3. Gerência de redes. 4. Redes de computadores. I. Vieira, Alex Borges, orient. II. Título.

Wallace Nascimento Paraizo

Gerenciamento de Tráfego Dinâmico com Ciência de Energia para Redes Definidas por Software

Dissertação apresentada ao Programa de Pós-Graduação em Ciência da Computação da Universidade Federal de Juiz de Fora, como requisito parcial para obtenção do título de Mestre em Ciência da Computação.

Aprovada em 21/06/2018:

BANCA EXAMINADORA

Professor Dr. Alex Borges Vieira - Orientador
Universidade Federal de Juiz de Fora

Professor Dr. Eduardo Pagani Julio
Universidade Federal de Juiz de Fora

Professor Dr. Mario Antônio Ribeiro Dantas
Universidade Federal de Juiz de Fora

Professor Dr. Leobino Nascimento Sampaio
Universidade Federal da Bahia

AGRADECIMENTOS

Agradeço aos meus pais por todo o apoio durante a minha vida. Além do mais agradeço a minha mãe Regina Célia que me apoiou nos estudos em momentos de dificuldades financeiras e período que ninguém acreditava em mim. Agradeço Minha esposa Linda Carla por além de aturar os diversos finais de semana sem passeios e filmes, me dar apoio para continuar lutando.

Agradeço a todos do Critt / UFJF e por compreender minha menor dedicação em alguns momentos.

Não posso deixar de agradecer ao meu orientador Alex, principalmente por acreditar em mim e ter compreendido os diversos momentos de dificuldades que passei por problemas de saúde na família. O professor Luciano também foi fundamental no decorrer do desenvolvimento do projeto, dispondo de seu escasso tempo para me ajudar. Sem sua valorosa contribuição para este trabalho, seria mais difícil de ser realizado.

Terei problemas ao agradecer nominalmente a todos os colegas que me ajudaram durante a passagem pelo PGCC, pois foram muitos.

Agradeço à UFJF e ao PGCC por possibilitarem essa formação e também agradeço os diversos professores e funcionários da UFJF pelo apoio desde o início do Mestrado. São pessoas dedicadas à educação pública de qualidade.

"Quando tudo nos parece dar errado acontecem coisas boas que não teriam acontecido se tudo tivesse dado certo."

Renato Russo

RESUMO

Nos dias atuais, o consumo de energia é uma questão central em redes de computadores. Em muitos casos, os recursos de rede são superdimensionados para lidar com a sobrecarga de tráfego, o que obviamente, incorre em desperdício de recursos e alto custo de energia durante a maior parte do tempo. Entretanto, é possível desativar enlaces e dispositivos de rede, como comutadores, para reduzir o consumo de energia durante períodos de subutilização e, como consequência, reduzir os custos operacionais. Neste trabalho, nós apresentamos um mecanismo de gerenciamento de tráfego dinâmico ciente do consumo de energia utilizando redes definidas por software ou *Software-Defined Networks*(SDN). SDN permite uma programabilidade de rede, de forma flexível, através de um controlador centralizado. Tal programabilidade não pode ser facilmente alcançada por arquiteturas de rede tradicionais. O mecanismo proposto explora esta característica para, dinamicamente, melhorar o roteamento de trafego através dos comutadores, desativando enlaces e permitindo que o controlador SDN desligue um comutador em tempo real. Além do mais, o controlador tenta agregar os tráfegos de fluxos, reduzindo o número total de caminhos fim-a-fim, permitindo uma melhor utilização da rede. Avaliamos o mecanismo proposto, emulando um ambiente SDN realista baseado em uma topologia típica de campus. Consideramos diferentes níveis de tráfego, partindo de uma carga leve até cenários com sobrecarga de tráfego. Os resultados evidenciam que a economia de energia alcançada pela solução proposta, é de 46,01% em um cenário de baixa carga, 36,72% em um cenário de carga média e 17,86% em um cenário considerado sobrecarregado.

Palavras-chave: Eficiência energética. Redes definidas por software. Gerência de redes. Redes de computadores.

ABSTRACT

Energy consumption is a central issue in computer networks nowadays. In most cases, networks resources are overprovisioned to handle overloaded traffic, which clearly incurs into waist of resources and high energy costs during the major time. However, it is possible to proper deactivate links and network devices, as switches, to reduce energy consumption during underutilization periods and, as consequence, decrease operational costs. In this work, we present an energy-aware dynamic traffic management mechanism for Software-Defined Networks (SDN). SDN allows a flexible network programmability through a centralized controller, which cannot be achieved by traditional architectures. The proposed mechanism explores this feature to optimize traffic routing throughout switches, disabling links and allowing the SDN controller to turn-off switches on the fly. Moreover, it dynamically tries to aggregate traffic flows, reducing the total number of end-to-end paths, which allows a better network usage. We evaluated the proposed mechanism by emulating a realistic campus-based SDN environment. We considered different traffic levels, ranging from lightweight to overloaded scenarios. Results highlight the energy savings achieved by the proposed solution is 46.01% in a low load scenario, 36.72 % in a medium load scenario and 17.86 % in a considered overloaded scenario.

Key-words: Energy-efficiency. Software-defined networking. Network management. Computer networks.

LISTA DE ILUSTRAÇÕES

Figura 1 – Emissões globais das TICs [Ericson, 2016] 13
Figura 2 – Otimização energética vs não otimização [Rodrigues, 2016] 15
Figura 3 – Padrão de tráfego semanal global [Markiewicz et al., 2014] 15
Figura 4 – Linha cronológica de tecnologias que ajudaram a desenvolver paradigma SDN [Costa, 2016] 20
Figura 5 – Arquitetura típica de uma rede SDN [Costa, 2016] 23
Figura 6 – Controlador SDN. [Lobato et al., 2013] 24
Figura 7 – Separação de planos. [Lobato et al., 2013] 24
Figura 8 – Relação entre o controlador e as interfaces *Northbound* e *Southbound*. [Costa, 2016] 25
Figura 9 – Taxonomia de abordagens de eficiência energética em redes com fio [Rodrigues, 2016] 30
Figura 10 – Taxonomia de abordagens de eficiência energética [Rodrigues, 2016] . . 32
Figura 11 – Escopo de arquitetura [Rodrigues, 2016] 33
Figura 12 – Proposta para concentrar fluxos e desligar pare dos comutadores 37
Figura 13 – Visão geral da nossa proposta 38
Figura 14 – Abordagem proposta. 39
Figura 15 – Fluxograma: Novo fluxo 41
Figura 16 – Modelo da rede 43
Figura 17 – Demonstração do algoritmo de redirecionamento 50
Figura 18 – Fluxograma monitoramento da rede 52
Figura 19 – Estados assumidos pelo enlace 55
Figura 20 – Comportamento do mecanismo no cenário estático 58
Figura 21 – Comportamento do mecanismo no cenário dinâmico 60
Figura 22 – % de economia de energia versus carga 61
Figura 23 – eventos que geram picos de latência 61
Figura 24 – Topologia típica de uma rede de campus. [Markiewicz et al., 2014] . . . 63
Figura 25 – Cenários: Enlaces homogêneos vs Enlaces heterogêneos 64
Figura 26 – Vazão média em cada cenário 65
Figura 27 – Comutadores ligados 66
Figura 28 – Topologia típica de uma rede de campus. [Markiewicz et al., 2014] . . . 67

LISTA DE ABREVIATURAS E SIGLAS

SDN	*Software Defined Networks*
TCAM	*Ternary Content Addressable Memory*
CDF	*Cumulative Distribution Function*
WAN	*Wide Area Network*
ASIC	*Application Specific Integrated Circuits*
FPGA	*Field Programmable Gate Arrays*
DPID	*Data Path ID*
D-ITG	*Distributed Internet Traffic Generator*
TIC	*Software Defined Networks*
AN	*Active Networks*
ATM	*Cumulative Distribution Function*
DCAM	*Devolved Control of ATM Networks*
GSMP	*General Switch Management Protocol*
ForCES	*Forwarding and Control Element Separa- tion*
RPC	*Routing Control Platform*
PCE	*Path Computation Element*
BGP	*Border Gateway Protocol*
API	*Application Programming Interface*
HUB	*Cumulative Distribution Function*
QoS	*Quality of Service*
NAT	*Network Address Translation*
MAC	*Media Access Control*
RAM	*Random Access Memory*

SUMÁRIO

1 Introdução 13
1.1 Motivação 13
1.2 Objetivos 16
1.3 Contribuições 16
1.4 O que não foi abordado 18
1.5 Organização da Dissertação 19

2 Redes definidas por Software 20
2.1 Definição 20
2.2 Linha do tempo 20
2.3 Arquitetura SDN 23
2.3.1 Controlador 23
2.3.2 API *Northbound* 25
2.3.3 API *Southbound* 26
2.3.4 Hospedeiros 26
2.3.5 Elementos ativos 26
2.3.6 Componentes 27
2.4 Openflow 27
2.5 Exemplos de aplicações práticas em SDN 27
2.5.1 Gerenciamento de redes empresariais 28
2.5.2 Economia de energia em SDN 28
2.5.3 Balanceamento de carga 28

3 Problema 30
3.1 Abordagens de eficiência energética 30
3.1.1 Taxonomia de [Bolla et al., 2010]. 30
3.1.2 Taxonomia de [Bianzino et al., 2012]. 31
3.1.3 Escopo de arquitetura 33
3.2 Definição do problema 34
3.3 Cenário considerado 34
3.3.1 Modelo de consumo de energia 35

4 Proposta 37
4.1 Modelo da rede 42
4.1.1 Classe Nó 43
4.1.2 Classe Hospedeiro 44

4.1.3 Classe Grafo . . . 44
4.1.3.1 Método de descoberta de caminhos . . . 44
4.1.3.2 Método de análise de caminhos . . . 45
4.1.3.3 Método decisor de caminho . . . 45
4.1.3.4 Método de cálculo de consumo . . . 46
4.2 Roteamento Ativo . . . 46
4.2.1 Componente POX - gerente de caminhos . . . 46
4.2.1.1 Algoritmo de redirecionamento . . . 48
4.3 Monitoramento ativo . . . 49
4.3.1 Componente POX - Monitor . . . 53
4.3.2 Estados dos enlaces de uma rede . . . 54

5 Avaliação . . . 57
5.1 Metodologia de avaliação . . . 57
5.2 Cenário estático . . . 58
5.3 Cenário dinâmico . . . 59
5.4 Cenário realista . . . 62
5.4.1 Enlaces homogêneos e heterogêneos . . . 63
5.4.2 Comparativo entre cenários homogêneos e cenários heterogêneos . . . 63
5.4.3 Comparativo entre nossa proposta e o proposto por [Markiewicz et al., 2014] . . . 66

6 Trabalhos relacionados . . . 69

7 Conclusões . . . 72
7.1 Trabalhos futuros . . . 72

REFERÊNCIAS . . . 73

A Classe Nó . . . 77

B Classe Host . . . 80

C Classe Grafo . . . 82

D Método de cálculo de consumo . . . 84

E Função que captura eventos do componente Discovery . . . 85

F Função que captura eventos do componente host-event . . . 86

G Registro de eventos no núcleo do POX . . . 87

H **Função que captura eventos de estatísticas de portas** **88**

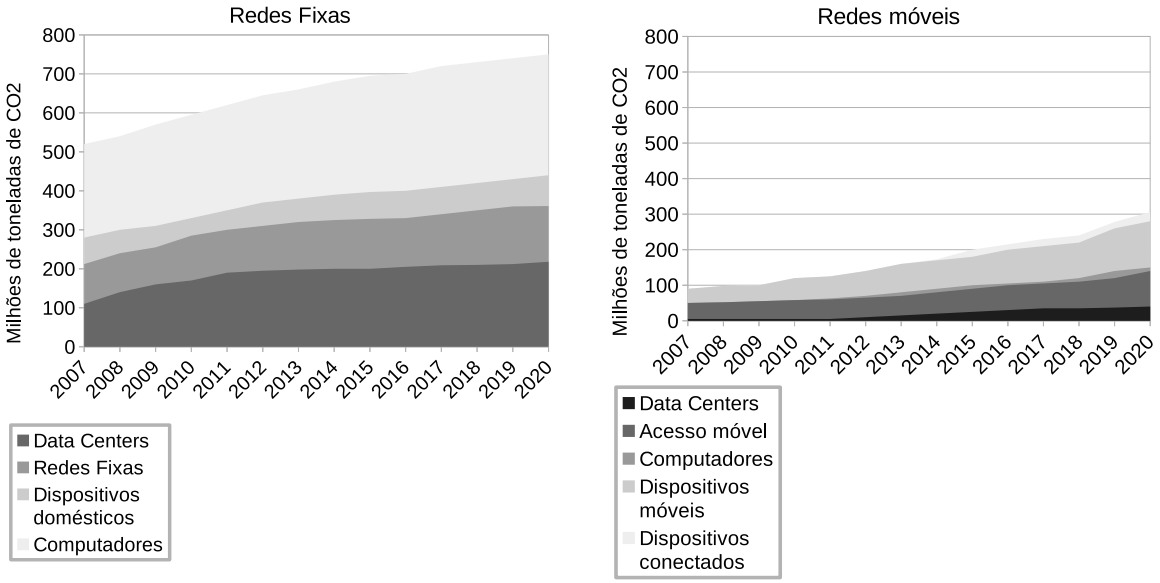
I **Publicação do evento intermedEvent** **89**

J **Algoritmo de redirecionamento** **90**

1 Introdução

1.1 Motivação

Em um mundo globalizado, onde o acesso às informações e a necessidade de comunicação estão cada vez maiores, o consumo de energia das tecnologias de comunicação e informação se torna cada vez mais significativo em termos de consumo global de energia. Dados consolidados de 2008 apontam que as tecnologias de comunicação e informação contribuem em 8% para o consumo mundial de energia e dados projetados apontam que, em 2020, 14% do consumo de energia global será atribuído às tecnologias de comunicação e informação [Markiewicz et al., 2014]. Além disso, as tecnologias de informação e comunicação contribuem para o aumento das emissões de dióxido de carbono, pois uma parcela da energia consumida é proveniente de matrizes energéticas não renováveis. Segundo [Ericson, 2016], até 2020 estima-se que as redes cabeadas serão responsáveis por 1,4% das emissões globais de dióxido de carbono, correspondendo a mais de 700 milhões de toneladas de CO2 como observado na Figura 1(a) e as redes móveis serão responsáveis por 0,5% das emissões, correspondendo a mais de 300 milhões de toneladas de CO2 conforme observado na Figura 1(b).



(a) Redes Fixas

(b) Redes móveis

Figura 1 – Emissões globais das TICs [Ericson, 2016]

Entre alguns fatores, as redes de computadores contribuem de forma relevante com o consumo global de energia, pois elas em muitos casos são superdimensionadas para atender mais que às demandas de pico de tráfego [Bianzino et al., 2010] e permanecem ligadas vinte quatro horas por dia, sete dias da semana e claramente não possuem consumo proporcional à carga de tráfego, pelo contrário, o consumo permanece constante

resultando em um grande desperdício de energia [Markiewicz et al., 2014]. Outro fator que podemos considerar está relacionado ao consumo das memórias TCAM (*Ternary content addressable memory*), muito utilizadas nos comutadores modernos, pois são vorazes consumidoras de energia elétrica [Giroire et al., 2014].

Segundo [Rodrigues, 2016], à medida que cresce o número de dispositivos conectados, as redes de computadores tendem a priorizar o alto desempenho e a alta disponibilidade, possuindo a capacidade de suportar períodos de pico de carga. Como nem sempre a rede será utilizada em sua máxima capacidade, haverá momentos em que a carga de utilização será baixa. Entretanto, o consumo permanecerá praticamente constante, pois o gasto de energia das redes de computadores tradicionais não apresenta variação relevante conforme se altera o nível de carga de tráfego. Momentos de baixa atividade da rede proporcionará uma oportunidade de aplicar estratégias de otimização do consumo de energia em cenários em que este não é proporcional à carga da rede. A primeira parte da Figura 2, apresenta o consumo de energia ao longo do tempo, sem qualquer mecanismo de otimização do consumo de energia. A segunda parte da figura apresenta o consumo de energia ao longo do tempo, porém agora possuindo um mecanismo para economizar energia. A Figura 2 t1, apresenta o tempo para ajuste da energia consumida de forma a atender a nova demanda de tráfego. O tempo de ajuste é necessário para que a partir da detecção de uma mudança de carga na rede, seja concretizada a reconfiguração necessária para atender à nova demanda. Os tempos t3, t4, t5 e t6, apresentam um momento em que a carga da rede diminui e o mecanismo reage desligando uma parte da infraestrutura, poupando energia. Finalmente o momento t7 também ilustra o tempo para ajuste para a adequação da energia consumida em relação à nova demanda por tráfego.

Quando aplicamos um mecanismo capaz de reagir dinamicamente às mudanças da carga na rede, alterando a velocidade das portas e/ou desligando portas e comutadores, será possível reduzir o consumo em momentos de baixa atividade. Por outro lado, caso a carga da rede aumente, o mecanismo deverá ser capaz de reagir ligando novos comutadores e portas, aumentando as velocidades dos enlaces, ou seja, reconfigurando a rede de modo a manter a qualidade de serviço e os requisitos de confiabilidade. Todo este esforço tem como objetivo tornar o consumo de energia da rede dinâmico em relação à carga de tráfego.

Na figura 3, podemos observar um padrão de tráfego semanal de um *data center* global interconectando WANs. Fica evidente que a carga de tráfego não apresenta um comportamento constante. Podemos notar uma grande variação do tráfego diurno para o tráfego noturno. Deste modo, os gerentes devem superdimensionar as redes para atender a este pico de demanda. Durante as noites, a maior parte dos comutadores permanecem ociosos e consumindo uma quantidade de energia similar ao consumo diurno, mesmo com uma carga de tráfego consideravelmente menor. Deste modo, é possível criar novas tecnologias que explorem características como, a variação de tráfego periódico, superdi-

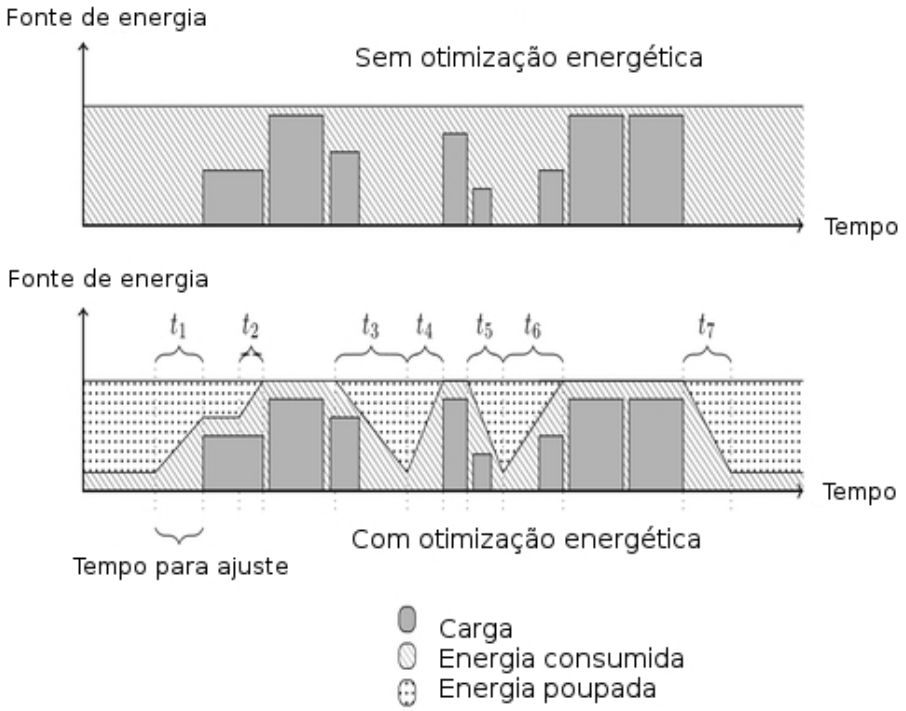


Figura 2 – Otimização energética vs não otimização [Rodrigues, 2016]

mensionamento de recursos e nível de consumo independente de sua utilização a fim de possibilitar às redes do futuro um consumo mais proporcional à carga do que as redes tradicionais.

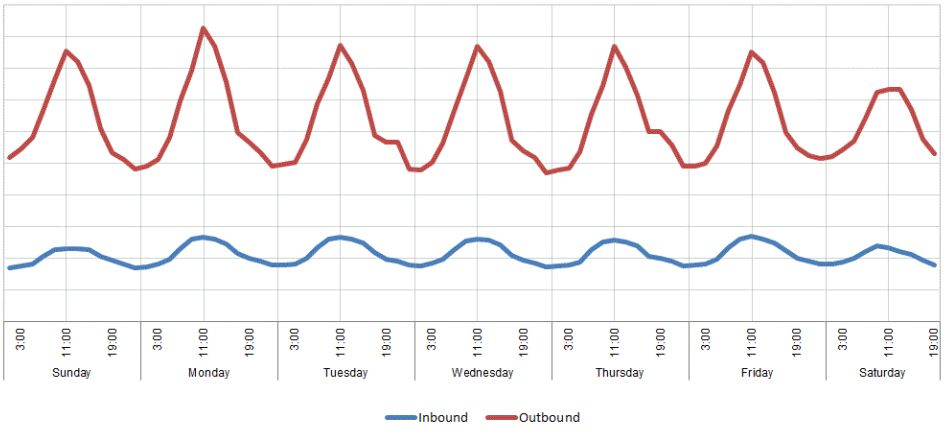


Figura 3 – Padrão de tráfego semanal global [Markiewicz et al., 2014]

.

Dessa forma, observamos a importância de iniciativas que de alguma forma tentem minimizar o consumo de energia em redes de comunicação.

1.2 Objetivos

Neste trabalho mostraremos que é possível trazer para as redes de computadores uma maior proporcionalidade no consumo de energia em relação à taxa de utilização da mesma. Veremos que, ao concentrar as demandas da rede em uma menor quantidade de caminhos e desligar os comutadores das alternativas não utilizadas, é possível reduzir o consumo de energia em relação a uma rede tradicional.

Para alcançar nosso objetivo de reduzir o consumo de energia em redes de computadores, criamos um mecanismo dinâmico que explora as características do SDN para manipular as rotas dos fluxos. Desta maneira, a maior parte dos fluxos ativos da rede serão encaminhados em uma quantidade mínima de caminhos fim a fim, através de reconfiguração dinâmica da rede.

Nossa principal contribuição foi a implementação de um mecanismo reativo e dinâmico, onde os fluxos primeiramente são alocados em um caminho não marcado como sobrecarregado. Após esta acomodação do fluxo no caminho, será realizado o monitoramento contínuo da rede. Caso o valor limiar indicativo de sobrecarga seja ultrapassado, esse novo fluxo será redirecionado para um caminho alternativo, caso este exista. De forma similar, monitoramos as alternativas de caminhos ligados, buscando um valor limiar indicativo de baixa carga de utilização de um enlace. Neste caso, de forma dinâmica redirecionamos os fluxos dos elementos de rede para esvaziar suas tabelas e, logo após desligá-los. Ou seja, não só o número de elementos de rede ligados varia dinamicamente, mas também os fluxos são redirecionados dinamicamente para concentrar o tráfego na menor quantidade de caminhos possível.

Em nossa abordagem, optamos por partir do princípio que não conhecemos de forma antecipada como será o comportamento dos fluxos e não realizamos otimizações baseadas nas matrizes de tráfego de forma a prever o comportamento futuro na rede. Ou seja, nós criamos um mecanismo que de forma autônoma redireciona dinamicamente os fluxos de caminhos saturados para caminhos mais livres, independente do nível de carga na rede.

1.3 Contribuições

Em nossa abordagem, buscamos implementar uma parte das necessidades de uma utilização em um cenário realista. Em linhas gerais, muitas das abordagens, como em [Markiewicz et al., 2014, Mahadevan et al., 2009b] focam em criar um algoritmo guloso que consiste em obter a menor quantidade possível de elementos de rede ligados respeitando que as demandas de tráfego sejam alocadas em caminhos com capacidade suficiente. Com base na topologia, nas demandas de tráfego obtidas através da matriz de tráfego originárias dos nós de acesso e nas listas de nós ligados, os algoritmos calculam qual a

menor quantidade de nós possível serão necessários para atender às demandas atuais de tráfego. Além do mais, os algoritmos iteram entre todos os pares de nós ativos para encontrar o melhor caminho dentre os caminhos que gerarão o menor acréscimo de consumo de energia.

Abordagens utilizando simulação, que nós podemos encontrar em [Markiewicz et al., 2014, Sasaki et al., 2015, Giroire et al., 2014], ignoram aspectos práticos, como o funcionamento do controlador, a linguagem de programação do controlador, as mensagens *Openflow* e etc.

Outro aspecto que podemos observar nas propostas analisadas, é a ausência de um mecanismo para redirecionar o tráfego de caminhos com baixa carga como em [Markiewicz et al., 2014]. Nestes casos, a simulação já possui ciência das demandas previamente e desta forma é possível evitar que um caminho se torne sobrecarregado.

Em nossa proposta, utilizamos um emulador para torna-la mais próxima da realidade. Além do mais não precisamos conhecer previamente como será a matriz de tráfego. Chamamos nossa estratégia de estratégia dinâmica, pois ela atua conforme a variação natural do tráfego no decorrer do cenário. Nossa estratégia funciona através do monitoramento periódico dos enlaces, e conforme o controlador aloca os novos fluxos, monitoramos a rede em busca de enlaces sobrecarregados ou subutilizados.

Para detectar uma sobrecarga ou uma sub-utilização, estabelecemos faixas de porcentagem de utilização dos comutadores em relação à sua capacidade máxima. A estas faixas demos o nome de estados de um enlace. Deste modo, caso o controlador detecte uma mudança de estado de um enlace para estados de sobrecarga ou baixa carga, ele gerará um evento que culminará na tentativa de redirecionamento de fluxos. No primeiro caso, o redirecionamento terá como objetivo aliviar a carga do link e no segundo caso, o redirecionamento terá como objetivo retirar os fluxos do comutador iterativamente até que todos os fluxos do comutador sejam transferidos. Quando o número de fluxos de um comutador atingir zero fluxos, o controlador irá desliga-lo.

Em [Heller et al., 2010], os autores adotaram de forma similar ao nosso trabalho a estratégia de desligar comutadores não utilizados ou subutilizados na rede. Porém a solução adotada pelos autores, diferentemente da nossa proposta, adota uma estratégia pró-ativa, ou seja, ela possui informações de como serão os tráfegos da rede com base na matriz de tráfego, na topologia e em um modelo de consumo de cada nó e, por isso, elas podem otimizar a alocação dos tráfegos e reagir a uma mudança no comportamento da rede. Segundo os autores, ao assumir que a estratégia *Elastictree* possui conhecimento prévio dos tráfegos na rede, seria inviável sua utilização em uma implantação real. Por outro lado, diferentemente de nosso trabalho, a abordagem de [Heller et al., 2010] considera manter uma certa redundância ativa para obter tolerância a falhas.

Nossa abordagem se diferencia das principais abordagens pois ela trabalha de forma reativa. Em outras palavras, não conhecemos previamente qual o tamanho da largura de banda será demandada por um determinado fluxo. Primeiramente alocamos um fluxo em um caminho e após o estabelecimento da comunicação entre os hospedeiros, monitoramos a rede para descobrir se o novo fluxo ou qualquer outro fluxo presente na rede provocou uma sobrecarga no enlace, e caso seja detectada uma sobrecarga, nosso algoritmo reage redirecionando os fluxos do enlace sobrecarregado. Este tipo de abordagem reativa depende do monitoramento constante e do redirecionamento dinâmico de fluxos para evitar uma sobrecarga comprometedora.

Focamos, em momentos de pouca atividade na rede (tráfego noturno), obter uma árvore geradora mínima exibido na Figura 3. No entanto, como trabalho futuro, teremos que considerar um nível mínimo de tolerância a falhas para torna-lo seguro em ambientes reais.

Avaliamos nosso mecanismo em um cenário realista considerando três níveis de carga de tráfego (baixa, média e alta). Comparamos nosso mecanismo com a implementação do pseudo código proposto por [Markiewicz et al., 2014] em cada nível de carga e comparamos os resultados. Podemos destacar que obtemos em média 46,01% de economia de energia máxima em períodos de baixa carga de tráfego em relação a uma rede tradicional e, em média, obtemos 2,4026% de economia a mais comparado ao código proposto por [Markiewicz et al., 2014] em carga de tráfego média.

1.4 O que não foi abordado

Não abordamos o problema de otimizar o processo de alocação das regras em comutadores *Openflow*. O artigo [Giroire et al., 2014] apresenta uma abordagem em que se deseja evitar que as alocações das regras nas tabelas de fluxos dos comutadores sejam ineficientes e redundantes. Como a quantidade total de regras que podem ser instaladas nas tabelas de fluxos de um comutador são limitadas, uma alocação ineficiente dessas regras pode ocasionar uma sobrecarga de entradas na tabela de fluxos e, consequentemente, o comportamento da rede pode não ser o esperado, o que pode causar congestionamentos, afetar a qualidade da rede e aumentar o consumo de energia.

Optamos por utilizar uma busca em profundidade para encontrar caminhos com o menor número de nós possível. Esta opção faz sentido quando é vista sob a ótica da economia de energia, pois quanto menos nós utilizamos, mais nós serão candidatos ao desligamento. Porém em um cenário real, não será sempre o caminho com o menor número de saltos que será o mais vantajoso em termos de qualidade de serviço. Ou seja, em um cenário real teremos que considerar os custos de cada nó da topologia.

Em nosso trabalho existem soluções implementadas em que monitoramos toda a

topologia e detectamos quando um enlace deixa a rede, porém não consideramos em nosso cenário a possibilidade de redirecionar fluxos em caso de falhas na rede.

Em nosso mecanismo para economizar energia elétrica precisamos que os comutadores sejam desligados ou entrem em estado de espera, porém não implementamos o efetivo desligamento. Entretanto desligamos logicamente cada comutador através do modelo da topologia da rede. No modelo da rede, cada nó é um objeto, possuindo atributos do consumo de cada porta, consumo geral, nível de carga de tráfego, estado (ligado ou desligado) e etc.

Apesar de levarmos em consideração o consumo das portas individualmente, optamos por não utilizar a estratégia de intercâmbio entre as taxas de dados *Ethernet* (10 Mbps, 100 Mbps, 1 Gbps, etc), pois representam pouca economia considerando a economia gerada pelo desligamento completo do comutador.

1.5 Organização da Dissertação

O restante deste trabalho está organizado da seguinte maneira. Primeiro, no Capítulo 3, apresentamos os fundamentos de redes verdes e o problema abordado. Então, no Capítulo 4, detalhamos o funcionamento do nosso mecanismo de gerenciamento de tráfego dinâmico com ciência do consumo de energia. No Capítulo 5, primeiro descrevemos a avaliação do mecanismo e, em seguida, os resultados da nossa simulação. No Capítulo 6 apresentamos uma revisão da literatura sobre economia de energia em redes utilizando redes definidas por *software*. Finalmente, nossas conclusões são apresentadas no Capítulo 7.

2 Redes definidas por Software

2.1 Definição

Redes Definidas por *software* são um novo paradigma de rede, surgido a partir da necessidade de solucionar as dificuldades encontradas pelos pesquisadores de rede ao testar novos protocolos e novas ideias nas redes tradicionais, pois estas não separavam o tráfego de produção do tráfego experimental, trazendo riscos para a realização dos experimentos. Segundo [McKeown et al., 2008a], devido às restrições das redes tradicionais, existiam muitas novas ideias da comunidade de pesquisadores que não eram testadas em redes de produção devido as limitações e riscos, levando a ideia de que as redes de computadores tradicionais estavam "ossificadas".

Segundo [Costa, 2016], as Redes Definidas por *Software* (SDN) definem a forma como será realizado o encaminhamento pacotes, mudando a forma original de encaminhamento das redes tradicionais. As Redes Definidas por *software* separam o plano de dados e o plano de controle. O objetivo desta separação é possibilitar que o plano de controle, antes distribuído entre os ativos de rede, fique centralizado em um *software* denominado controlador.

Segundo [McKeown et al., 2008a], utilizando as Redes Definidas por *Software*, podemos facilitar a inovação e a evolução da rede, possibilitando o desenvolvimento de novos serviços e a execução de experimentos inovadores. Além do mais, as Redes Definidas por *Software* não afetam o tráfego de produção, possibilitando a reformulação de diversos serviços típicos de rede, tornando-os mais flexíveis.

2.2 Linha do tempo

Podemos observar na Figura 4 uma linha do tempo contendo as principais tecnologias que propiciaram a evolução e desenvolvimento do paradigma SDN.

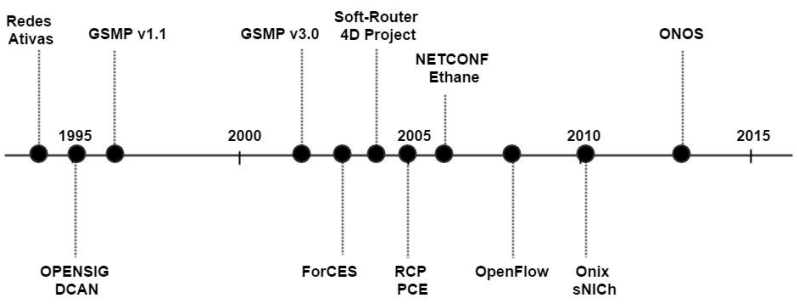


Figura 4 – Linha cronológica de tecnologias que ajudaram a desenvolver paradigma SDN [Costa, 2016]

Apesar de atualmente o termo *Software-Defined Networks* (SDN) ser amplamente difundido entre especialistas na área, até 2009 ele não havia sido utilizado. Segundo [Zilberman et al., 2015], o termo foi utilizado pela primeira vez à menos de dez anos no artigo tecnológico de [Greene, 2009]. As Redes Definidas por *Software* evoluíram ao longo do tempo mesmo antes da definição do termo. Neste período (meados dos anos 90) as redes trabalhavam com os planos de controle e plano de dados contidos em um mesmo elemento comutador. Esta disposição proporcionava que a "inteligência" da rede estivesse distribuída entre os seus elementos de rede, dificultando que uma modificação tivesse abrangência em toda a rede. Estas limitações das redes tradicionais diminuem a margem para que os pesquisadores proponham modificações no seu funcionamento.

Conforme [Costa, 2016], estas características das redes tradicionais restringem a capacidade de configuração e extensão, tornando-se um gargalo importante. De forma a solucionar estas limitações, várias propostas surgiram para flexibilizar as redes. Uma das primeiras propostas de flexibilização das redes foi a das Redes Ativas (*Active Networks - AN*), formulada entre 1994 e 1996 e publicada no trabalho de [Tennenhouse et al., 1997]. A iniciativa de redes ativas proporcionou uma forma de programação na qual elementos ativos realizassem determinados cálculos sobre os pacotes, permitindo a modificação do conteúdo dos mesmos [Tennenhouse et al., 1997, Kreutz et al., 2015].

Segundo [Costa, 2016], as Redes Ativas contribuíram para a criação das Redes Definidas por *Software*, pois foram pioneiras na utilização de programação da rede facilitando a criação de novos serviços em redes de produção. As Redes Ativas não conseguiram a adesão em massa, pois haviam problemas de segurança e desempenho, e os roteadores modificados possuíam uma baixa capacidade de processamento de pacotes, quando comparado com roteadores tradicionais [MACEDO et al., 2015, NUNES et al., 2014].

Em 1995 a ATM desenvolveu o DCAN (*Devolved Control of ATM Networks*). O projeto proporcionou a criação de uma infraestrutura para controlar e gerenciar as redes ATM. O DCAN levou em consideração a separação de planos, criando um protocolo básico entre o gerenciador e a rede [DCAN, 1995, NUNES et al., 2014].

Ainda em 1995 foi criado um grupo de trabalho chamado *OpenSignaling* (OPENSIG), com o objetivo de criar um debate em torno de ideias que tornassem as redes mais abertas e flexíveis. Como resultado deste debate surgiu um novo protocolo, o GSMP (*General Switch Management Protocol*). O GSMP permitia que o controlador gerenciasse portas, estatísticas de conexões, reserva de recursos, também possibilitando o estabelecimento e liberação de conexões em um determinado comutador [NUNES et al., 2014, GSMP, 2002]. Com a criação do protocolo, de forma rudimentar pela primeira vez havia um controle de equipamentos ativos da rede, através da separação dos planos de controle e encaminhamento.

Em 2003 surgiu a arquitetura ForCES (*Forwarding and Control Element Separation*). A arquitetura definiu o que era um elemento de encaminhamento e o elemento de controle. Conforme o próprio nome da arquitetura sugere, foi realizada a separação do plano de controle do plano de dados. ForCES trabalha utilizando qualquer topologia, permitindo um grande número de comutadores e elementos de controle, além de oferecer uma interface aberta programável para cada plano [MACEDO et al., 2015, Doria et al., 2010].

Nos anos seguintes sugiram novas iniciativas, como *Routing Control Platform* (RCP) e a *Soft-Router*, assim como o protocolo *Path Computation Element* (PCE), que contribuíram para que o controle fosse lógico e centralizado [Feamster et al., 2014]. A arquitetura *Soft-Router* permite que o controlador centralizado insira regras na tabela de fluxos do plano de dados, utilizando API da ForCES [Lakshman et al., 2004]. A arquitetura RPC utiliza o protocolo padrão BGP com o objetivo de instalar regras na tabela de fluxo dos roteadores legados [Caesar et al., 2005].

Já nos anos 2000, surgiram um conjunto de esforços relevantes para ampliar as capacidades de programação da rede e separação dos planos de controle e plano de dados. Como resultado surgiu o projeto *Ethane*, publicado no trabalho [Casado et al., 2007]. Os autores criaram uma arquitetura de rede corporativa que utilizava um controlador logicamente centralizado para atuar como gerenciador de políticas de acesso e de encaminhamento e um comutador *Ethane* que fornece uma tabela de fluxo e um canal de comunicação seguro [NUNES et al., 2014]. Os comutadores *Ethane* possuíam apenas a função de encaminhamento, sendo dependentes do controlador para a tomada de decisão de como executar o encaminhamento [Feamster et al., 2014]. Ou seja, podemos notar que o projeto *Ethane* é muito próximo do conceito das Redes Definidas por *Software* atuais.

Conforme [Costa, 2016], podemos considerar a aplicação de segurança criada para controle de acesso na rede *Ethane* como sendo a primeira aplicação SDN criada na história. O projeto *Ethane* foi um divisor de águas do paradigma SDN, pois serviu de inspiração para a criação do protocolo mais utilizado atualmente para a comunicação entre controlador e comutadores: o protocolo *Openflow*.

O protocolo *Openflow* surgiu na universidade de *Stanford* e foi publicado em 2008 no trabalho [McKeown et al., 2008b], seguindo os conceitos básicos contidos no trabalho *Ethane* para criar um protocolo que vá além do uso acadêmico, sendo também viável para utilização em comutadores comerciais. O protocolo *Openflow* contribuiu para que a programabilidade de redes e separação entre planos de controle e planos de dados passasse para um nível prático inédito no paradigma SDN até então. A partir do histórico do paradigma SDN apresentado, podemos observar que o paradigma evoluiu aos poucos e somente no final dos anos 2000 é que surgiu um trabalho que proporcionou uma adesão, tanto de pesquisadores como dos fabricantes [Kreutz et al., 2015].

2.3 Arquitetura SDN

Em SDN podemos categorizar os elementos ativos da rede em duas categorias principais. O controlador lógico centralizado e os elementos de encaminhamento responsáveis por colocar em prática as decisões do controlador. Além do mais, temos outras entidades que participam tanto das redes tradicionais como das redes SDN, como computadores, dispositivos sem fio, servidores e etc. Nas próximas subseções detalharemos cada elemento da arquitetura SDN. Na Figura 5 temos os elementos típicos de uma rede SDN.

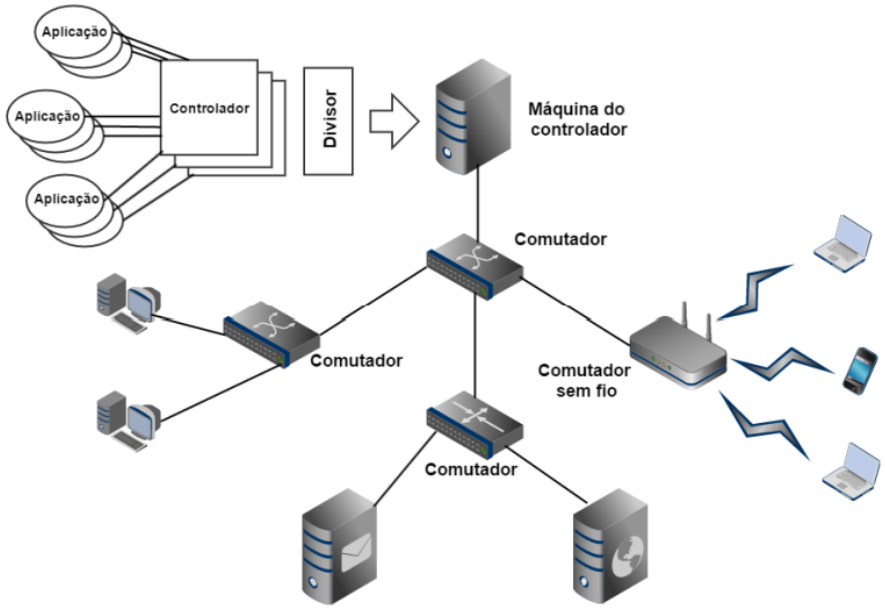


Figura 5 – Arquitetura típica de uma rede SDN [Costa, 2016]

2.3.1 Controlador

Segundo [Gude et al., 2008], podemos fazer uma analogia em que o controlador é como um sistema operacional da rede, pois fornece uma interface simplificada de programação, onde operadores de rede poderão criar seus componentes ou aplicações. Ou seja, o "sistema operacional da rede" permite a criação de aplicações de gerenciamento escritas em código de alto nível, possibilitando ao programador abstrair os recursos físicos de cada fabricante/modelo de um ativo de rede.

O controlador é um dispositivo ativo, que age para definir como cada elemento comutador irá se comportar na rede. Ele assume o papel do plano de controle, separado do plano de dados ainda presentes nos elementos comutadores. Podemos visualizar esta nova configuração através da ilustração 7. Ele pode ser um elemento único na rede ou pode ser um elemento logicamente centralizado, podendo existir vários controladores se comportando como um único controlador através da arquitetura distribuída. Como

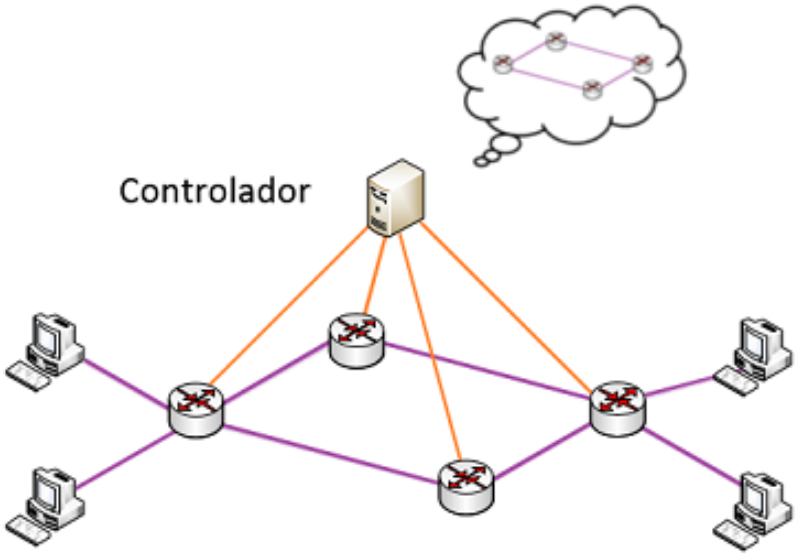


Figura 6 – Controlador SDN. [Lobato et al., 2013]

vantagens de um controlador distribuído perante um controlador único podemos citar sua maior escalabilidade, maior resiliência e maior desempenho. Ainda é possível criar redundância do plano de controle ao adicionar mais de um controlador em um mesmo elemento comutador [Costa, 2016].

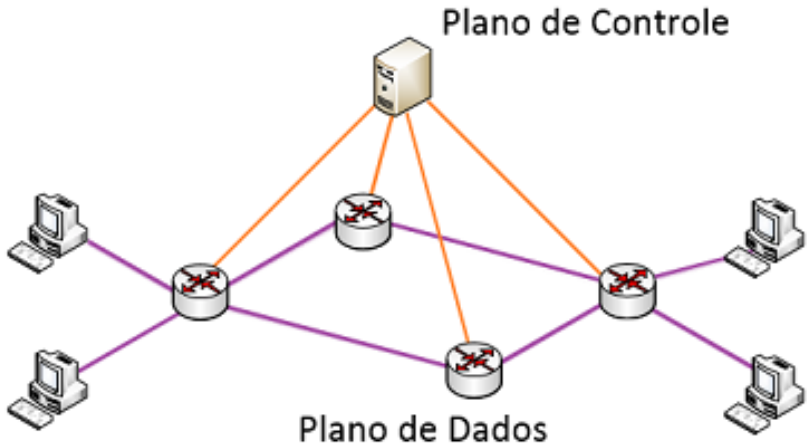


Figura 7 – Separação de planos. [Lobato et al., 2013]

O controlador utiliza APIs para se comunicar com os elementos comutadores. O controlador envia suas mensagens seguindo um padrão, sendo então recebidas pelos elementos comutadores através de um canal de controle seguro. Esta comunicação utiliza uma API de baixo nível, porém o controlador abstrai os detalhes de baixo nível para os programadores de rede, exportando uma interface de programação para os mesmos. [MACEDO et al., 2015].

Os controladores, por concentrarem a "inteligência" da rede, também possibilitam a existência de uma visão centralizada da topologia, como podemos observar na Figura 6, oferecendo informações preciosas para o desenvolvimento de aplicações que precisem deste tipo de dado para a tomada de decisão sobre como o sistema deve atuar [Costa, 2016].

Ainda mantendo o relacionamento com o controlador temos as APIs *Northbound* e *Southbound*. Elas tem a função de estabelecer camadas de comunicação entre o controlador e os outros elementos presentes na arquitetura SDN. Na Figura 8, podemos visualizar a relação de cada API com seu respectivo elemento da arquitetura.

Nas próximas subseções veremos com mais detalhes a função de cada interface mais detalhadamente.

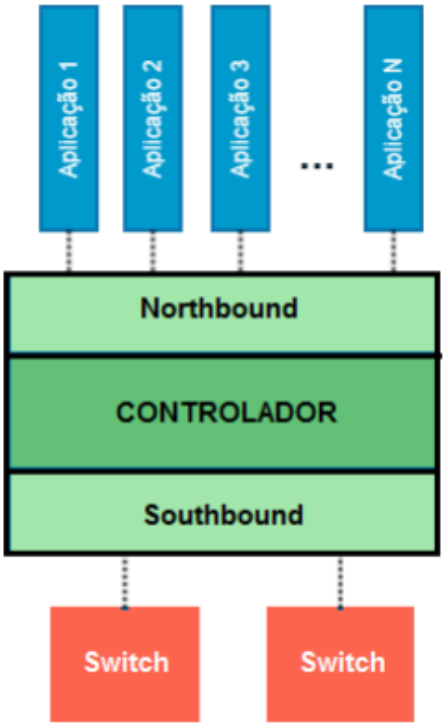


Figura 8 – Relação entre o controlador e as interfaces *Northbound* e *Southbound*. [Costa, 2016]

2.3.2 API *Northbound*

O controlador precisa se comunicar com as aplicações desenvolvidas de modo a trabalharem de forma conjunta. Para suprir esta necessidade existem as APIs *Northbound*, que são responsáveis por ser a ponte entre a camada de aplicação e a camada de controle. As APIs facilitam a programação da rede, pois permitem que programas desenvolvidos em linguagens de alto nível as controlem. O termo "norte" tem relação com a direção que se dá a comunicação: Controlador-aplicação, onde a aplicação é considerada o nível mais alto da arquitetura. As APIs *Northbound* não possuem uma padronização quanto a um padrão

que seja amplamente utilizado, sendo que cada controlador (Nox, Trema, *Floodlight*), possui sua própria especificação para a API *Northbound* [Costa, 2016]. No entanto, foi elaborado o projeto *OpenDayLight* que tem como objetivo criar um padrão para as APIs *Northbound*, permitindo reutilizar os programas de controle [MACEDO et al., 2015].

2.3.3 API *Southbound*

Ao contrário das APIs *Northbound* que trabalham em um nível de abstração alto, o nível de abstração das APIs *Southbound* é baixo, pois foi concebido para manipular as APIs de baixo nível dos elemento comutadores. Porém, o desenvolvedor de aplicativos para a rede não precisa se preocupar com os detalhes da comunicação entre o controlador e os elementos comutadores, pois os protocolos que realizam a comunicação são geridos pelo controlador. Por realizar uma ponte de comunicação entre o controlador e os elementos mais baixos em nível de abstração da arquitetura, surgiu a expressão "sul", devido a esta comunicação entre controlador-comutador [Costa, 2016].

2.3.4 Hospedeiros

Os hospedeiros são os equipamentos que utilizam a arquitetura SDN, no entanto, não sofrem as ações do controlador, pois não fazem parte dos elementos principais da arquitetura. Podemos listar como hospedeiros os celulares, computadores, sensores, televisores e etc

2.3.5 Elementos ativos

Consideramos elementos ativos aqueles elementos de comutação que sofrem a ação direta do controlador. Nas redes tradicionais, além da função de encaminhamento de pacotes, estes elementos ainda concentravam a função de controle. Ou seja, cada elemento ativo possuía a capacidade de decidir sobre como seria realizado o encaminhamento de pacotes. Como consequência, a "inteligência" da rede se encontrava distribuída, não possuindo um elemento central que poderia decidir o comportamento da rede de forma global.

Nas Redes Definidas por *Software*, a camada de controle foi extraída dos elementos comutadores e foi concentrada no controlador. Assim, caso um pacote chegue em um comutador e o mesmo não possua uma regra de encaminhamento em sua tabela de fluxos, este não terá autonomia para decidir uma ação para este pacote. Deste modo, caso este cenário persista, o comutador irá consultar o controlador em busca de uma regra para este tipo de pacote. Caso encontre, o controlador irá instalar a regra no comutador solicitante e só a partir desta ação o comutador poderá exercer sua função de encaminhamento de pacotes.

2.3.6 Componentes

Os componentes são as aplicações que executam sobre os controladores. É por meio das aplicações que os desenvolvedores de rede determinam como será o comportamento de toda a rede. Os desenvolvedores não precisam conhecer os detalhes de implementação de cada comando necessário para a comunicação do controlador com comutadores de diferentes modelos e marcas. Por este motivo, o desenvolvimento de componentes é conduzido utilizando linguagens de alto nível, de modo a simplificar o processo para o desenvolvedor. Dentre as possibilidades de criação de componentes, temos desde aplicações que simulam um simples *HUB* até aplicações complexas que se utilizem da visão global da rede para decidir caminhos onde determinados tipos de pacotes irão trafegar.

2.4 Openflow

O principal protocolo utilizado para realizar a comunicação através do plano de controle entre controlador e comutador é o protocolo aberto denominado *Openflow*. Ele foi desenvolvido por [McKeown et al., 2008a] em 2008 e os autores tinham a ideia de que o protocolo *Openflow* seria uma ferramenta para que pesquisadores testassem protocolos experimentais em suas redes de uso diário [McKeown et al., 2008a].

Em [McKeown et al., 2008a], os autores observaram que os comutadores *Ethernet* tradicionais possuíam uma série de funções comuns que funcionavam na maioria dos dispositivos presentes no mercado. Também notaram que esses possuíam tabelas de fluxo construídas com memórias TCAM que eram usadas para implementar Qos, NAT, *Firewalls* e coleta de estatísticas. Aproveitando estas características, os autores criaram o padrão de forma a ser possível programar a tabela de fluxos de equipamentos de diferentes fabricantes de comutadores e roteadores.

Após a criação do protocolo *Openflow*, houve uma padronização da maneira de como o controlador e os elementos comutadores se comunicam. Segundo [Costa, 2016], o controlador é desenvolvido em uma linguagem de programação em alto nível, e a partir da adoção do protocolo *Openflow* , o controlador é capaz de enviar para o comutador um conjunto de operações na forma de comandos, que serão recebidos e compreendidos graças a padronização. Os comandos que chegam nos comutadores são capazes de inserir, modificar ou remover entradas na tabela de fluxos.

2.5 Exemplos de aplicações práticas em SDN

As Redes Definidas por *Software*, foram pensadas para vencer as limitações impostas pelas redes tradicionais. As redes legadas se tornaram "ossificadas", pois com o tempo elas limitaram as pesquisas e as inovações. O paradigma SDN permite que o tráfego experimental e o tráfego de produção sejam separados, possibilitando a criação de inovações

em ambientes reais. Outra distinção em relação às redes tradicionais é a possibilidade do controlador possuir a visão centralizada da rede. Ao concentrar o plano de controle em apenas um elemento logicamente centralizado, os desenvolvedores de rede podem criar componentes que alterem o comportamento de todos os elementos comutadores.

Nas próximas subseções veremos alguns exemplos de aplicações práticas do paradigma SDN.

2.5.1 Gerenciamento de redes empresariais

Segundo [Costa, 2016], nas redes tradicionais, a maioria das políticas de gerenciamento de redes tem ação limitada a apenas um elemento. Então, à medida em que cresce o número de nós de uma rede, também aumenta a complexidade de manter as configurações de rede consistentes. Ao aderir ao paradigma SDN é possível ter a visão global da rede, simplificando o monitoramento de fluxos e configurações [JARSCHEL et al., 2014]. Considerando os benefícios das Redes Definidas por *Software*, podemos obter políticas de gerenciamento adaptadas dinamicamente e de maneira automática conforme varia o estado da rede [Costa, 2016].

2.5.2 Economia de energia em SDN

Cada vez mais pesquisadores tem se preocupado em construir sistemas que sejam mais eficientes no consumo de energia elétrica. Nos grandes data-centers e nas empresas, a infraestrutura de rede atinge dimensões consideráveis, tornando a busca pela eficiência uma questão de redução de custos e redução de impactos ambientais. Para enfrentar este desafio, pesquisadores tem buscado utilizar SDN para aumentar a eficiência energética das redes de computadores.

Uma parte das contribuições nesta área pretende utilizar a visão global oferecida pelas Redes Definidas por *Software*, para identificar a ociosidade da mesma e concentrar os fluxos em uma sub-rede, com o objetivo de desligar ou reduzir as taxas de transferência dos elementos que não fizeram parte desta sub-rede. Segundo [Guedes et al., 2012], além da redução da taxa de transmissão e desligamento por completo de uma parte da rede, é possível implantar pontos de controle para capturar pacotes indesejados evitando que eles atinjam máquinas que trabalhem com o modelo *wake-on-lan*, prevenindo um despertar desnecessário. A partir destes esforços, é possível tornar as redes de computadores mais proporcionais no consumo de energia em relação à sua carga de utilização.

2.5.3 Balanceamento de carga

O balanceamento de carga é um serviço muito requisitado nas empresas e *data-centers*. Nas redes tradicionais, geralmente são utilizados balanceadores proprietários, que geralmente, são caros [Costa, 2016]. Com o advento das Redes Definidas por *Software*, é

possível programar um balanceador de carga. Este programa irá rodar sobre o controlador, que enviará comandos para os elementos comutadores de forma a utilizar da melhor forma possível os recursos disponíveis na rede.

3 Problema

3.1 Abordagens de eficiência energética

Atualmente existem diversas estratégias que possuem como objetivo aumentar a eficiência energética das redes de computadores. Os autores de [Rodrigues, 2016] citam duas taxonomias importantes para a classificar as abordagens de eficiência energética: A classificação de [Bolla et al., 2010] e a classificação de [Bianzino et al., 2012]. [Bolla et al., 2010] apresentam uma pesquisa dos métodos existentes para aumentar a eficiência energética em redes cabeadas. A classificação das estratégias de economia de energia de [Bolla et al., 2010] considera a economia obtida por equipamentos individuais e a classificação de [Bianzino et al., 2012] considera a economia alcançada por estratégias que envolvem mais de um elemento de rede.

3.1.1 Taxonomia de [Bolla et al., 2010].

Os autores resumiram as estratégias em três tópicos principais como podemos observar na Figura 9.

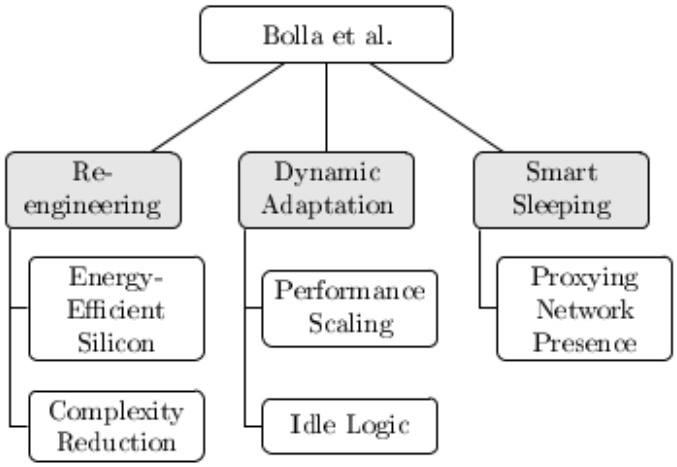


Figura 9 – Taxonomia de abordagens de eficiência energética em redes com fio [Rodrigues, 2016]

.

- Reengenharia: Segundo [Bolla et al., 2010], as abordagens de reengenharia tem como objetivo criar tecnologias para aumentar a eficiência energética para usá-las em equipamentos de rede. O principal objetivo da reengenharia é criar elementos de rede que são mais eficientes, principalmente através de novos circuitos presentes na arquitetura do equipamento de rede. Como exemplos de reengenharia para aumentar a eficiência temos as ASICs, FPGAs, processadores de pacotes, novas tecnologias de memórias e etc. Outra abordagem possível na reengenharia

segundo [Rodrigues, 2016] é a redução da complexidade do equipamento para executar softwares embarcados. A estratégia de reengenharia pode alcançar as maiores taxas de economia de energia, porém estes avanços são os mais desafiadores em termos de inovação em redes de computadores [Rodrigues, 2016];

- Adaptação dinâmica: Segundo [Bolla et al., 2010], as abordagens de adaptação dinâmica tem como objetivo modular as capacidades dos recursos internos dos dispositivos de rede como capacidades computacionais, largura de banda de enlaces e processamento de pacotes conforme varia o nível de carga do tráfego na rede, tornando o consumo elétrico do dispositivo mais proporcional à carga de tráfego do que dispositivos que não contam com mecanismos de adaptação dinâmica. Segundo [Rodrigues, 2016], estes tipos de abordagens possuem dois tipos de recursos de gerenciamento energia. Um recurso disponibilizado pelo *hardware* e outro pela lógica de inatividade;

- Modo de espera: Segundo [Bolla et al., 2010], os mecanismos de modo de espera permitem o gerenciamento de energia do equipamento de modo a desligar quase completamente ou alternar para estados de baixo consumo de energia, paralisando suas funcionalidades. Durante os períodos em que o equipamento permanece em modo de espera, além da paralisação de suas atividades, os aplicativos também são encerrados e o equipamento interrompe a conexão com a rede. Porém, antes de interromper sua operação, o equipamento transfere suas funcionalidades na rede para outro equipamento.

Apesar de parecer que o modo de espera funcione a partir de mecanismos de engenharia de tráfego, ciente do consumo de energia e da visão global da rede, ele apenas atua em um nó individualmente. De forma similar, a adaptação dinâmica e a reengenharia também possuem como escopo de atuação elementos individuais da rede.

3.1.2 Taxonomia de [Bianzino et al., 2012].

As taxonomias de [Rodrigues, 2016, Bianzino et al., 2012], nos fornece um leque de abordagens em termos de economia de energia, mais amplo do que o conjunto da taxonomia de [Bolla et al., 2010]. Em [Bianzino et al., 2012], temos além dos modos de economia abordados por [Bolla et al., 2010], temos também a consolidação de recursos e estratégias de virtualização. Segundo [Bianzino et al., 2012], a consolidação de recursos é ampla o suficiente para abordar estratégias de migrações de nós e estratégias de engenharia de tráfego com ciência do consumo de energia. [Rodrigues, 2016] apresenta a figura 10 referente ao trabalho de [Bianzino et al., 2012]:

• Consolidação de recursos: Segundo [Bianzino et al., 2012] a consolidação de recursos agrupa todas as abordagens que possuem como objetivo proporcionar uma redução

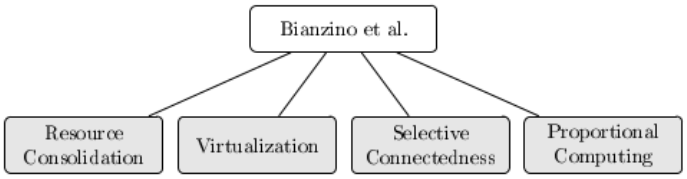


Figura 10 – Taxonomia de abordagens de eficiência energética [Rodrigues, 2016]

.

global de consumo de energia a partir de elementos de rede subutilizados. Devido ao comportamento previsível do tráfego diário, semanal e mensal, surge uma possibilidade de desligar elementos de rede sobressalentes, que estão presentes na rede para lidar com os picos de tráfego, e que na maior parte do tempo encontram-se ociosos.

As abordagens, que adotam o conceito de consolidação de recursos, permitem que a rede seja reconfigurada de modo que o tráfego fique concentrado em um pequeno número de elementos, sendo que uma parte da rede permanece sem uso e desligada. Os comutadores que possuem baixa carga de tráfego em uma estratégia de consolidação de recursos, são candidatos ao desligamento. Porém antes de desligar é preciso redirecionar os fluxos presentes nestes dispositivos e logo que todos os fluxos presentes nas tabelas de fluxo dos comutadores sejam redirecionados, eles finalmente poderão ser desligados com objetivo de economizar energia.

Neste trabalho optamos por utilizar a consolidação de recursos como estratégia para economizar energia;

• Virtualização: Segundo [Bianzino et al., 2012], a virtualização permite o reagrupamento de funções, de modo a permitir que mais de um serviço opere em um mesmo *hardware*, ocasionando uma utilização mais consistente dos recursos comparados a utilização de serviços alocados em várias máquinas separadas fisicamente. Como consequência deste agrupamento temos uma redução do consumo e custos de *hardware*, pois evita-se a subutilização de recursos;

• Conectividade seletiva: A conectividade seletiva de dispositivos consiste em mecanismos distribuídos que permitem que um comutador vá para um estado ocioso, transferindo os seus serviços para outros elementos na rede. Conforme [Rodrigues, 2016], a conectividade seletiva é um conceito similar ao mecanismo de modo de espera do trabalho de [Bolla et al., 2010];

• Computação proporcional: Segundo [Rodrigues, 2016] a computação proporcional é baseada na ideia que um sistema deveria consumir a energia proporcionalmente ao nível de carga de utilização do mesmo. Segundo [Bianzino et al., 2012], dentro da computação proporcional temos dois extremos de proporcionalidade do consumo em relação a carga de trabalho. No pior caso temos os dispositivos que ignoram qualquer estraté-

gia de economia de energia e seu consumo é constante independentemente da carga de trabalho do dispositivo. Estes dispositivos só possuem duas faixas de consumo. Ou eles estão consumindo o máximo de energia ou estão completamente desligados. Por outro lado temos os dispositivos totalmente cientes do consumo de energia. Estes dispositivos conseguem manter uma proporção entre o nível de carga e o consumo de energia. Entre os dois extremos existem dispositivos que não são nem completamente proporcionais e nem que mantém o consumo independente da carga de trabalho.

[Bianzino et al., 2012], nos oferece como exemplos de computação proporcional,a variação dinâmica de voltagem e taxa de enlaces adaptativa. A variação dinâmica de voltagem reduz a diferença de potencial de um processador conforme é reduzida a carga do sistema. A taxa de enlaces adaptativa também depende da carga de tráfego e conforme diminui a demanda por um determinado enlace, reduz-se sua velocidade negociada. Como enlaces com menor velocidade consome menos energia elétrica que enlaces mais velozes, geramos economia ao adaptar o enlace a carga de tráfego.

3.1.3 Escopo de arquitetura

Esta subseção apresenta o escopo de arquitetura. É importante conhecer o escopo de arquitetura, pois podemos classificar eficientemente os diferentes tipos de abordagem no contexto de eficiência energética. [Rodrigues, 2016] apresenta a Figura 11 contendo um diagrama que resume a taxonomia do escopo de arquitetura, descritas a seguir:

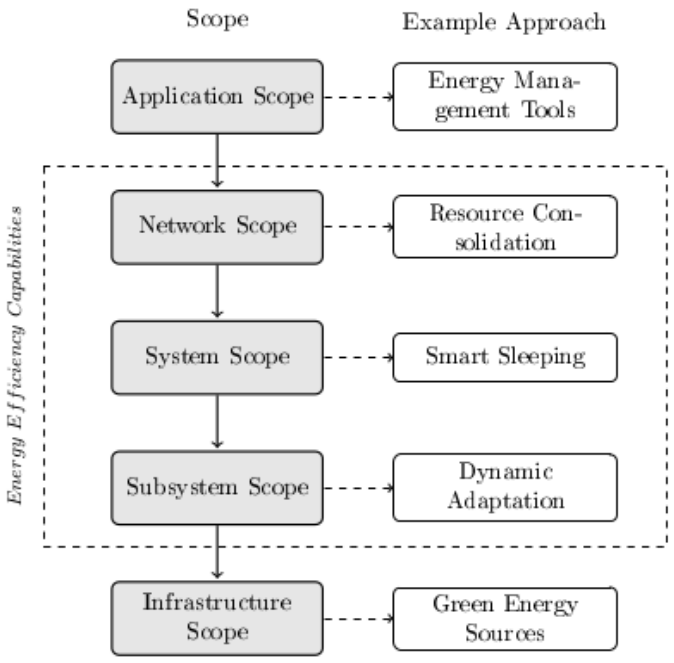


Figura 11 – Escopo de arquitetura [Rodrigues, 2016]

• Escopo de aplicação: Abrange esforços de incorporação de ciência de energia no desenvolvimento do *software*;

• Escopo de rede: Consiste no gerenciamento de nós e roteamento de tráfego ciente do consumo de energia;

• Escopo do sistema: Inclui capacidades de coordenação de nós da rede;

• Escopo de sub-sistema: É responsável por modular o desempenho de componentes internos, como portas, processadores e etc;

• Escopo de infraestrutura: Está relacionado com estratégias que almejam economizar energia à partir de equipamentos de infraestrutura que sustentam os elementos ativos de redes de computadores, como ar-condicionado, fontes de energia renovável e etc.

3.2 Definição do problema

Atualmente as redes legadas não possuem um consumo proporcional à carga de utilização [Mahadevan et al., 2009a], sendo projetadas para permanecerem ligadas 24 horas por dia 7 dias por semana e suportar mais que o pico de tráfego, permitindo cobrir eventos inesperados [Bianzino et al., 2010].

Segundo [Rodrigues, 2016] a economia de energia em redes de computadores pode ser alcançada a partir de diversas estratégias:

• Desligar equipamentos subutilizados de acordo com a demanda por tráfego na rede. Os fluxos são agregados de modo que equipamentos únicos são parcialmente ou inteiramente desligados, através do desligamento de portas e através do desligamento de comutadores. Desligando portas não será possível alcançar níveis consideráveis de economia de energia, entretanto, ao desligar um comutador será, possível alcançar valores significativos de economia de energia. Conforme [Dharmesh and Varma, 2012], o consumo de um comutador varia menos que 8% quando não está com carga e quando está sendo utilizado em sua capacidade máxima.

• Reagrupar um conjunto de serviços, permitindo que mais de um serviço opere em uma mesma máquina. Com isto conseguimos reduzir o consumo de energia, pois uma única máquina sendo bem utilizada consome menos que diversas máquinas subutilizadas.

• Utilizar equipamentos de rede que possuam ciência do consumo de energia e apresentem um consumo de energia proporcional à carga de utilização.

3.3 Cenário considerado

Neste trabalho, para redes definidas por *software*, nós assumimos que a rede é baseada no protocolo *openflow*. Alguns nós serão considerados nós de acesso, onde o tráfego

é originado ou destinado de N hospedeiros, enquanto os outros nós serão considerados nós de encaminhamento, pois não possuem hospedeiros e somente possuem a função de encaminhar dados. Nossa topologia terá a presença de um único controlador que será capaz de monitorar as estatísticas de fluxos de todos os comutadores da rede. Com estas informações, será possível mensurar a taxa de transferência em cada enlace da rede e em cada porta de um comutador.

Conhecendo a capacidade do enlace será possível estimar a demanda atual com relação à demanda de tráfego máxima suportada em cada porta. Com base na demanda atual de tráfego, o controlador decide manter os fluxos em um caminho ou utilizar um novo caminho, gerenciando o tráfego com o objetivo de manter ligado a menor quantidade possível de elementos ativos da rede, e ao mesmo tempo manter o atendimento de todas as demandas. Desta forma, podemos manter o consumo de energia mais proporcional à utilização da rede, resultando em uma utilização mais eficiente de energia.

Apesar da presença de um único controlador, nosso mecanismo é extensível a implementações onde o controlador esteja distribuído, porém logicamente centralizado. Em [Dixit et al., 2013] os autores propuseram o *ElastiCon*, que consiste em uma arquitetura de controlador distribuído elástico, em que a quantidade de controladores SDN distribuídos varia conforme se alteram as condições de tráfego na rede.

Nós trabalhamos com topologias pré definidas na nossa avaliação. Porém nosso mecanismo está preparado para detectar automaticamente qualquer topologia, e será capaz de redirecionar e rotear os fluxos independente da topologia.

3.3.1 Modelo de consumo de energia

De acordo com [Sasaki et al., 2015], o consumo de energia de um nó pode ser representado de acordo com a equação abaixo:

CTs = CBs + EC_1Gbps * n + EC_100Mbps * m, onde:

CTs : Consumo total do comutador
CBs : Consumo básico de um comutador (Fan, Placa mãe e etc.)
EC_1Gbps : Energia consumida por uma porta 1 Gbit/s
EC_100Mbps : Energia consumida por uma porta 100 Mbit/s
n : Número de portas 1 Gbit/s
m : Número de portas 100 Mbit/s

Item de consumo	Potência
Consumo básico	146 Watts
Portas de 100 Mbit/s	0,18 Watts
Portas de 1 Gbit/s	0,87 Watts

Tabela 1 – Valores adotados da potência consumida por um comutador

Neste trabalho adotamos o modelo de consumo de elementos ativos de rede de acordo com [Sasaki et al., 2015]. Em relação aos valores de potência consumida, adotamos valores similares a [Sasaki et al., 2015] e [Mahadevan et al., 2009b]. A tabela 1 apresenta os valores de referência em todos os cenários deste trabalho.

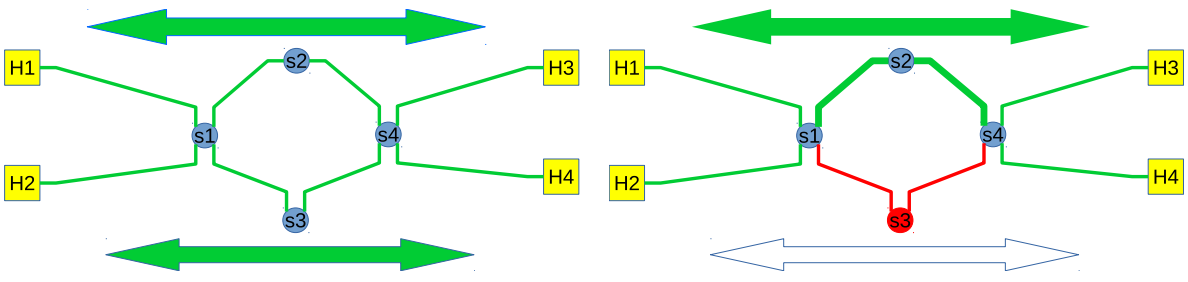
4 Proposta

Nesta seção apresentaremos em detalhes nosso mecanismo de gerenciamento de tráfego ciente do consumo de energia, utilizando as possibilidades de criação de programas para mudança do comportamento padrão da rede introduzidas pelas redes definidas por *software*.

Nossa estratégia principal, é agregar os fluxos da rede na menor quantidade possível de caminhos ativos e manter o restante dos nós desligados para economizar energia. Na Figura 12 (a) apresentamos uma topologia sem a atuação de estratégias de economia de energia. H1, H2, H3 e H4 são hospedeiros e S1, S2, S3 e S4 comutadores. Por outro lado o trafego é concentrado em um caminho (S1-S2-S4), de modo a permitir que uma parte da rede permaneça sem atividades e desligada. Esta situação pode ser observada na Figura 12 (b), onde o comutador S3 se encontra desligado enquanto não houver sobrecarga nos enlaces do caminho S1-S2-S4.

Podemos considerar que o nosso mecanismo segue a linha de pesquisa chamada de engenharia de tráfego verde. Segundo a taxonomia empregada por [Bianzino et al., 2012], nossa estratégia está enquadrada na consolidação de recursos. Nossa proposta tem como pilar principal o monitoramento periódico da rede e o tempo entre cada varredura da rede pode ser ajustado conforme a necessidade de velocidade de reação dos algoritmos de gerenciamento de tráfego. Ou seja, com uma periodicidade menor da varredura, podemos reagir mais rápido a um novo evento na rede e evitar que uma situação indesejada como uma sobrecarga permaneça por muito tempo sem uma reação do controlador. Entretanto, uma periodicidade muito curta pode impactar na escalabilidade da solução para redes com grande número de nós, pois é realizada uma consulta de todas as portas e todos os comutadores em um determinado período.

Através do monitoramento periódico, nosso mecanismo consegue ter um retrato de como se encontra o estado atual da rede, e com este retrato podemos atualizar o modelo da topologia. Podemos observar na visão geral do funcionamento da nossa proposta apresentada na Figura 13, que o monitoramento da rede exerce um papel essencial para



(a) Fluxos em ambos os caminhos (b) Fluxos concentrados em um caminho

Figura 12 – Proposta para concentrar fluxos e desligar pare dos comutadores

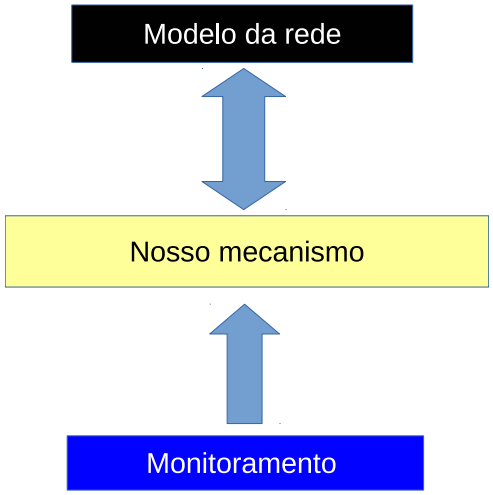


Figura 13 – Visão geral da nossa proposta

que nosso mecanismo tenha uma visão fiel do nível de utilização de cada enlace da rede. Nosso mecanismo somente recebe as informações do monitoramento, ou seja, não há consulta ao módulo de monitoramento. Além do mais a Figura 13 apresenta o Modelo da rede, onde as informações de estado de cada elemento da topologia, são armazenadas, e também consultadas por nosso mecanismo.

Um aspecto importante que está relacionado diretamente com o monitoramento, são os valores limiares que classificam o nível de carga de tráfego a que um enlace está submetido. Caso a carga de tráfego de um enlace ultrapasse ou caia abaixo de um valor limiar, será gerado um evento que atualizará o modelo da rede e, se cabível, o controlador poderá entrar em ação para redirecionar o tráfego de um enlace para outro ou retirar uma possível sobrecarga ou desligar um enlace subutilizado. Na seção 5 apresentaremos os valores limiares de cada carga de utilização.

Podemos perceber a importância do monitoramento contínuo da rede para a nossa proposta, pois nosso mecanismo não possui ciência antecipada de como serão as futuras demandas por tráfego. Como consequência desta decisão de implementação, as novas demandas podem ser alocadas em um determinado enlace e a nova demanda poderá ultrapassar o valor limiar indicativo de sobrecarga. Ou seja, nossa abordagem reage aos eventos gerados pelo monitoramento da rede. Basicamente, esta reação se concretiza ao se redirecionar o tráfego de um enlace para outro. Por esse motivo, nós denominamos nossa abordagem de estratégia reativa.

Por outro lado, às estratégias que tem previsão de como serão as necessidades de largura de banda das demandas por tráfego, chamamos de estratégias pró-ativas. Este tipo de estratégia consegue antes mesmo de alocar um caminho para um novo fluxo, estimar se aquele caminho pode ou não receber a nova demanda, através da consulta das matrizes de tráfego utilizadas nos cenários considerados. Trabalhos que adotam este

tipo de abordagem, não precisam redirecionar dinamicamente o tráfego, pois não haverá sobrecarga devido a um fluxo com uma demanda desconhecida.

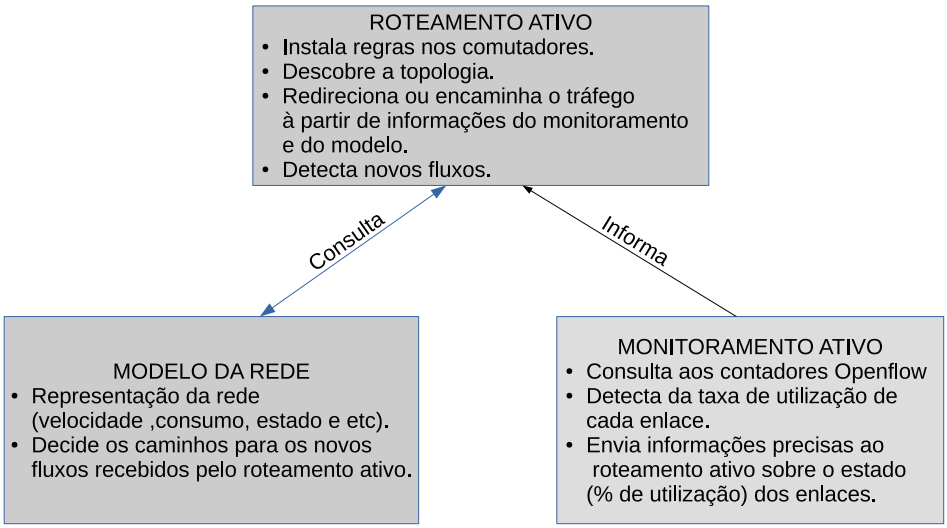


Figura 14 – Abordagem proposta.

Para solucionar o problema de como acomodar os fluxos da maneira mais eficiente possível em termos de economia de energia e com a menor quantidade de comutadores ligados ao mesmo tempo, temos na literatura vários exemplos de otimizações.

Em [Heller et al., 2010], os autores propuseram alguns métodos para se obter o número ótimo de comutadores ativos considerando a demanda por tráfego. Em uma das soluções propostas, eles construíram um modelo formal utilizando linguagens de alto nível para modelagens de otimizações. Porém é conhecido que o problema de se obter a menor quantidade de comutadores ativos possível para atender a uma determinada demanda é um problema NP-completo, tornando a otimização inviável computacionalmente para redes com milhares de nós. Para contornar esta limitação computacional e garantir que o controlador redirecione o tráfego rapidamente e evite congestionamentos e suas consequências, os autores optaram por desenvolver heurísticas que resolvam o problema rapidamente para a maior parte dos casos, aumentando consideravelmente a escalabilidade das soluções. Considerando tais limitações, optamos por não construir um solução ótima, tornando o sistema mais próximo de uma utilização em uma rede real.

Nossa proposta é um mecanismo que tem como objetivo trazer proporcionalidade de energia para redes que não possuem consumo proporcional de energia. Implementamos nosso modelo baseado no paradigma das redes definidas por software para reduzir o consumo de energia através do desligamento de comutadores pouco utilizados ou não

utilizados na rede. Para alcançar nossos objetivos dividimos o sistema em três funções principais: modelo da rede, roteamento ativo e monitoramento ativo de tráfego.

O modelo da rede representado pela Figura 14, armazena o estado atual da rede, representando todos os nós, enlaces e suas características de velocidade, consumo, situação (ligado ou desligado) e etc. Ele é de fundamental importância na tomada de decisão de um caminho, pois é com essa "rede virtual" que o algoritmo decisor de caminhos trabalha. O algoritmo decisor de caminhos é um algoritmo que toma a decisão sobre qual será o melhor caminho, em termos de economia de energia, dentre os caminhos disponíveis seguindo diversos critérios de desempate caso exista mais de um caminho disponível.

Em nossa estratégia, logo ao iniciar a montagem e identificação automática da topologia da rede, já obtemos o benefício da economia de energia pois ligamos apenas o subgrafo referente a uma árvore geradora mínima do grafo original. Ao mesmo tempo, iniciamos o monitoramento periódico da rede com o objetivo de identificar essencialmente um enlace com sobrecarga ou um enlace com baixa carga de tráfego, sendo estes os principais eventos que podemos detectar. Com esta configuração inicial, a rede está preparada para receber o primeiro tráfego entre um hospedeiro de origem e um de destino.

Como podemos observar no Fluxograma 15, que representa de forma simplificada a chegada de um novo fluxo em um comutador, o primeiro fluxo da rede parte de um hospedeiro e alcança o primeiro comutador de acesso (comutador que possui ligação direta com o hospedeiro). O comutador recebe o primeiro fluxo e envia uma mensagem *Openflow* para o controlador, pois ele não possui uma regra instalada indicando em qual caminho deve ser encaminhado o fluxo. O controlador recebe a mensagem e consulta o modelo da rede para executar a busca pelo melhor caminho, caminho este que resulta em no menor incremento possível de consumo de energia. Para obter este caminho desejável no contexto de economia de energia, primeiramente buscamos um percurso em que a maioria dos comutadores, e se possível, todos os comutadores já estejam ativos no momento da escolha do mesmo. Caso encontre um caminho em que todos os seus nós já se encontrem ativos, o controlador instalará a regra no primeiro comutador do caminho indicando a porta do próximo nó. Este processo se repete até que todos os nós do caminho possuam as regras de encaminhamento corretas.

Esta é a situação ideal, pois nenhum comutador a mais será utilizado, e a taxa de utilização dos enlaces entre os comutadores do caminho sofrerá acréscimos até um certo valor limiar. Teremos ainda a situação em que um ou mais enlaces pertencentes a um caminho estarão sobrecarregados e este acontecimento será identificado pelo monitoramento periódico das portas de cada comutador. Esta situação não desejável será registrada no modelo da rede e o controlador, ao tomar a decisão de um caminho, evitará caminhos que possuam enlaces identificados com sobrecarga. Ao mesmo tempo, entrará em ação o redirecionamento de fluxos no enlace identificado como sobrecarregado para caminhos

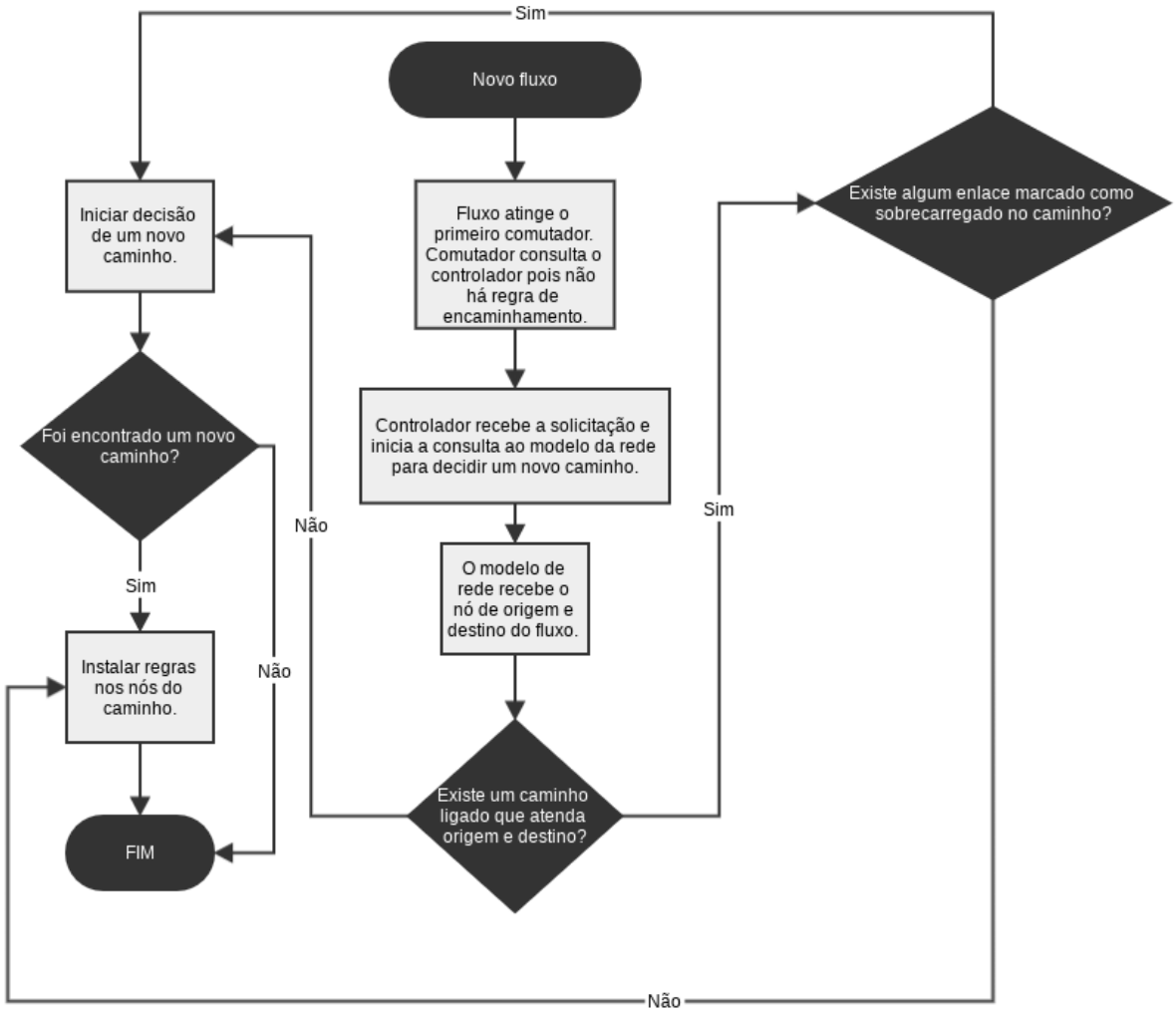


Figura 15 – Fluxograma: Novo fluxo

considerados com baixa carga ou com carga normal. O objetivo desta ação é evitar uma utilização excessiva de um enlace. Novamente priorizamos redirecionar os fluxos do enlace sobrecarregado para caminhos com comutadores ligados. Caso não seja possível, será preciso ligar caminho(s) redundante(s), gerando um acréscimo no consumo de energia.

Inevitavelmente teremos o cenário em que não existirá caminhos alternativos aptos a receberem os novos fluxos, pois ou não há alternativas sem sobrecarga ou não há caminho redundante. Somente neste caso um caminho sobrecarregado receberá novos fluxos. Por outro lado, caminhos que eram utilizados como alternativas para aliviar a carga de enlaces sobrecarregados, podem se tornar subutilizados após algum intervalo de tempo. Este evento é detectado pelo monitoramento periódico da rede após a taxa de utilização de um determinado enlace atingir um valor limiar, e gerará uma tentativa de redirecionamento dos fluxos que passam pelo enlace. A partir do início do processo de redirecionamento de fluxos, caso exista mais de um fluxo no enlace, o controlador irá repetir o processo de

redirecionamento até que não haja mais fluxos neste enlace.. Como existem vários enlaces em cada comutador, somente desligamos um comutador após todos os enlaces estarem livres de fluxos ativos, ou seja, quando o número de fluxos ativos em uma tabela de fluxos de um determinado comutador for igual a zero, podemos desligá-lo.

Ainda teremos cenários onde as opções para o redirecionamento de fluxos não estejam aptas a receber fluxos advindos de redirecionamento. Isso pode acontecer caso os comutadores de destino do redirecionamento tenham atingido valores maiores que um limiar que os impeçam de receber fluxos redirecionados. Neste caso, o redirecionamento é abortado até que exista uma opção viável para receber tais fluxos redirecionados.

Tendo diversos detalhes em mente, podemos imaginar o comportamento da rede em caso de crescimento gradual da carga geral: Ao iniciar uma topologia, ligamos apenas a sub rede equivalente a uma árvore geradora mínima da topologia original. A partir da geração dos primeiros fluxos nesta rede, os caminhos já ligados serão preferencialmente utilizados até que seja detectado uma sobrecarga em um enlace. A partir deste evento, é ligado um novo caminho para diminuir a carga do enlace com sobrecarga. Os novos fluxos serão, novamente acomodados nos caminhos preferencialmente ligados até ser detectado uma nova sobrecarga, gerando novos redirecionamentos. Caso a carga geral da rede esteja sempre crescendo, o processo de ligar novos caminhos se repetirá até que não haja mais opções de caminhos disponíveis. Observamos também a situação oposta, onde o tráfego da rede começa a diminuir. Neste caso, o controlador irá detectar enlaces que se encontram em situação de baixa carga e redirecionará de forma iterativa cada fluxo pertencente ao enlace subutilizado. Após todos os fluxos dos enlaces de um comutador serem redirecionados, será possível desligar o comutador por completo, pois sua tabela de fluxos se encontra livre de regras instaladas. Este processo de desligamento de caminhos ocorrerá até que apenas a árvore geradora mínima esteja ligada.

4.1 Modelo da rede

O modelo da rede em nossa abordagem é um conjunto de classes e métodos para criar uma imagem realista do estado atual da rede. O modelo da rede permite ao controlador ter uma visão global dos elementos da rede de forma detalhada, contendo além dos elementos ativos, informações de hospedeiros e informações de consumo de cada equipamento. O modelo da rede é de suma importância para as decisões sobre quais caminhos poderão ser utilizados, e que resultem no menor consumo de energia possível. A Figura 16 apresenta uma topologia que foi representada pelo modelo da rede. Sempre que houver uma mudança em algum equipamento ou enlace, após um intervalo de tempo o modelo da rede atualizará a nova informação com o auxílio do monitoramento da rede e do roteamento ativo.

Nas próximas subseções apresentaremos em detalhes os elementos que formam a

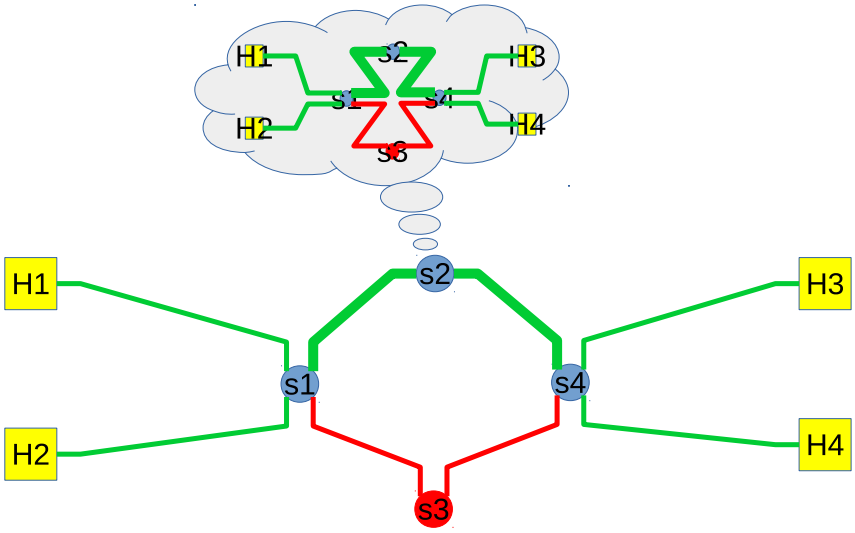


Figura 16 – Modelo da rede

estrutura do modelo da rede.

4.1.1 Classe Nó

A classe Nó faz parte do modelo da rede e sua função principal é proporcionar a criação de objetos que representem os comutadores da topologia. A classe nó permite que sejam gerados os objetos dos comutadores, proporcionando o armazenamento de informações de cada comutador da rede individualmente. Cada objeto nos permite obter um conjunto de operações específicas para os comutadores. Também podemos atualizar os atributos do nó ou obter informações importantes de cada nó que embasarão as decisões sobre caminhos que proporcionem o menor acréscimo do consumo de energia dentre as opções disponíveis.

Dentre os atributos da classe Nó, podemos destacar os atributos referentes ao consumo estimado de energia, pois é a partir dos valores de consumo de cada nó, é que poderemos estimar qual o consumo geral da rede. Como podemos observar no código em *Python* apresentado no Apêndice A, ao se instanciar um objeto da classe Nó devemos informar o consumo básico do comutador. O consumo básico é referente à potência elétrica básica consumida por comutadores, não incluindo a potência consumida pelas suas portas. Também devemos informar a potência consumida por portas de 100 Mbit/s, 1 Gbit/s e etc, pois cada velocidade negociada para a porta possui um consumo diferente, sendo que quanto maior a velocidade maior o consumo [Markiewicz et al., 2014]. Também armazenamos os atributos de estado geral do comutador, que indica que um comutador está ligado ou desligado, conjunto de nós adjacentes do nó em questão, nível de carga de cada porta, indicando sobrecarga ou subutilização e etc. Ou seja a Classe Nó é uma peça importante para o modelo da rede.

4.1.2 Classe Hospedeiro

Assim como a classe Nó, a classe hospedeiro é de suma importância para a composição do modelo da rede. A sua função principal é proporcionar a criação de objetos que representem cada hospedeiro conectado à rede. A classe Hospedeiro permite que sejam gerados os objetos dos hospedeiros, proporcionando o armazenamento dos atributos de cada comutador da rede de forma individual. Diferentemente da classe Nó, nós não consideramos o consumo de cada hospedeiro, pois eles não fazem parte dos elementos ativos da rede. Podemos observar no código em *Python* apresentado no Apêndice B que dentre os atributos pertencentes ao objeto instanciado na classe Hospedeiro podemos destacar o número da porta de conexão com o comutador de acesso, o número da identificação do caminho de dados ou DPID (*datapath id*) do comutador, estado (ativo ou inativo) e velocidade de negociação.

Os objetos da classe Hospedeiro são importantes por vários motivos, mas é importante destacar que sem eles seria desafiador para o nosso mecanismo mapear rotas de um hospedeiro a outro, pois não seria possível conhecer em qual porta o hospedeiro se conecta ao comutador de acesso. Seria inviável até mesmo mapear rotas sem a informação da porta do comutador que possui enlace com o hospedeiro.

4.1.3 Classe Grafo

A classe Grafo é a principal classe para a composição do modelo da rede. O modelo gerado ao se instanciar a classe Grafo, contém a imagem global da rede, tornando viável todas as operações de roteamento, cálculo de consumo, cálculo de caminhos, decisão de caminho com menor consumo e etc. A classe Grafo provê importantes métodos para todas as operações de roteamento e re-roteamento da nossa proposta. Parte do seu código contendo o método construtor é apresentado no Apêndice C.

Nas próximas subseções da subseção 4.1.3 conheceremos com mais detalhes os métodos presentes na classe Grafo.

4.1.3.1 Método de descoberta de caminhos

O objetivo do método de descoberta de caminhos é mapear todos os caminhos entre um nó A e um nó B que possuam o menor número de saltos possível, pois caminhos com tais características exigem menos comutadores ligados, tornando o consumo de energia menor em relação a caminhos com maior número de saltos.

Inicialmente o sistema mantém todos os nós desligados, exceto os nós de acesso. Um das primeiras etapas será calcular as rotas com o menor número de saltos entre todos os pares de hospedeiros na rede. Começando por uma origem, o algoritmo verifica todos os nós adjacentes da origem. Cada nó adjacente será um caminho possível até o destino.

Cada nó adjacente acessado será marcado para não ser novamente visitado. Novamente o algoritmo visita os nós adjacentes (dos adjacentes da origem) e prossegue este passo até encontrar o nó de destino. Como resultado será criada uma lista com todas as opções com menor número de saltos de cada par origem e destino.

A decisão de priorizar caminhos com o menor número de saltos se mostra racional em termos de economia de energia, porém para uma implantação real, precisaremos considerar também caminhos que possam proporcionar um equilíbrio entre economia de energia e qualidade de serviço.

4.1.3.2 Método de análise de caminhos

A análise de caminhos é utilizada para armazenar as estatísticas geradas pelo algoritmo de descoberta de caminhos de forma a subsidiar as decisões do método decisor de caminhos que será detalhado na subseção 4.1.3.3. Durante a busca de caminhos para cada par origem destino, os nós da topologia serão visitados e contabilizados. Ao final da busca de todos os pares origens-destinos possíveis, temos um valor x que representa a quantidade de vezes que um dado nó foi visitado. A partir desta contabilidade, o decisor de caminho priorizará os nós mais escolhidos na construção do mapa de caminhos, pois tais nós pertencem ao maior número de caminhos de forma simultânea e para nossa estratégia eles assumem uma importância maior que nós que foram menos acessados durante as buscas da descoberta de caminhos.

4.1.3.3 Método decisor de caminho

Um par origem e destino pode conter vários caminhos que os ligam. Porém é preciso decidir o melhor caminho dentre as opções quando se deseja criar um fluxo entre eles. Para esta necessidade, criamos um algoritmo decisor de caminho. O caminho escolhido pelo nosso método decisor será com base no caminho em que seus nós gerem o menor acréscimo de consumo comparado às opções de caminhos disponíveis entre um nó A e um nó B. Primeiramente, o algoritmo irá averiguar se o hospedeiro de origem do fluxo e o hospedeiro de destino se encontram em um mesmo comutador. Caso a resposta seja positiva não, é preciso ligar e verificar um caminho. Caso origem e destino não possuam um comutador em comum, o algoritmo fará uma análise nó a nó de cada caminho possível entre origem e destino. Neste caso, o algoritmo priorizará nós e consequentemente caminhos em que a maioria dos nós estejam já ativos. Em caso de empate entre dois ou mais caminhos, entrará em ação um segundo critério, no qual verificamos na lista qual é o caminho que possua a maior quantidade de nós com maior pontuação na análise de caminhos (nós que participem da maior quantidade da caminhos comparados a outros nós menos importantes). Se ainda assim mais de um caminho permanecer na lista de caminhos possíveis, então finalmente escolheremos o caminho em que seus nós possuam

os menores números de identificação ou melhor especificando *Datapath ID* do comutador.

4.1.3.4 Método de cálculo de consumo

O método de cálculo de consumo, tem como objetivo principal calcular o consumo geral da rede periodicamente, consultando a potência consumida por cada nó ativo. O método também calcula a potência que seria consumida se todos os nós na rede estivessem ligados, proporcionando o cálculo de economia alcançada pela estratégia de gerenciamento de tráfego verde. Em uma primeira etapa, o método calcula o consumo individual de um comutador, somando os consumos atuais de suas portas ativas mais o seu consumo básico. Logo após o cálculo individual de consumo de todos os nós, realizamos a soma dos consumos individuais para se obter o consumo total da rede. No Apêndice D é apresentado o código em *Python* da nossa implementação para calcular o consumo da rede.

4.2 Roteamento Ativo

O roteamento ativo é uma unidade lógica da nossa proposta que abrange a comunicação com os comutadores via mensagens *Openflow*, a descoberta automática da topologia através dos componentes *discovery e host_ tracker*, detecção de novos fluxos na rede e roteamento e re-roteamento de tráfegos na rede.

Na próxima subseção mostraremos com mais detalhes o funcionamento do roteamento ativo.

4.2.1 Componente POX - gerente de caminhos

O componente gerente de caminhos pertence à lógica de roteamento ativo, possuindo como uma de suas principais funções a de executar o gerenciamento de tráfego na rede, utilizando outras classes, como a classe Grafo e componentes como o Monitor, como apoio a todas as operações de mudança nas tabelas de fluxo dos comutadores.

O componente é responsável por gerar as mensagens *Openflow* que atualizarão as tabelas de fluxo dos comutadores, sendo capaz de executar tanto o roteamento padrão de fluxos (quando um fluxo é alocado em um caminho pela primeira vez), quanto o redirecionamento de fluxos (quando um fluxo já alocado em um caminho é redirecionado para um caminho alternativo). Em ambas as tarefas, o componente consulta as informações da rede junto ao modelo para gerar um caminho válido e que poupe a maior quantidade de energia dentre as opções de caminhos disponíveis.

O roteamento padrão de fluxos consiste em uma escolha de caminhos dentre os disponíveis e consequente instalação de regras. Ele funciona de forma reativa, ou seja, no momento em que um novo fluxo alcança o comutador de acesso de um hospedeiro, este não possui instalada uma regra para continuar encaminhando os pacotes através da rede. Em

casos como o apresentado, o comutador não possuindo uma regra que atenda ao padrão dos pacotes do fluxo, consulta o controlador enviando uma mensagem *Openflow Packet In*. Por sua vez, o controlador consulta o modelo da rede e obtém um caminho. Ao receber o caminho do modelo de rede, o controlador retorna uma mensagem *Openflow Packet Out*, informando a porta do próximo comutador da lista de caminhos. Este processo vai se repetindo até atingir o nó de destino. Após instalada as regras em todos os comutadores do caminho, o fluxo pode agora trafegar livremente sem provocar novas consultas ao controlador.

Por outro lado, o algoritmo de redirecionamento de fluxos trabalha com o envio de mensagens *Openflow* de forma pró-ativa, ou seja a mensagem parte diretamente do controlador para o comutador, sem depender da etapa de envio de mensagens *Openflow Packet In* do comutador para o controlador. Esta escolha se justifica pelo fato de que, durante os experimentos com o redirecionamento de fluxos trabalhando com o envio de mensagens *Openflow* operando de forma reativa, ocorreram severas perdas de pacotes durante a instalação de regras em cada nó do caminho. A quantidade de pacotes perdidos variou conforme o número de nós nesse caminho. Verificamos que quanto maior o número de nós no caminho, uma quantidade maior de pacotes eram perdidos. Este efeito indesejado ocorreu pois o modo reativo exige um par de comunicações entre o controlador e cada comutador do caminho, causando um atraso. E devido ao atraso, enquanto as mudanças de rota ocorriam em um nó, nos outros permaneciam por um período de tempo com a regra antiga, causando a perda de pacotes. Foi solucionado o problema de perda de pacotes durante o redirecionamento de fluxos, adotando uma instalação de regras pró-ativa nas tabelas de fluxo dos comutadores. Quando o controlador detecta que há uma necessidade de redirecionar fluxos, este envia as novas regras de forma pró-ativa, de forma a reduzir o impacto das mudanças e evitar perdas de pacotes.

Outro empecilho encontrado ao realizar o redirecionamento de fluxos, foi a ordem em que cada ação necessária para se concretizar a mudança de rotas era executada. Para realizar o redirecionamento, precisamos modificar a saída do primeiro e do último comutador adjacente aos comutadores que receberão as novas regras, criando um desvio. Além disso, precisamos instalar as novas regras no caminho alternativo, e por último será necessário remover as regras agora inúteis. Quando não seguimos uma ordem para as operações, ocorreram algumas perdas de pacotes durante o processo de redirecionamento. Foi necessário primeiramente instalar as regras nos comutadores que farão parte do novo caminho. Posteriormente foi criado o desvio mudando as portas do primeiro e último comutador adjacente ao novo caminho, e por último removemos as regras sem uso do antigo caminho. Este procedimento de redirecionamento de fluxos será detalhado na seção 4.2.1.1.

Além da função de redirecionamento e roteamento de fluxos, o componente tam-

bém possui a capacidade de escutar os eventos gerados pelos componentes padrão do controlador POX, *discovery e host_ tracker*, e também possui a função de escutar os eventos gerados pelo componente POX personalizado, responsável pelo monitoramento da rede. O componente *discovery* possibilita que sempre que um novo enlace de rede for detectado, seja gerado um evento com as informações de qual a DIPD dos comutadores participantes do enlace e quais as portas utilizadas para a conexão entre os comutadores. No Apêndice E podemos observar parte do código da função que lida com o evento que foi disparado pelo componente *discovery*.

A partir destas informações, o componente gerente de caminhos alimenta o modelo da rede contendo todos os pares de ligações e portas utilizadas na rede, possibilitando ao controlador ter a visão global dos elementos de comutação da rede. O componente *discovery* não contém os hospedeiros da rede, pois ele apenas detecta elementos comutadores. Para detectar os hospedeiros, utilizamos o componente *host_ tracker*. No Apêndice F podemos observar uma parte do código em *Python* da função que lida com o evento *host_ tracker*:

Este componente detecta hospedeiros sempre que eles iniciam uma comunicação na rede, e o nosso componente gerente de caminhos captura o evento gerado pelo *host_ tracker* e adiciona ao modelo da rede as informações referentes aos hospedeiros, como DIPD do comutador de acesso ligado ao hospedeiro, porta em que o hospedeiro se liga ao comutador e endereço MAC do hospedeiro.

4.2.1.1 Algoritmo de redirecionamento

O algoritmo de redirecionamento foi criado para executar o redirecionamento de fluxos dos enlaces que se encontram subutilizados ou sobrecarregados. Ele está inserido no componente gerente de caminhos e é executado toda vez que um evento de baixa carga ou sobrecarga é disparado. O algoritmo cria mensagens *Openflow* para cada comutador participante do caminho de origem e caminho de destino. A primeira versão do algoritmo trabalhava de forma reativa no envio das mensagens *Openflow*. Ou seja, quando um evento de baixa carga ou sobrecarga era disparado, nosso componente enviava a regra para o primeiro nó do novo caminho e aguardava que o pacote atingisse o próximo nó do caminho, gerando as mensagens *Openflow Packet In* e *Openflow Packet Out*. Este modo de instalação de regras se mostrou falho neste contexto, pois houve perdas de pacote durante o processo de redirecionamento. Apenas após a finalização da instalação de todas as regras, a perda de pacotes cessava.

Neste trabalho devido aos problemas já mencionados, optamos por criar o algoritmo de redirecionamento utilizando a forma pró-ativa para o envio das mensagens *Openflow*. Ou seja, quando o algoritmo detecta que foi gerado um evento de sobrecarga ou baixa carga, as regras serão instaladas em todos os comutadores do caminho sem preci-

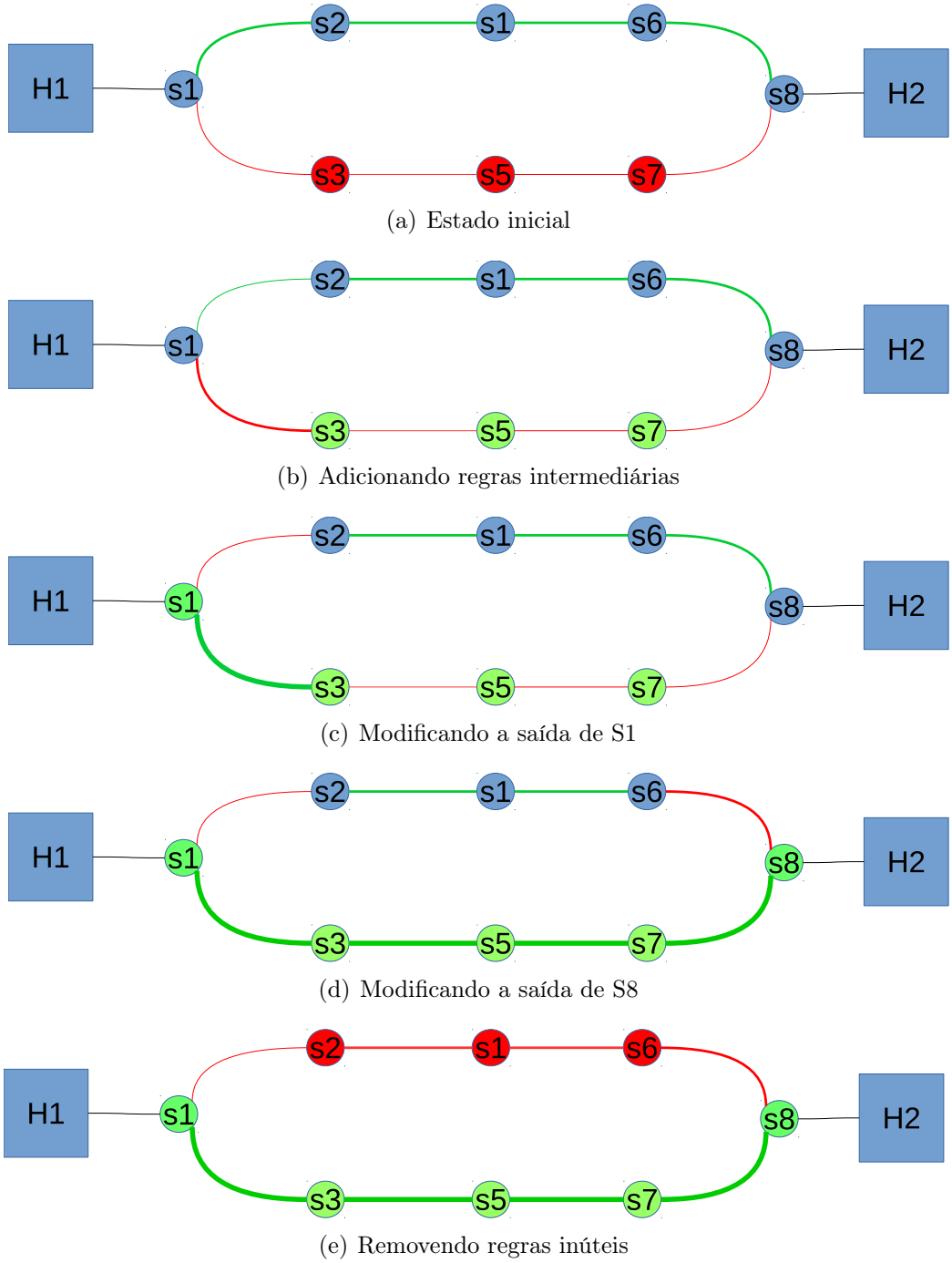
sar aguardar a chegada das mensagens *Openflow Packet In* em cada nó do caminho. Além de optar por uma instalação pró-ativa das regras nas tabelas de fluxo dos comutadores, também precisamos seguir uma ordem para a instalação de regras.

A Figura 17 apresenta cada etapa do processo de redirecionamento, apresentando uma topologia exemplo na qual os quadrados representam os hospedeiros H1 e H2 onde fluxos são gerados entre estes, e os círculos representam os comutadores. A cor azul representa o caminho atualmente utilizado, a cor vermelha representam nós sem regras ou com regras desinstaladas, e finalmente a cor verde representa nós em que serão destinados os fluxos redirecionados. A Figura 17 (a) representa o estado inicial do processo de redirecionamento. Neste estado, os fluxos percorrem o caminho original S1-S2-S4-S6 e S8. Após um evento de sobrecarga em algum enlace do caminho original, será iniciado o algoritmo de redirecionamento para aliviar a carga do enlace,

A primeira ação produzida pelo algoritmo será a instalação das regras de encaminhamento do fluxo escolhido para ser redirecionado nos comutadores intermediários S3-S5 e S7 representados na Figura 17 (b). Neste instante, o fluxo permanece em seu caminho original e assim permanecerá enquanto todas as regras necessárias para o encaminhamento não estejam devidamente instaladas no caminho alternativo. O próximo passo, considerando que as regras estão devidamente instaladas nas tabelas de fluxos do caminho alternativo, será modificar a regra de saída do comutador S1, criando um desvio para o novo caminho. A Figura 17 (c) exemplifica este procedimento. Fazendo parte desta mesma ação de desvio de fluxo, podemos observar a Figura 17 (d), onde a porta de redirecionamento do fluxo em S8 também foi desviada para o caminho alternativo. A partir deste instante, o fluxo redirecionado passará a fazer o percurso S1-S3-S5-S7 e S8. O último procedimento realizado pelo algoritmo será remover as regras desnecessárias dos comutadores já não utilizados pelo fluxo. A Figura17 (e) exemplifica a remoção das regras sem uso. Apresentamos o código em *Python* do algoritmo de redirecionamento no Apêndice J.

4.3 Monitoramento ativo

O monitoramento ativo consiste na utilização dos contadores *Openflow* para se obter estatísticas de uso da rede, de forma a avaliar a carga de utilização dos enlaces. O protocolo *Openflow* nos permite obter estatísticas de portas, informações dos fluxos presentes em um comutador, estatísticas agregadas dos fluxos de um comutador e etc. Em nossa estratégia, priorizamos as informações dos contadores *Openflow* das portas individuais em cada comutador, pois é a partir dos contadores das portas que podemos avaliar a carga de utilização de um enlace. Obtendo as informações dos contadores das portas e conhecendo a taxa de transferência máxima do enlace, nosso mecanismo tem condições de avaliar o nível de utilização de todos os enlaces da rede, permitindo ao



(a) Estado inicial

(b) Adicionando regras intermediárias

(c) Modificando a saída de S1

(d) Modificando a saída de S8

(e) Removendo regras inúteis

Figura 17 – Demonstração do algoritmo de redirecionamento

componente gerente de caminhos avaliar a necessidade de redirecionamento de fluxos para reconfigurar a rede, atendendo às novas características da demanda.

Podemos observar no Fluxograma 18, em linhas gerais, o funcionamento do monitoramento da rede. O monitoramento da rede verifica a taxa de utilização de todos os enlaces da rede e caso se atinja um valor limiar, verificamos se o valor indica uma sobrecarga no enlace. Se sim marcamos o enlace como sobrecarregado, iniciamos o procedimento de redirecionamento de um fluxo presente neste enlace e prosseguimos monitorando os enla-

ces. Caso o valor não indique uma sobrecarga, o algoritmo irá verificar se o valor limiar é indicativo de baixa carga. Se sim terá início o procedimento de redirecionamento de fluxos do enlace. Após o procedimento de redirecionamento, será realizada uma verificação de quantas regras existem na tabela de fluxos dos comutadores que participam do enlace subutilizado. Caso o número de regras seja maior que 1, será repetido o procedimento de redirecionamento de fluxos, com o objetivo de eliminar todos os fluxos presentes em suas tabelas. Caso o número de regras seja igual a zero, será realizado o desligamento lógico do comutador. Por outro lado se o valor limiar não seja indicativo de uma sobrecarga ou de uma subutilização de um enlace, ele será indicativo de um enlace bloqueado. Ao detectar um enlace bloqueado o algoritmo registra esta informação no modelo da rede para evitar que este enlace receba fluxos advindos de redirecionamento. Entenderemos o motivo deste comportamento do algoritmo ao encontrar um enlace bloqueado na subseção 4.3.2.

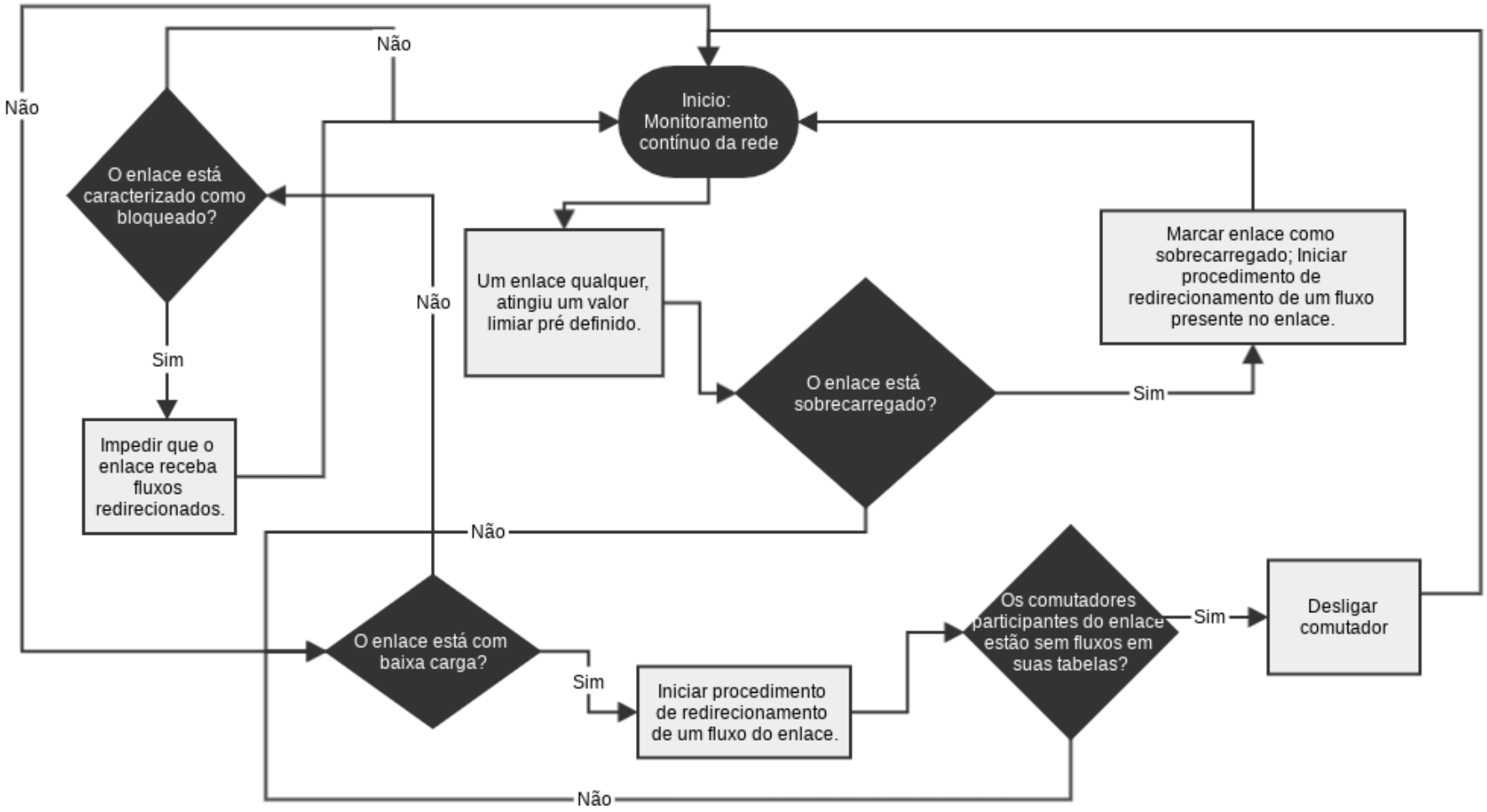


Figura 18 – Fluxograma monitoramento da rede

4.3.1 Componente POX - Monitor

O componente produzido para o controlador POX que denominamos Monitor, foi criado com o objetivo executar o monitoramento dos contadores *Openflow*, e gerar eventos relacionados aos níveis de utilização da rede. Nós detalharemos cada nível de carga de um enlace com mais detalhes na subseção 4.3.2. O componente Monitor executa o conceito de monitoramento ativo de nossa proposta. Para nosso componente ser capaz de manipular os contadores do protocolo *Openflow*, precisamos nos registrar nos eventos disponibilizados pelo núcleo do controlador POX e pelo módulo *openflow*. No Apêndice G apresentamos o código em *Python* onde observamos os registros nos eventos *FlowStatsReceived*, que obtém a lista de contadores dos fluxos da tabela de fluxos de um determinado comutador, e *PortStatsReceived* que obtém a lista de contadores das portas pertencentes a um determinado comutador.

Apesar das várias opções de inscrição em eventos disponibilizadas pelo núcleo do controlador que retornam estatísticas diversas, nós utilizamos apenas as estatísticas *PortStatsReceived* para avaliar as condições de tráfego em todos os enlaces, mas a partir do evento *FlowStatsReceived*, podemos conhecer qual o fluxo é o responsável pela maior demanda por largura de banda em um comutador e adotar estratégias de redirecionamento de tráfego que priorizem o redirecionamento de fluxos com maiores ou menores demandas, dependendo da capacidade de largura de banda disponível no caminho receptor da demanda. O parâmetro *self._handle_portstats_received* indica para o método *addListenerByName* o nome do método responsável por tratar os dados gerados quando o evento é disparado. O mesmo raciocínio se aplica ao evento *FlowStatsReceived*. No Apêndice H, podemos observar como lidamos com os dados produzidos pelo evento *Portstats*.

No código em *Python* das estatísticas recebidas pelo evento *PortStats*, nós criamos uma função que trata o evento com o objetivo de calcular a largura de banda consumida em cada porta de um determinado comutador. Em uma primeira chamada da função, nós não calculamos a largura de banda consumida, somente armazenamos na lista de estatísticas do nó a contabilidade dos dados transferidos pela porta em *Bytes*. Nas próximas chamadas à função calculamos a largura de banda consumida comparando com a contagem em *Bytes* da última vez que foi chamada a função.

Em nossos experimentos, calculamos a largura de banda consumida por uma porta a cada segundo, porém podem ser usados intervalos maiores para melhorar a escalabilidade da solução. Após realizar os cálculos de largura de banda consumida em cada porta ativa na rede, nós comparamos o consumo de banda atual com a largura de banda máxima do enlace e verificamos se houveram mudanças de estados em um enlace. A explicação detalhada de cada estado dos enlaces serão dadas na sub-seção 4.3.2.

Como utilizamos mais de um componente *openflow* na nossa solução, precisamos

realizar a comunicação entre um componente e outro em eventos específicos, como uma sobrecarga de um enlace. O controlador POX disponibiliza para esse fim a possibilidade de publicarmos nossos eventos que ficarão disponíveis para todos os componentes que desejarem neles se inscrever. Deste modo criamos três eventos distintos, cada um sendo disparado em condições diferentes de tráfego no enlace. O primeiro evento é disparado ao ser detectada uma sobrecarga em um enlace na rede. O segundo evento é disparado em casos de subutilização da rede. O terceiro é disparado em variações intermediárias de carga de tráfego em um enlace. Apesar de possuirmos dois distintos estados intermediários eles disparam o mesmo evento, pois ambos não geram ações de redirecionamento de fluxos no componente gerente de caminhos.

O Apêndice I nos apresenta um exemplo de chamada do método que publica o evento. O primeiro parâmetro que passamos para a função do evento *intermedEvent*, é a fração da largura de banda consumida pela capacidade máxima de largura de banda do enlace. O segundo parâmetro se refere ao rótulo de nível de carga que provocou o disparo do evento. Conforme o tipo de rótulo variamos o comportamento do controlador. O terceiro parâmetro se refere à identificação do caminho de dados DPID do comutador *Openflow*. É com esta identificação que o controlador irá enviar mensagens *Openflow* para o comutador correto. E por fim, enviamos o número da porta em que foi gerado o evento. Nós utilizamos o componente Gerente de caminhos para se inscrever e lidar com os eventos gerados pelo componente Monitor.

4.3.2 Estados dos enlaces de uma rede

Em nossa abordagem optamos por categorizar os enlaces de acordo com a taxa de transferência do mesmo em relação a sua largura de banda máxima. Para tais categorias damos o nome de estados de um enlace. O principal motivo de segregarmos cada estado é devido à possibilidade de mudarmos o comportamento do controlador conforme um enlace muda de um estado para o outro a fim de atingir o benefício da economia de energia. Para transitar de um estado para o outro é preciso que a taxa de utilização de um enlace ultrapasse ou se torne menor que um determinado valor limiar, que pode ser ajustado pelo administrador da rede. Nós definimos os valores limiares de cada estado na seção 2.

Com a categorização por estados, podemos evitar que um comutador com enlaces sobrecarregados degrade a qualidade da rede e, ao mesmo tempo, evitamos que enlaces subutilizados permaneçam ligados, consumindo energia desnecessariamente. Em nosso método proposto, os enlaces em uma rede serão categorizados em quatro estados de acordo com o nível de utilização de largura de banda em relação à largura total de banda disponível. Em cada estado haverá um comportamento diferente em relação a capacidade de aceitar ou rejeitar fluxos e também definirá qual comutador será candidato

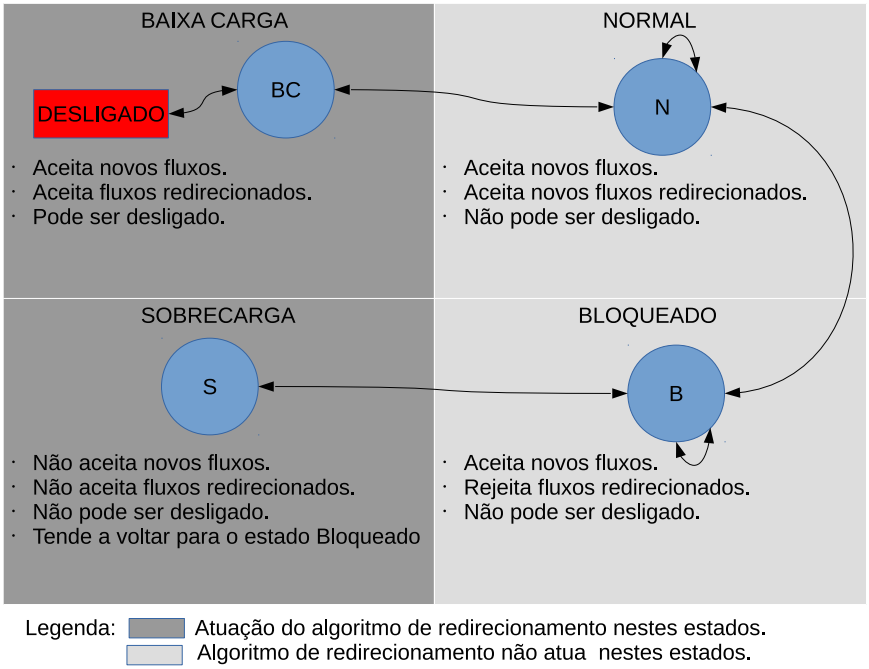


Figura 19 – Estados assumidos pelo enlace

ao desligamento com o objetivo de economia de energia.

O primeiro estado de um enlace, considerando seu nível de utilização em relação a sua taxa de transferência máxima, será o estado Baixa Carga observado na Figura 19 (BC). Ele será adotado em enlaces com pouca utilização ou sub-utilização. Os comutadores que possuírem enlaces com este estado poderão, receber novos fluxos a partir de decisões do controlador, e somente os comutadores que possuem portas que estejam no estado Baixa Carga, serão candidatos ao redirecionamento de fluxos para um caminho alternativo e consequente desligamento. Ou seja, o motivo da existência do estado Baixa Carga é proporcionar um esvaziamento na tabela de fluxos de um comutador, para em seguida promover o desligamento do mesmo.

O segundo estado é chamado de Normal observado na Figura 19 (N). Neste estado estão enquadrados todos os enlaces com uma taxa de utilização considerada de baixo comprometimento dos indicadores de qualidade da rede, pois apesar de não haver subutilização, também não observamos características de um enlace sobrecarregado. Enlaces que possuem o estado normal, aceitam novos fluxos e aceitam fluxos advindos de redirecionamentos. Quando um enlace se encontra em estado normal, o mecanismo de redirecionamento de tráfego não entra em ação. Podemos concluir que o objetivo do estado normal é manter um comutador ligado e manter seus enlaces receptivos para novos fluxos ou fluxos redirecionados.

O terceiro estado é chamado de Bloqueado. Neste estado consideramos que os

enlaces de um comutador estão próximos de uma sobrecarga de tráfego, porém ainda temos uma margem antes de atingir este patamar. O estado Bloqueado permite que novos fluxos sejam adicionados nas portas com este estado, entretanto rejeitará fluxos advindos de redirecionamentos de tráfego. Ou seja, o estado Bloqueado mantém um enlace disponível para novos fluxos, porém o protege de fluxos advindos de redirecionamento. O objetivo desta estratégia é evitar que um fluxo redirecionado seja novamente redirecionado ao chegar em seu destino. Este comportamento indesejado pode gerar um *looping* de redirecionamento. Caso não tivéssemos adotado o estado bloqueado, um fluxo redirecionado de um enlace deste tipo poderia ser insuficiente para tirar o enlace de destino do estado de baixa carga, mas suficiente para tirar o enlace de origem da condição de sobrecarregado. Assim, o fluxo chegaria ao destino e logo em sequencia seria novamente enviado para a origem, gerando uma nova sobrecarga. Neste caso, o processo de redirecionamento se repetiria indefinidamente, até que o enlace de destino atingisse o estado normal ou o enlace de origem comportasse o fluxo redirecionado sem gerar sobrecarga.

Finalmente, temos o estado denominado Sobrecarga. Este estado estará presente em enlaces onde a largura de banda atual esteja próxima da largura de banda máxima suportada atualmente. Os enlaces com este estado não poderão receber novos fluxos, e também não poderão receber fluxos advindos de redirecionamentos, e suas portas identificadas pelo controlador como sobrecarregadas, serão candidatas ao redirecionamento de fluxos até a porta e o enlace atingirem o nível bloqueado. Como o nosso objetivo é utilizar ao máximo todos os recursos ativos na rede sem comprometer os parâmetros de qualidade, não continuaremos iterando para redirecionar cada fluxo até atingir o estado "baixa carga". A principal motivação deste estado é evitar que um enlace permaneça sobrecarregado e prejudique a qualidade da conexão.

5 Avaliação

Nesta seção, apresentaremos o comportamento de nossa abordagem em termos de economia de energia. Avaliamos o modelo proposto utilizando o simulador *Mininet* 2.3.0 [Lantz et al., 2010]. Usamos o controlador POX como controlador *Openflow*. Para fazer o papel do comutador *Openflow*, utilizamos o *OpenVSwitch* 2.5.2. Os experimentos foram executados em um computador possuindo processador Intel Core i5 de sétima geração com *clock* máximo de 3.1 GHz e 8GB de RAM. O sistema operacional utilizado foi o *Lubuntu* 16.04. O tráfego aleatório UDP com largura de banda variável, foi gerado pelo gerador de tráfego D-ITG [Avallone et al., 2004]. Segundo [Avallone et al., 2004], o gerador de tráfego de Internet distribuído é uma plataforma que nos fornece uma produção de tráfego que segue padrões definidos pelo tempo entre as partidas e tamanhos dos pacotes. Utilizamos valores de consumo do comutador semelhante a [Sasaki et al., 2015] e [Mahadevan et al., 2009b]: Consumo básico de 146 Watts e o consumo de cada porta de 1 Gbit/s ligada de 0.87 Watts.

Construímos cenários de redes homogêneas e heterogêneas em relação à velocidade negociada para os enlaces. Consideramos que todos os nós possuem uma largura de banda máxima equivalente a 1 Gbit/s nos cenários homogêneos e de 250 Kbit/s a 1 Gbit/s em cenários heterogêneos. Detalharemos mais este assunto na subseção 5.4.1.

Como vimos na subseção 4.3.2, categorizamos os enlaces em quatro níveis distintos de carga, sendo que cada nível representa um estado de um enlace. Em nossa abordagem, consideramos os valores exibidos na tabela 2.

Estado	% de utilização do enlace
Sobrecarga	>80%
Bloqueado	>60% e <80%
Normal	>20% e <60%
Baixa carga	>0% e <20%

Tabela 2 – Estado do enlace vs Nível de utilização

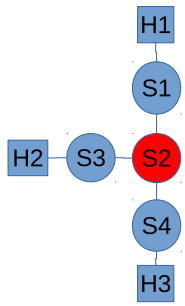
5.1 Metodologia de avaliação

Para avaliar nosso modelo proposto, preparamos 3 cenários para explorar as capacidades adicionadas ao controlador pelo nosso mecanismo. Primeiramente, montamos um cenário estático para mostrar a capacidade da nossa proposta em uma rede com ausência de tráfego em uma parte ou toda a rede. Posteriormente, montamos um cenário exemplo com uma topologia e características que permitem demonstrar de forma simples o funcionamento do nosso mecanismo. Para os primeiros cenários, utilizamos uma topologia contendo 4 comutadores e 3 hospedeiros. Esta topologia pode ser visualizada na Figura 21. Finalmente, apresentaremos o cenário realista no qual comparamos nossa proposta

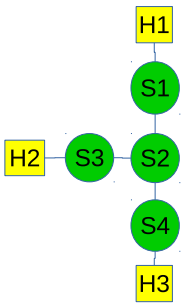
com o algoritmo proposto por [Markiewicz et al., 2014], considerando três níveis de carga de tráfego distintos (baixa carga, média e alta), e analisamos os resultados em termos de economia de energia e vazão média de carga em cada nível.

5.2 Cenário estático

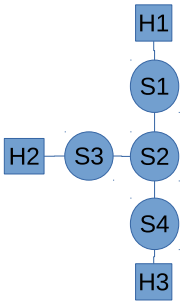
Nesta subseção apresentaremos uma das estratégias adotadas para alcançar economia de energia em baixos níveis de utilização de uma rede. A primeira estratégia consiste em desligar comutadores em casos de segmentos da rede que não possuam tráfegos ativos.



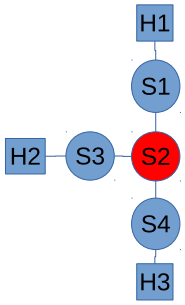
(a) Estado inicial: Apenas comutadores de acesso serão ligados.



(b) Primeiros fluxos: Comunicação entre H1, H2 e H3.



(c) Encerramento de todos os fluxos.



(d) Desligamento de S2 devido a inatividade da rede.

Figura 20 – Comportamento do mecanismo no cenário estático

Para demonstrar esta funcionalidade, utilizamos uma topologia com quatro nós, como observado na Figura 20(a). Os comutadores são representados por círculos e os hospedeiros por quadrados. Logo após a execução do cenário, se iniciará a montagem da topologia, e o controlador irá detectar os novos comutadores e enlaces. Inicialmente, somente comutadores de acesso serão mantidos ligados, e todos os demais serão mantidos desligados. Observe que na Figura 20(a), S2 está com a coloração vermelha, representado um comutador desligado. Após alguns segundos, tem-se início os fluxos entre os hospedeiros H1, H2 e H3. A primeira ação do controlador será a ativação de todos os nós que participam dos novos caminhos ativos. Os nós ligados por ação do controlador, são

representados em verde na Figura 20(b). Após 10 segundos de atividade, serão encerrados os fluxos, e se dará início ao período de inatividade total de comunicações entre os hospedeiros representado pela Figura 20(c). Após dois minutos de inatividade, será gerado um evento POX *host_ event*, informando que um determinado hospedeiro deixou a rede devido a inatividade. Com base no evento, será efetuada uma busca pelos comutadores do caminho utilizado pelo hospedeiro à procura de comutadores sem fluxos ativos. Caso a busca seja efetiva, será(ão) desligados o(s) comutador(es) que atendam à esta restrição. Podemos observar na Figura 20(d) esta situação. Caso ainda existam fluxos na tabela de fluxos do comutador, não será efetuado o desligamento do mesmo.

Estas ações proporcionaram economia de energia em um cenário controlado, onde a maioria da comunicação é encerrada. Na prática porém, teremos poucas situações em que a rede estará em tão baixo nível de carga. Nas próximas subseções trataremos de outro tipo de estratégia para economizar energia.

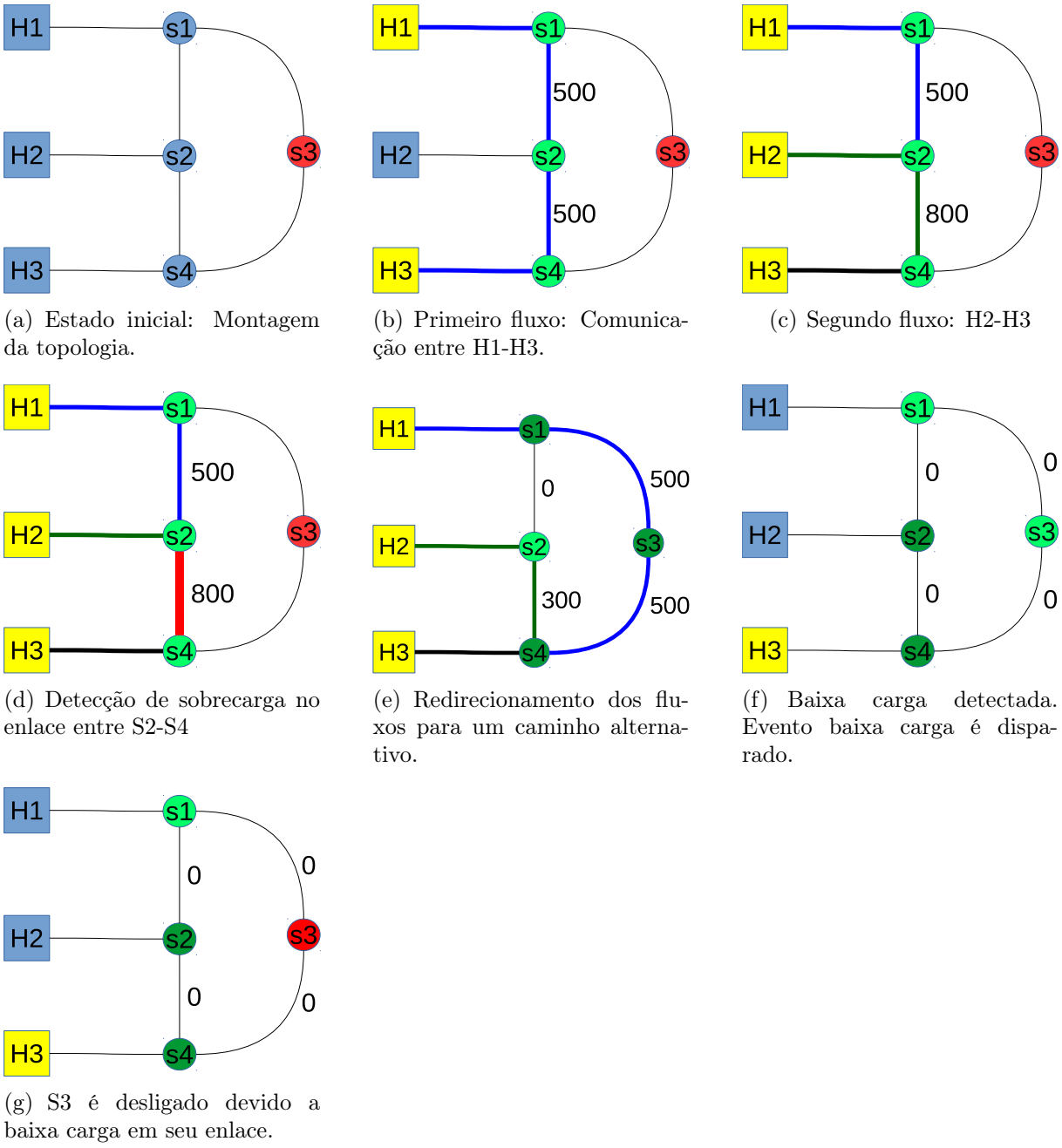
5.3 Cenário dinâmico

Na Figura 21, os comutadores são representados por círculos e os hospedeiros por quadrados. Os comutadores e hospedeiros ligados e sem fluxos ativos possuem cor azul, enquanto comutadores ligados e com fluxos possuem cor verde claro. Os comutadores representados pela cor vermelha, estão desligados. O hospedeiro H3 representa nosso servidor, enquanto H1 e H2 são os clientes. Todos os enlaces são *Gigabit Ethernet*. No começo, não há carga na rede e todos os comutadores estão no estado "baixa carga". O controlador inicia a detecção automática da topologia da rede para construir a árvore geradora mínima, conectando todos os hospedeiros. Observe na Figura 21(a) que o comutador S3 fica desligado. Esta Figura representa o estado inicial da rede quando ainda não há nenhum fluxo ativo presente.

O cliente H1 inicia a comunicação com o servidor e a largura de banda demandada por ele é de 500 Mbit/s. Como podemos observar na Figura 21(b), o controlador encontra o menor caminho S1-S2-S4 entre H1 e H3 na árvore geradora mínima calculada. Como os comutadores estão em baixa carga, eles aceitam o fluxo e a rota é instalada.

Após 5 segundos do início da comunicação entre H1 e H3, tem início a comunicação entre o cliente H2 e o servidor H3 (Figura 21(c)), e a largura de banda demandada por ele é de 300 Mbit/s. O controlador identifica o caminho S2-S4 já ligado na árvore geradora mínima. Como os comutadores estão no estado de carga normal, eles aceitam o fluxo e a rota é instalada.

Neste momento, o enlace S2-S4 entra em estado de sobrecarga (Figura 21(d)). À partir deste instante, o controlador analisa os fluxos presentes no enlace S2-S4 e tenta desviá-los para um caminho alternativo, caso exista, composto por comutadores ligados



(a) Estado inicial: Montagem da topologia.

(b) Primeiro fluxo: Comunicação entre H1-H3.

(c) Segundo fluxo: H2-H3

(d) Detecção de sobrecarga no enlace entre S2-S4

(e) Redirecionamento dos fluxos para um caminho alternativo.

(f) Baixa carga detectada. Evento baixa carga é disparado.

(g) S3 é desligado devido a baixa carga em seu enlace.

Figura 21 – Comportamento do mecanismo no cenário dinâmico

e com o estado “normal” ou “baixa carga”. Caso não encontre tais opções de caminhos, o controlador busca por possibilidades, considerando também os comutadores desligados. Havendo um caminho alternativo, o(s) fluxo(s) são redirecionados.

O controlador constata que o fluxo entre H1 e H3 possui uma opção de rota desligada (S1-S3-S4) . Então ele irá ligar e instalar a regra de encaminhamento do fluxo em S3 e mudará a regra de encaminhamento em S1 e S4 (Figura 21(e)). Neste momento nenhum comutadores trabalha em sobrecarga, porém toda a topologia está ligada, e o consumo será mais alto que no início da execução.

Supondo que os fluxos entre H1 - H3 e H2 - H3 terminem, os enlaces presentes em S3 entrarão no estado “baixa carga”, como apresentado na Figura 21(f) . O controlador se certificará que todos os fluxos remanescentes na tabela de fluxos de S3 serão redirecionados para comutadores ligados e em estado “normal” ou “baixa carga”. Caso a operação seja possível, o controlador desligará S3 (Figura 21(g)).

A Figura 22 apresenta o gráfico da porcentagem de economia de energia e o gráfico da porcentagem de carga através do tempo da emulação. Podemos notar que a economia está relacionada com a carga, porém não é diretamente proporcional, pois a economia continua até o(s) enlace(s) atingirem limites onde o controlador decide ligar os caminho(s) alternativo(s). Como exemplo deste comportamento, podemos observar o período de 17s até 21s. Em 21s, a carga sobre o enlace ligado sobe e ele fica sobrecarregado. O controlador "percebe"a situação e em menos de 0,2 milissegundos reage ligando um caminho alternativo, fazendo desabar a economia.

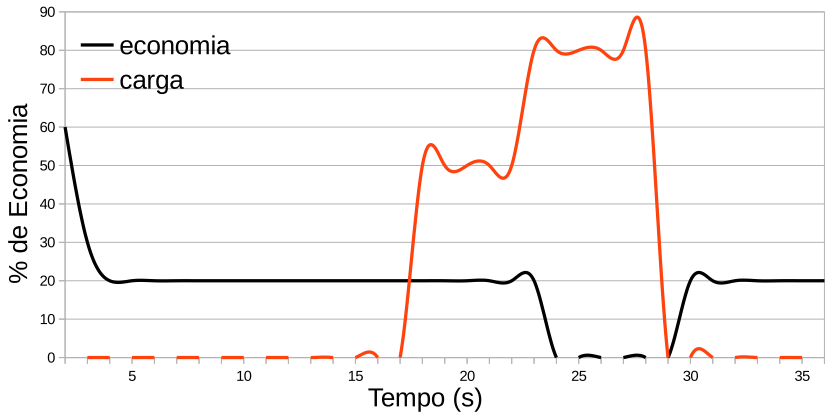


Figura 22 – % de economia de energia versus carga

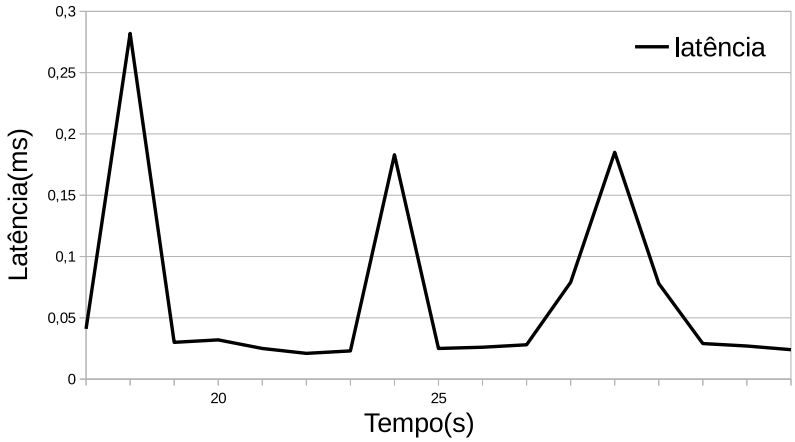


Figura 23 – eventos que geram picos de latência

Na Figura 23, podemos observar o gráfico contendo o atraso após cada evento durante a emulação. O primeiro pico de latência observado corresponde à primeira ins-

talação das regras nos comutadores de forma reativa (através da mensagem *openflow packet in*), após o início da geração de tráfego. O atraso ocorre devido à utilização do canal entre comutador e controlador, pois ainda não existe regra instalada, o que força o comutador a consultar o controlador em busca de uma regra de encaminhamento.

O segundo pico ocorre devido ao processo de redirecionamento de tráfego após a identificação de sobrecarga em um caminho. Como este processo acontece de forma reativa pelo controlador, não é necessário esperar o *packet-in* do comutador para tomar uma atitude. O próprio controlador de forma pró-ativa, instala as regras de uma só vez em todo o novo caminho do fluxo. De forma similar ao segundo pico na Figura 23, o terceiro pico ocorre após a identificação de caminhos ativos em situação de baixa carga. Neste caso, o controlador irá buscar um caminho ligado em que possa receber os fluxos originários do caminho em baixa carga. Ao encontrar tal caminho, o controlador irá redirecionar o fluxo, causando um acréscimo momentâneo na latência da rede. Ou seja, o controlador "observa" que há baixa carga nos enlaces do comutador de encaminhamento, disparando um evento que leva a uma nova seção de redirecionamento de fluxos, porém desta vez com o objetivo de desligar o comutador.

5.4 Cenário realista

Nesta seção, apresentaremos um comparativo, em termos de economia de energia, entre nossa proposta e algoritmo proposto por [Markiewicz et al., 2014]. Também comparamos o desempenho em termos de economia de energia entre cenários com enlaces homogêneos e enlaces heterogêneos.

Assumimos que o ambiente de testes possui uma topologia semelhante à apresentada na figura 24. Essa topologia, possui 45 nós divididos entre, 18 comutadores, 27 hospedeiros e 95 enlaces entre os nós. A topologia apresentada representa uma rede típica de campus [Markiewicz et al., 2014]. Os nós de 5 a 18 são considerados nós de acesso, e os nós 1, 2, 3 e 4, nós de encaminhamento. O restante dos nós serão os hospedeiros, onde estarão hospedados os clientes/servidores dos geradores de tráfego utilizados nas emulações.

Os autores de [Markiewicz et al., 2014] propuseram três cargas proporcionais para cada nível, considerando o comportamento de tráfego noturno, a média de tráfego diurno e o tráfego de pico anual. Conforme os autores, o tráfego de pico anual é cinco vezes maior que o tráfego noturno, e o tráfego médio diurno é três vezes maior que o tráfego noturno. Geramos cada nível de carga de tráfego, utilizando uma distribuição de *Poisson* com uma expectativa de intervalo igual a três e tamanho igual a quarenta, para determinar o momento em que um novo fluxo de largura de banda aleatória (variando conforme os níveis de geração de carga) é adicionado ao sistema. De acordo com [Cao et al., 2003], o *packet inter-arrival time* da internet tende a respeitar a distribuição *Poisson*.

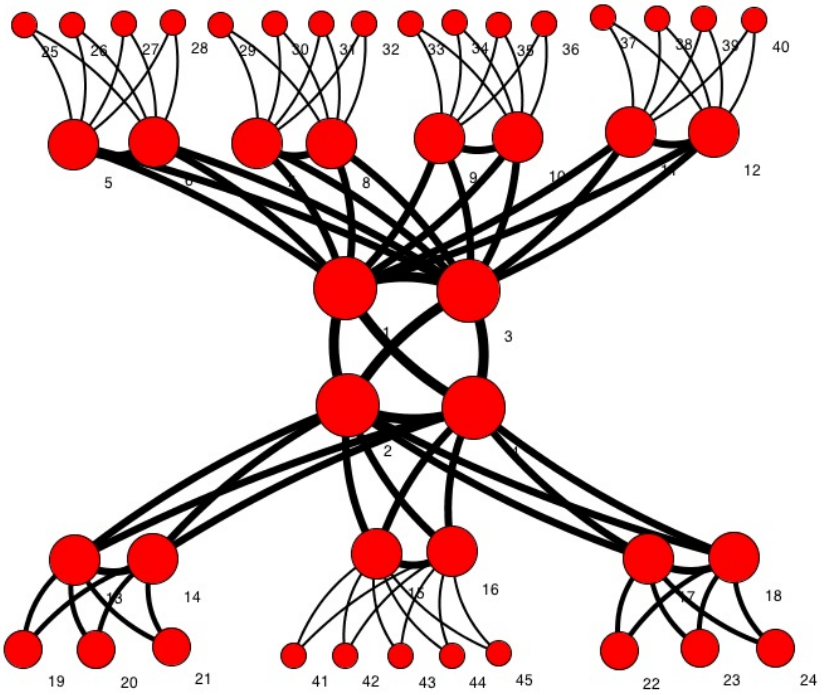


Figura 24 – Topologia típica de uma rede de campus. [Markiewicz et al., 2014]

5.4.1 Enlaces homogêneos e heterogêneos

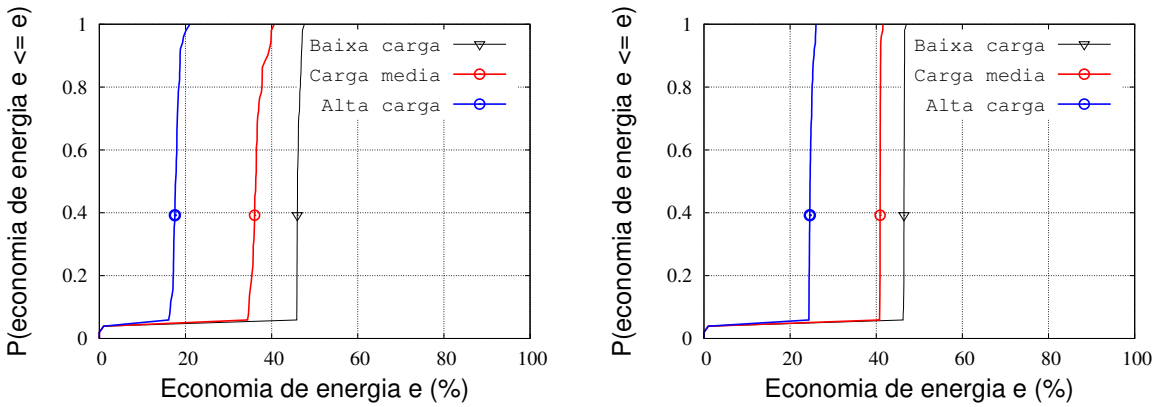
Consideramos os enlaces homogêneos quando todos os enlaces pertencentes à topologia possuem a mesma velocidade negociada. Nossos cenários homogêneos utilizaram todos os enlaces com velocidade negociada igual a 250 Mbit/s.

Consideramos os enlaces heterogêneos quando pelo menos um enlace da topologia possui um valor para a velocidade negociada diferente dos demais. Em nossos cenários heterogêneos, consideramos que, cada enlace pertencente aos comutadores do núcleo, possuem a velocidade de suas interfaces *Ethernet* quatro vezes mais velozes do que os outros enlaces da rede. A princípio, utilizamos o valor de 4 Gbit/s para os enlaces do núcleo da rede e 1 Gbit/s para os demais, porém, devido a uma limitação de nossa versão do *Mininet*, não foi possível utilizar uma velocidade negociada maior que 1000 Mbit/s. Por este motivo reduzimos as velocidades dos enlaces padrão, mantendo a proporção de uma velocidade negociada 4 vezes maior para os enlaces do núcleo. Ou seja, chegamos aos valores de 1000 Mbit/s para os enlaces do núcleo e 250 Mbit/s para os demais enlaces.

5.4.2 Comparativo entre cenários homogêneos e cenários heterogêneos

Nesta subseção, apresentaremos um comparativo em termos de economia de energia, entre um cenário que apresente enlaces homogêneos e um cenário que apresente enlaces heterogêneos. Em ambos os cenários utilizamos os mesmos parâmetros de geração de tráfego entre níveis de carga (baixa, média e alta).

Em cada cenário executado com baixa carga, um novo fluxo possuirá uma taxa de transferência mínima de 0 Mbit/s e máxima de 50 Mbit/s. Em carga média de tráfego, cada novo fluxo pode variar entre 51 Mbit/s a 150 Mbit/s. Em carga máxima, cada novo fluxo pode variar entre 151 Mbit/s a 250 Mbit/s. Cada novo fluxo possui uma duração fixa de 15 segundos e o tempo máximo de cada execução é de 120 segundos. Executamos 50 vezes com esta configuração e obtivemos a porcentagem de energia economizada em relação ao consumo total possível para a topologia em cada execução. Em seguida, ordenamos os resultados e calculamos a função de distribuição cumulativa.



(a) Homogêneo - % de economia de energia versus carga

(b) Heterogêneo - % de economia de energia versus carga

Figura 25 – Cenários: Enlaces homogêneos vs Enlaces heterogêneos

Em baixa carga, o comportamento do cenário com enlaces homogêneos e enlaces heterogêneos foi semelhante. Observe que a curva referente à baixa carga na figura 25(a) e a curva correspondente à mesma carga na figura 25(b), apresentam o mesmo comportamento. A diferença entre os cenários considerando um nível de confiança de 95% é de aproximadamente 0,0355%. Em relação à vazão média apresentada pelos cenários em baixa carga, também observamos uma equivalência entre os resultados. Como podemos observar Figura 26, em ambos os cenários obtemos uma média de 25 Mbit/s aproximadamente em Baixa Carga. Considerando um nível de confiança de 95% obtemos um empate técnico entre os cenários. Observando o mapa dos comutadores ligados na maior parte das execuções nas figuras 27(a) e 27(c), concluímos que ambos os cenários apresentaram a menor quantidade de comutadores ligados dentre todas as cargas analisadas. Devido à baixa vazão entre os enlaces da rede, não observamos redirecionamento de fluxos e consequente acréscimo no consumo de energia.

Em carga média, obtivemos uma vazão média similar entre os cenários com enlaces homogêneos e heterogêneos. Considerando um nível de confiança de 95%, temos um empate técnico, pois o cenário homogêneo apresentou em seu intervalo de confiança as vazões de 87,480 Mbit/s a 98,837 Mbit/s e o cenário heterogêneo apresentou um intervalo de confiança que partiu de 89,901 Mbit/s a 101,436 Mbit/s. Apesar desta equivalência

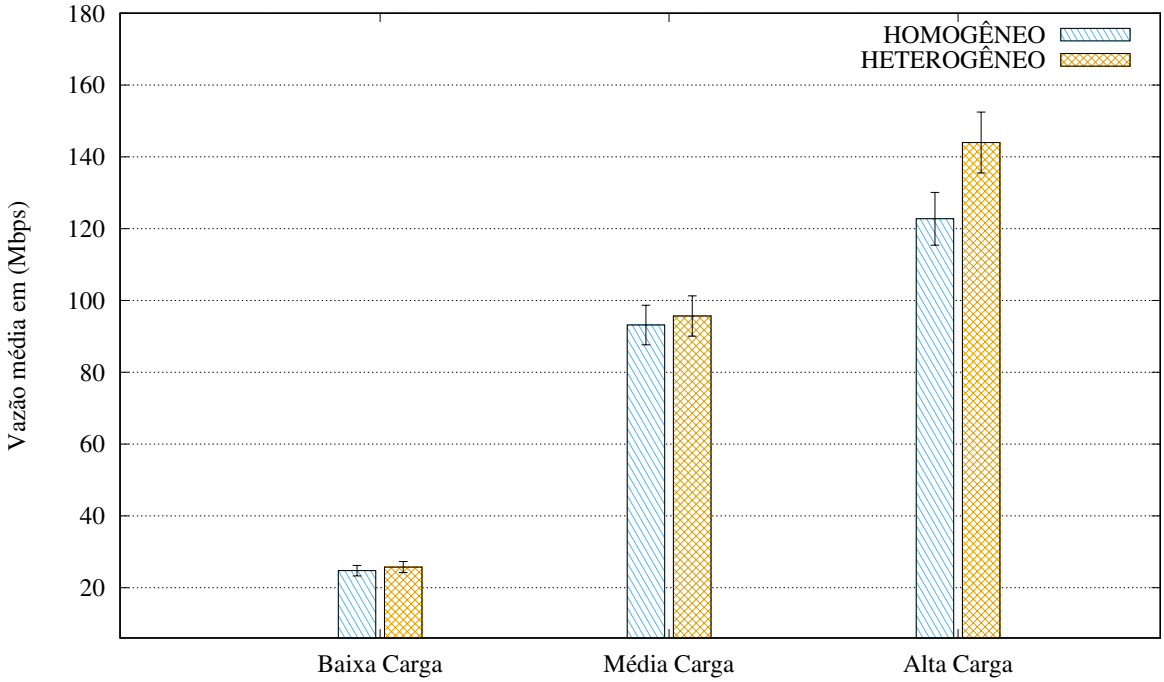
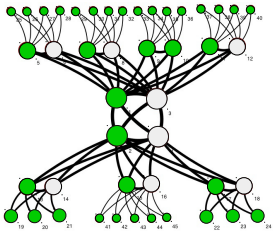


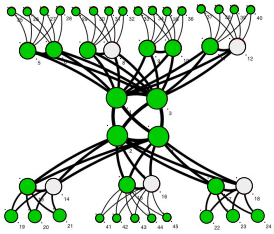
Figura 26 – Vazão média em cada cenário

entre as vazões, quando comparamos os números de consumo entre os cenários, vemos que mesmo considerando as margens de erro com um nível de confiança de 95%, temos aproximadamente 3,75% de diferença entre os cenários. Observando a curva referente a média carga na figura 25(a) e a curva correspondente à mesma carga na figura 25(b), vemos que o cenário heterogêneo obteve vantagem em termos de economia de energia comparado ao cenário homogêneo. Comparando as figuras 27(b) e 27(e), é visível que mais comutadores estão desligados no cenário heterogêneo do que no cenário homogêneo, pois devido à maior capacidade dos enlaces presentes no cenário heterogêneo, houve menor necessidade de redirecionamentos de tráfego e consequente ativação de comutadores inativos. Estes resultados indicam que em ambas topologias não houve uma limitação do núcleo para o tráfego de pacotes, apesar de o núcleo no cenário homogêneo trabalhar próximo de seu limite.

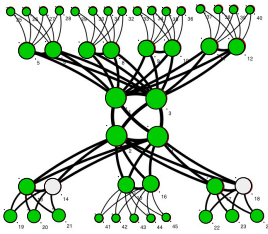
Em alta carga, também observamos que no cenário heterogêneo obtemos uma maior economia de energia em comparação ao cenário homogêneo. O cenário homogêneo, mesmo considerando a margem de erro com um nível de confiança de 95%, consumiu 6,58% a mais que o cenário heterogêneo. Analisando os comutadores comumente ligados durante a execução dos cenários observados nas figuras 27(c) e 27(f), podemos observar que no cenário homogêneo. o núcleo ficou a maior parte do tempo com todos os comutadores ligados devido a sobrecarga de seus enlaces. No cenário heterogêneo, houve momentos em que as opções de comutadores disponíveis no núcleo não foram ativadas, pois não houve sobrecarga durante 100% do tempo em seus enlaces. Em relação à vazão média, o



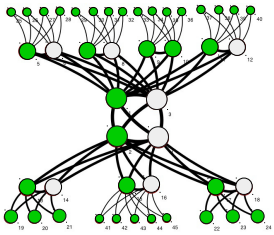
(a) Homogêneo - Carga Baixa



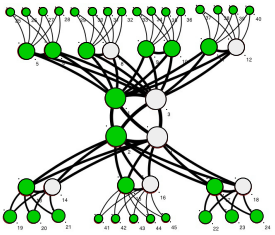
(b) Homogêneo - Carga Média



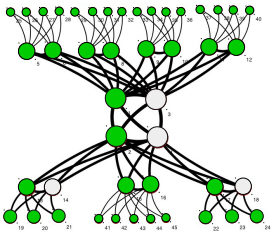
(c) Homogêneo - Carga Alta



(d) Heterogêneo - Carga Baixa



(e) Heterogêneo - Carga Média



(f) Heterogêneo - Carga Alta

Figura 27 – Comutadores ligados

cenário heterogêneo claramente permitiu que um maior volume de dados trafegasse entre os seus hospedeiros. Mesmo considerando a margem de erro para um nível de confiança de 95%, observamos na figura 26, uma vantagem para o cenário heterogêneo, apesar de nos limites das margens de erro superior e inferior existir uma diferença de apenas 3,69%. O cenário heterogêneo obteve uma vazão média de 143,989 Mbit/s contra 122,747 Mbit/s no cenário com enlaces homogêneos. Podemos concluir que os enlaces do núcleo do cenário homogêneo, atingiu o seu limite de carga e se tornou um gargalo na rede, provocando uma redução nas vazões médias obtidas. Em oposição, os enlaces do núcleo do cenário heterogêneo comportaram parcialmente os fluxos gerados, proporcionando economia de energia e maior fluxo de dados entre os comutadores.

5.4.3 Comparativo entre nossa proposta e o proposto por [Markiewicz et al., 2014]

Os autores em [Markiewicz et al., 2014] propuseram um pseudo código apresentado no Algoritmo 1. Este código define uma estratégia gulosa para encontrar o melhor caminho (que gere o menor acréscimo energético) e, caso necessário, o código atualiza a rede ligando novos caminhos. Implementamos esse pseudo código em nosso ambiente de testes *Mininet*, de modo que para cada cenário executado com a nossa proposta, os mesmos parâmetros de geração de carga, duração de um fluxo, duração de uma execução e duração de cada fluxo, foram utilizados para o algoritmo por eles proposto.

Observando a Figura 28, podemos concluir que, tanto o nosso mecanismo quanto o algoritmo proposto por [Markiewicz et al., 2014], obtiveram médias similares em baixa

Algoritmo 1: Algoritmo proposto por [Markiewicz et al., 2014]

```
1 def estrategiaGulosa(Rede): ;
2 Lista ParesDeNosAtivos ;
3 sortNodePairs ( strategy , activeNodePairs ) ;
4 para todo activeNodePairs n faça
5 |   encontreOMelhorCaminho (n) ;
6 |   updateNetworkStatus () ;
7 fim
```

carga de tráfego. Nossa proposta obteve 46,1% de economia média e o proposto pelos autores obteve 45,929%. Mesmo levando ao limite as margens de erro para mais ou para menos, considerando um nível de confiança de 95%, observamos que há empate técnico de nossa proposta comparado com o proposto pelos autores, tanto para enlaces homogêneos quanto para enlaces heterogêneos. Este empate foi possível pois, em carga baixa de tráfego, os enlaces não são sobrecarregados em nenhum cenário. Devido à ausência de sobrecarga, apenas a árvore geradora mínima permaneceu ligada em ambas abordagens em cenários com enlaces homogêneos e enlaces heterogêneos.

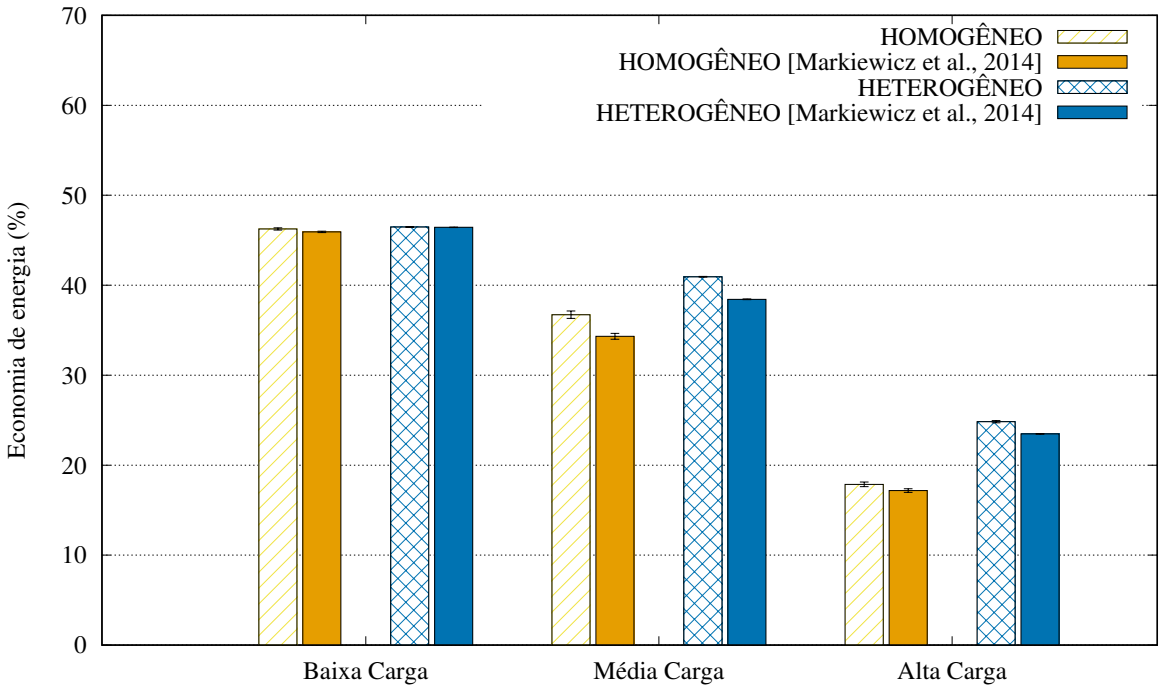


Figura 28 – Topologia típica de uma rede de campus. [Markiewicz et al., 2014]

Para os cenários executados com carga de tráfego média, observamos uma diferença entre a nossa proposta e o algoritmo proposto por [Markiewicz et al., 2014], tanto para enlaces homogêneos quanto para enlaces heterogêneos. Nossa proposta foi, em média, 2,4026% mais econômica que a proposta dos autores considerando enlaces homogêneos,

e 2,50%, em média, mais econômica considerando enlaces heterogêneos. Considerando um nível de confiança de 95%, não observamos sobreposição entre os intervalos de cada proposta. Constatamos também que nossa estratégia obteve a maior vantagem percentual em termos de economia de energia comparado ao proposto pelos autores em carga média, pois em carga alta de tráfego temos pouca margem para economizar energia e qualquer que seja a estratégia utilizada para economizar, encontrará um limite onde a maioria dos caminhos disponíveis terão que ser ligados para acomodar todos os fluxos.

Para os cenários executados com carga alta de tráfego, obtivemos apenas 0,6812% de economia a mais que o proposto por [Markiewicz et al., 2014], considerando cenários com enlaces homogêneos, e 1,35% considerando enlaces heterogêneos. Levando em consideração os limites do intervalo de confiança, para um nível de confiança de 95%, nossa proposta ainda leva uma pequena vantagem de 0,22% com enlaces homogêneos e 1,20% com enlaces heterogêneos. Esta pequena diferença se deve ao fato de que em alta carga, a maioria dos caminhos disponíveis em qualquer estratégia já estão utilizados e nossa iniciativa de desligar caminhos subutilizados praticamente não é aplicada em um ambiente sobrecarregado.

6 Trabalhos relacionados

Atualmente, o consumo de energia tem sido uma preocupação global, devido aos impactos ambientais e custos elevados de geração de energia. Tradicionalmente, os recursos naturais eram explorados como se fossem recursos infinitos e o objetivo primário de todo equipamento de tecnologia era obter o melhor desempenho, mesmo que para atingir este objetivo fosse sacrificada a economia de energia. As redes atuais não fogem a este comportamento pois foram projetadas para fornecer alta disponibilidade, resiliência e alta qualidade de serviço, trabalhando 24 horas pronta para receber picos de tráfego. Infelizmente estes benefícios vem acompanhados de uma baixa proporcionalidade de consumo de acordo com a variação da taxa de utilização da rede. Por estes motivos, os pesquisadores vem propondo estratégias para reduzir o consumo das redes.

Em vários estudos, encontramos abordagens nas quais através do gerenciamento de tráfego, pesquisadores concentram o fluxo em uma quantidade mínima de enlaces e comutadores, com o objetivo de desligar o restante da rede subutilizada. Um dos trabalhos mais importantes na busca por economia de energia em redes é [Heller et al., 2010]. No trabalho os autores propuseram o *ElasticTree*, um gerenciador de energia da rede que desliga enlaces e comutadores desnecessários e mantém ativos um subconjunto da rede original, possuindo a preocupação com a tolerância de falhas e o desempenho. Este trabalho não apenas propõe uma otimização ou um algoritmo guloso, ele implanta o gerenciador em um ambiente de testes realista.

Em [Markiewicz et al., 2014], os autores propuseram explorar os padrões diários de tráfego em uma rede típica de campus para reduzir o consumo de energia elétrica. Para atingir o objetivo proposto, foi criada uma heurística gulosa que itera entre todos os pares de nós ativos para encontrar o melhor caminho dentre os caminhos que gerarão o menor acréscimo de consumo de energia. Com base na demanda atual, o algoritmo busca a menor quantidade possível de nós utilizados para transportar o tráfego. Porém, o trabalho não avança sobre uma abordagem mais realista, e não utiliza um emulador ou um ambiente real para construir os cenários, além de não ser reativo, pois ele se utiliza das matrizes de tráfego para prever o comportamento futuro na rede, inviabilizando uma implantação real.

Em [Sasaki et al., 2015] foi proposto um método para reduzir a velocidade de enlaces subutilizados, redirecionando os tráfegos menos importantes (tráfego em tempo não real), mantendo o tráfego em tempo real em um caminho mais curto possível, de modo que este tráfego mantenha baixa latência. Para tomar a decisão de reduzir a velocidade de um enlace, o algoritmo monitora a rede em busca de um valor limite baixo para reduzir a velocidade do enlace caso a utilização fique abaixo deste limite, ou um valor limite alto, para aumentar a velocidade do enlace se a utilização ultrapassar esse limite.

Tabela 3 – Abordagens na literatura

Trabalho	Cenário	Software	Estratégia de roteamento	Foco principal
[Sasaki et al., 2015]	Simulação	Não informado	Reativa	Priorizar o tráfego real, poupando energia.
[Rahnamay-Naeini et al., 2016]	Simulação	APMonitor	Pró-ativa	Heurística de roteamento ciente do consumo.
[Heller et al., 2010]	Real	GLPK CPLEX	Pró-ativa	Implementação realista.
[Markiewicz et al., 2014]	Simulação	CPLEX	Reativa	Heurística de roteamento ciente do consumo.
[Dharmesh and Varma, 2012]	Simulação	OPNET	Reativa	Algoritmos de roteamento cientes do consumo.
[Kakadia and Varma, 2012]	Emulação	Mininet	Reativa	Mecanismo realista de redução do consumo.
Nosso trabalho	Emulação	Mininet	Reativa	Mecanismo realista de redução do consumo.

Tabela 4 – Comparação entre as soluções de engenharia de tráfego verde.

A tabela 4 contém um comparativo entre as abordagens que pretendem tornar as redes de computadores mais proporcionais em relação à energia consumida, conforme varia a carga na rede. Cada trabalho é comparado quanto ao tipo de ambiente utilizado montar os cenários, tipo de *Software* utilizado, tipo de estratégia de roteamento e o foco principal do trabalho.

7 Conclusões

Neste trabalho, nós apresentamos o funcionamento do nosso mecanismo de gerenciamento de tráfego dinâmico, implementado em um ambiente emulado, na qual exploramos as facilidades trazidas pelas redes definidas por *software* para concentrar o tráfego em uma sub-rede, mantendo uma parte da topologia sem tráfego e com seus enlaces e comutadores desligados, proporcionando economia de energia. Assim, obtemos uma maior economia de energia em relação à carga de rede, do que obteríamos se tivéssemos utilizado uma rede tradicional.

Para avaliar o desempenho de nossa proposta em termos de economia de energia, nós implementamos o pseudo código proposto por [Markiewicz et al., 2014] no ambiente do *Mininet*, emulando uma rede operacional realista e comparamos os resultados com os obtidos por nossa proposta, considerando os mesmos parâmetros de geração de carga, mesma topologia, versões dos softwares e mesmo hardware. Geramos a carga de tráfego considerando três faixas proporcionais de tráfego para cada nível de carga. Também realizamos experimentos que possuíam todos os enlaces com a mesma velocidade negociada (cenários homogêneos) e experimentos em que a velocidade negociada dos enlaces dos comutadores do núcleo da rede eram quatro vezes maiores que o restante dos enlaces (cenários heterogêneos). A faixa de carga de tráfego mais baixa, representou o tráfego noturno, a faixa de carga média representou a média diária de tráfego e, por último, a carga alta representou o tráfego de pico anual.

Os resultados mostraram que nós obtemos uma economia de energia média de 46,01% em cenários onde a carga de tráfego da rede tenha um comportamento semelhante à baixa carga do tráfego noturno. Neste cenário, conforme o esperado, obtemos a maior economia de energia comparado com redes sem mecanismos de economia de energia. Em média carga de tráfego, obtemos uma economia média de 36,72% e em alta carga obtemos uma economia média de 17,86%. Quando comparamos o nosso mecanismo com o pseudo código proposto por [Markiewicz et al., 2014], nossa proposta obteve uma vantagem média em termos de economia de energia de até 2,4026% em carga média de tráfego.

7.1 Trabalhos futuros

Um dos principais recursos que deveremos considerar futuramente é o mecanismo de detecção de falhas na rede. Devemos melhorar o algoritmo de busca de caminhos e considerar os custos de cada nó na topologia. Temos que testar nosso mecanismo em uma variedade maior de topologias. Verificar a escalabilidade de nossa proposta em ambientes realistas e com milhares de nós.

REFERÊNCIAS

[Avallone et al., 2004] Avallone, S., Guadagno, S., Emma, D., Pescape, A., and Ventre, G. (2004). D-itg distributed internet traffic generator. *First International Conference on the Quantitative Evaluation of Systems, 2004. QEST 2004. Proceedings.*

[Bianzino et al., 2010] Bianzino, A. P., Chaudet, C., Rossi, D., and Rougier, J.-L. (2010). A survey of green networking research. *IEEE Communications Surveys & Tutorials*, 14(1):3 – 20.

[Bianzino et al., 2012] Bianzino, A. P., Chaudet, C., Rossi, D., and Rougier, J.-L. (2012). A survey of green networking research. *IEEE Communications Surveys and Tutorials, Volume: 14, NO. 1, First Quarter 2012*, 14(1):3–20.

[Bolla et al., 2010] Bolla, R., Bruschi, R., Davoli, F., and Cucchietti, F. (2010). Energy efficiency in the future internet: A survey of existing approaches and trends in energy-aware fixed network infrastructures. *IEEE Communications Surveys and Tutorials (Volume: 13, Issue: 2, Second Quarter 2011)*, 13:223–244.

[Caesar et al., 2005] Caesar, M., Caldwell, D., Feamster, N., Rexford, J., Shaikh, A., and van der Merwe, J. (2005). Design and implementation of a routing control platform. In *Proceedings of the 2Nd Conference on Symposium on Networked Systems Design & Implementation-Volume 2*, pages 15–28. USENIX Association.

[Cao et al., 2003] Cao, J., Cleveland, W. S., Lin, D., and Sun, D. X. (2003). Internet traffic tends to poisson and independent as the load increases. *Springer, New York, NY*, 171(3):105–110.

[Casado et al., 2007] Casado, M., Freedman, M. J., Pettit, J., Luo, J., McKeown, N., and Shenker, S. (2007). Ethane: Taking control of the enterprise. In *ACM SIGCOMM Computer Communication Review*, volume 37, pages 1–12. ACM.

[Costa, 2016] Costa, L. C. (2016). Balanceamento de carga utilizando planos de dados openflow comerciais. *Disponível em: https://repositorio.ufjf.br/jspui/handle/ufjf/2258.*

[DCAN, 1995] DCAN (1995). Devolved control of atm networks. *Disponível em: http://www.cl.cam.ac.uk/research/srg/netos/projects/archive/dcan/. Acessado em: Janeiro de 2016.*

[Dharmesh and Varma, 2012] Dharmesh, K. and Varma, V. (2012). Energy efficient data center networks - a sdn based approach. *IBM Collaborative Academia Research Exchange (I-CARE).*

[Dixit et al., 2013] Dixit, A., Hao, F., Mukherjee, S., Lakshman, T., and Kompella, R. (2013). Towards an elastic distributed sdn controller. *ACM SIGCOMM Computer Communication Review*, 43(4):7–12.

[Doria et al., 2010] Doria, A., Salim, J. H., Haas, R., Khosravi, H., Wang, W., Dong, L., Gopal, R., and Halpern, J. (2010). Forwarding and control element separation (forces) protocol specification. Technical report.

[Ericson, 2016] Ericson, B. (2016). Ericsson energy and carbon report - including results from the first-ever national assessment of the environmental impact of ict. *Disponível em: https://www.ericsson.com/assets/local/about-ericsson/sustainability-and-corporate-responsibility/documents/ericsson-energy-and-carbon-report.pdf.*

[Feamster et al., 2014] Feamster, N., Rexford, J., and Zegura, E. (2014). The road to sdn: an intellectual history of programmable networks. *ACM SIGCOMM Computer Communication Review*, 44(2):87–98.

[Giroire et al., 2014] Giroire, F., Moulierac, J., and Phan, T. K. (2014). Optimizing rule placement in software-defined networks for energy-aware routing. *Global Communications Conference (GLOBECOM)*, pages 2523 – 2529.

[Greene, 2009] Greene, K. (2009). Tr10: Software-defined networking. *MIT Technology Review. Disponível em: http://www2.technologyreview.com/article/412194/tr10-software-definednetworking/. Acessado em: Janeiro de 2016.*

[GSMP, 2002] GSMP (2002). General switch management protocol (gsmp) v3 specification. *Disponível em: https://www.ietf.org/rfc/rfc3292.txt. Acessado em: Janeiro de 2016.*

[Gude et al., 2008] Gude, N., Koponen, T., Pettit, J., Pfaff, B., Casado, M., McKeown, N., and Shenker, S. (2008). Nox: towards an operating system for networks. *ACM SIGCOMM Computer Communication Review*, 38(3):105–110.

[Guedes et al., 2012] Guedes, D., Vieira, L. F. M., Vieira, M., Rodrigues, H., and Nunes, R. V. (2012). Redes definidas por software: uma abordagem sistêmica para o desenvolvimento de pesquisas em redes de computadores. *Minicursos do Simpósio Brasileiro de Redes de Computadores-SBRC*, 30(4):160–210.

[Heller et al., 2010] Heller, B., Seetharaman, S., Mahadevan, P., Yiakoumis, Y., Sharma, P., Banerjee, S., and McKeown, N. (2010). Elastictree: Saving energy in data center networks. *In Proceeding of the 7th USENIX conference on Networked systems design and implementation (NSDI'10).*

[JARSCHEL et al., 2014] JARSCHEL, M., ZINNER, T., HOSSFELD, T., TRAN-GIA, P., and KELLERER, W. (2014). Interfaces, attributes, and use cases: A compass for sdn. *IEEE Communications Magazine*, 52(6):210–217.

[Kakadia and Varma, 2012] Kakadia, D. and Varma, V. (2012). Energy efficient data center networks-a sdn based approach. *IBM Collaborative Academia Research Exchange.*

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

[Lakshman et al., 2004] Lakshman, T., Nandagopal, T., Ramjee, R., Sabnani, K., and Woo, T. (2004). The softrouter architecture. In *Proc. ACM SIGCOMM Workshop on Hot Topics in Networking*, volume 2004. Citeseer.

[Lantz et al., 2010] Lantz, B., Heller, B., and McKeown, N. (2010). A network in a laptop: rapid prototyping for software-defined networks. *Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Network*, 1-6(19):2523 – 2529.

[Lobato et al., 2013] Lobato, A., Figueiredo, U., and Alves, L. (2013). Redes definidas por software. *UFRJ. Disponível em: https : //www.gta.ufrj.br/grad/13$_1$/sdn/index.html Acessado em: Janeiro de 2018.*

[MACEDO et al., 2015] MACEDO, D. F., GUEDES, D., VIEIRA, L. F. M., VIEIRA, M. A. M., and NOGUEIRA, M. (2015). Programmable networks-from software-defined radio to software-defined networking. *IEEE Communications Surveys and Tutorials*, 17(2):1102–1125.

[Mahadevan et al., 2009a] Mahadevan, P., Sharma, P., Banerjee, S., , and Ranganathan, P. (2009a). A power benchmarking framework for network devices. *Proceedings of the 8th International IFIP-TC 6 Networking Conference*, pages 795 – 808.

[Mahadevan et al., 2009b] Mahadevan, P., Sharma, P., Banerjee, S., and Ranganathan, P. (2009b). Energy aware network operations. *Proceedings of the 28th IEEE International Conference on Computer Communi- cations Workshops, ser. INFOCOM'09*, pages 25–30.

[Markiewicz et al., 2014] Markiewicz, A., Tran, P. N., and Timm-Giel, A. (2014). Energy consumption optimization for software defined networks considering dynamic traffic. *Cloud Networking (CloudNet), 2014 IEEE 3rd International Conference*, pages 155 – 160.

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

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

[NUNES et al., 2014] NUNES, B. A. A., MENDONCA, M., NGUYEN, X., OBRACZKA, K., and TURLETTI, T. (2014). A survey of software-defined networking: Past, present, and future of programmable networks. *IEEE Communications Surveys and Tutorials*, 16(3):1617–1634.

[Rahnamay-Naeini et al., 2016] Rahnamay-Naeini, M., Baidya, S. S., Siavashi, E., and Ghani, N. (2016). A traffic and resource-aware energy-saving mechanism in software defined networks. In *Computing, Networking and Communications (ICNC), 2016 International Conference on*, pages 1–5. IEEE.

[Rodrigues, 2016] Rodrigues, B. B. (2016). Greensdn: energy efficiency in software defined networks. *Dissertação (Mestrado em Sistemas Digitais) - Escola Politécnica, University of São Paulo, São Paulo.*

[Sasaki et al., 2015] Sasaki, S., Ogura, K., Bista, B. B., and Takata, T. (2015). A proposal of qos-aware power saving scheme for sdn-based networks. *2015 18th International Conference on Network-Based Information Systems.*

[Tennenhouse et al., 1997] Tennenhouse, D. L., Smith, J. M., Sincoskie, W. D., Wetherall, D. J., and Minden, G. J. (1997). A survey of active network research. *IEEE communications Magazine*, 35(1):80–86.

[Zilberman et al., 2015] Zilberman, N., Watts, P. M., Rotsos, C., and Moore, A. W. (2015). Reconfigurable network systems and software-defined networking. *Proceedings of the IEEE*, 103(7):1102–1124.

A Classe Nó

```
# -*- coding: utf-8 -*-
"""
Created on Tue May 16 22:30:47 2017

@author: wallace

obs ao add um adjacente, armazenar INFO de porta
"""

class No(object):

    def __init__(self, identificacao, estado, tipo, consumo_basico=146,
consumo_1gb=0.87, consumo_100mb=0.18, velocidade=1000):
        self.identificacao = identificacao
        self.estado = estado #ligado ou desligado
        self.estado_operacao = {} #baixo, normal, travado, sobresecarga
        self.tipo = tipo #acesso, encaminhamento
        self.adjacentes = []
        self.consumo_basico=consumo_basico #placa mae fan e etc
        self.consumo_1gb=consumo_1gb
        self.consumo_100mb=consumo_100mb
        self.velocidade=velocidade #velocidade das conexoes do no.

    def get_identificacao(self):
        return self.identificacao

    def get_estado(self):
        return self.estado


    def get_adjacentes(self):
        return self.adjacentes

    def get_tipo(self):
        return self.tipo

    def get_consumo_basico(self):
```

```
        return self.consumo_basico

    def get_consumo_1gb(self):
        return self.consumo_1gb

    def get_consumo_100mb(self):
        return self.consumo_100mb

    def get_estado_operacao(self, porta):
        if porta in self.estado_operacao:
            return self.estado_operacao[porta]

    def set_tipo(self, tipo):
        #fazer verificacao de tipo antes de atribuir
        self.tipo=tipo

    def chavear_estado(self):
        if self.estado == "ligado":
            self.estado="desligado"
        else:
            self.estado="ligado"

    def ligar(self):
        self.estado = "ligado"

    def desligar(self):
        self.estado = "desligado"

    def add_adjacente(self, objeto_no):
        self.adjacentes.append(objeto_no)

    def add_lista_adjacentes(self, lista):
        self.adjacentes=lista

    def set_estado_operacao(self, operacao, porta):
        self.estado_operacao[porta]=operacao
        #print "estado", self.estado_operacao[porta]

    def exibir_adjacentes(self):
```

```
        for x in self.adjacentes:
            print x.get_identificacao()
```

B Classe Host

```
# -*- coding: utf-8 -*-
"""
Created on Sun Jun 25 18:38:16 2017

@author: wallace
"""

class Host(object):
    def __init__(self, mac, switch_acesso,
porta,estado,velocidade=1000):
        self.mac = mac
        #ativo inativo
        self.estado = estado
        #switch em que o host se conecta à rede
        self.switch_acesso=switch_acesso
        #velocidade das conexoes do no.
        self.velocidade=velocidade
        #porta em que o switch de acesso se comunica com o host.
        self.porta=porta

    def get_mac(self):
        return self.mac

    def get_estado(self):
        return self.estado


    def get_switch_acesso(self):
        return self.switch_acesso

    def set_estado(self, estado):
        self.estado=estado

    def set_switch_acesso(self, switch_acesso):
        self.switch_acesso=switch_acesso

    def chavear_estado(self):
```

```
        if self.estado == "ativo":
            self.estado="inativo"
        else:
            self.estado="ativo"
```

C Classe Grafo

```
# -*- coding: utf-8 -*-
"""
Created on Tue May 16 23:58:47 2017

@author: wallace
"""
class Grafo(object):

    def __init__(self, identificacao, lista_nos_acessos):
        self.identificacao = identificacao
        #todos os objetos tipo no e seus obj nos adjacentes
        self.nos = {}
        self.hosts={}
        #lista de ocorrencias por par comunicante
        self.lista_ocorrencias={}
        #lista geral de ocorrencias da combinacao de todos os pares
        self.geral_ocorrencias={}
        #lista de nos de acesso
        self.nos_acessos=lista_nos_acessos
        #lista de nos que nao serao desligados
        self.manter_ligado=[]
        #lista que armazenara as opcoes em cada caminho
        self.lista_opcoes=[]
        #lista gerada de todos os caminhos
        self.multiplos_caminhos=[]
        #lista com os caminhos que foram decididos
p os pares comunicantes
        self.caminhos_decididos=[]
        #permite armazenar os nos que devem ser evitados na tomada de
        #decisao por caminhos
        self.nos_sobrecarga=[]
        #aqui serao armazenadas as informacoes dos fluxos ativos,
        #contendo mac origem destino e caminho
        self.fluxos_ativos={}
        #consumo em watts instantaneo de todos os nos ligados na rede
        self.consumo_instantaneo=0
        #consumo total da rede considerando nos ligados e desligados
```

```
        self.consumo_total=0
        self.lista_total=[]
        self.lista_instantaneo=[]
        Timer(1, self._timer_func, recurring=True)
```

D Método de cálculo de consumo

```
/**
* Funcao que executa o calculo de consumo baseado no modelo da rede.
*/
   def calcular_consumo_no(self, no):

        num_links=len(self.nos[no].get_adjacentes())

        return self.nos[no].get_consumo_basico()+num_links*self.nos[no]
.get_consumo_1gb()

    def calcular_consumo_total(self):
        consumo_total=0
        consumo_ligados=0
        consumo_desligados=0

        for x in self.nos:

            if self.nos[x].get_estado()=="ligado":
                consumo_ligados+=self.calcular_consumo_no(x)
            else:
                consumo_desligados+=self.calcular_consumo_no(x)
        consumo_total=consumo_ligados+consumo_desligados

        self.consumo_instantaneo=consumo_ligados
        self.consumo_total=consumo_total
        return [consumo_total,consumo_ligados,
(consumo_desligados/consumo_total)*100]
```

E Função que captura eventos do componente Discovery

```
/**
* Funcao que lida com o evento discovery do nucleo do POX.
*/
#Evento capturado quando o componente discovery descobre um novo link
def _handle_LinkEvent (event):
  # armazena os dados do link (sw1, porta1)(sw2, porta2)
  (dp1,p1),(dp2,p2) = event.link.end

  if dp1 not in lista_links:
     lista_links[dp1]=[]

  lista_links[dp1].append([dp1,dp2, p1, p2])

  #trabalhando com a classe grafos:
  #montando o grafo e nos adjacentes

  grafo.add_no(dp1, dp2, "desligado", "indefinido")
```

F Função que captura eventos do componente host-event

```
/**
* Funcao que lida com o evento host_event do nucleo do POX.
*/
def _handle_HostEvent (event):
    #armazenamos na variável h a String do endereço MAC
    h = str(event.entry.macaddr)
    #armazenamos em s a ID do comutador de acesso do hospedeiro
    s = event.entry.dpid
    #armazenamos em p a porta de saída do comutador para o hospedeiro.
    p = event.entry.port
    lista_hosts[h]=[ s, p]
    log.debug("Link: Host | Switch | Porta \%s", lista_hosts )
```

G Registro de eventos no núcleo do POX

```
/**
* Registro dos eventos no nucleo do POX.
*/
core.openflow.addListenerByName ("FlowStatsReceived",
self._handle_flowstats_received);

core.openflow.addListenerByName   ("PortStatsReceived",
self._handle_portstats_received);
}
```

H Função que captura eventos de estatísticas de portas

```
/**
* Funcao que trata as estatisticas recebidas pelo evento PortStats.
*/
def _handle_portstats_received (self, event):
    #Converte as estatísticas das portas em uma lista
    stats = flow_stats_to_list(event.stats)
    velocidade=0
    #Para cada porta do comutador faça
    for f in event.stats:
        if event.connection.dpid not in self.stats_anterior:
          #Cria uma referência para último valor de estatística da porta
            self.stats_anterior[event.connection.dpid]={}
        else:
#Se a porta ainda não foi adicionada a uma referência anterior adicione
            if f.port_no not in
self.stats_anterior[event.connection.dpid]:
self.stats_anterior[event.connection.dpid][f.port_no]=
f.tx_bytes
            else:
#Calcule a largura de banda consumida atual utilizando a referencia
#anterior e adicione a largura de banda atual na referencia
#para o próximo calculo
                velocidade=(((f.tx_bytes-self.stats_anterior
[event.connection.dpid][f.port_no])/1024.0)/1024)*8
                self.stats_anterior[event.connection.dpid][f.port_no]=
f.tx_bytes
```

I Publicação do evento intermedEvent

```
/**
* Chamada do metodo que publica o evento intermedEvent.
*/
self.publishintermedEvent(velocidade/max, "bloqueado",
event.connection.dpid, f.port_no);
}
```

J Algoritmo de redirecionamento

```
# Primeiro - Adicionar regras nos switches intermediarios do novo caminho
        for x in range(0, len(add_regras)):

                porta1=20
                porta2=20
                boa_carga=True

                msg = of.ofp_flow_mod()
                msg2 = of.ofp_flow_mod()

                msg = of.ofp_flow_mod()
                msg.match.dl_src = EthAddr(host1)
                msg.match.dl_dst = EthAddr(host2)

                msg2 = of.ofp_flow_mod()
                msg2.match.dl_dst = EthAddr(host1)
                msg2.match.dl_src = EthAddr(host2)

                for y in lista_links[add_regras[x]]:
                    if y[1]==novo_caminho[novo_caminho.
index(add_regras[x])+1]:
                        porta1=y[2]
                    if y[1]==novo_caminho[novo_caminho.
index(add_regras[x])-1]:
                        porta2=y[2]

                msg.actions.append(of.ofp_action_output(port=porta1))
                msg2.actions.append(of.ofp_action_output(port=porta2))

                ##verificacao de estado do switch que sera utilizado
                boa_carga=grafo.verificar_estado_caminho
(desligar_caminho, novo_caminho, porta1, porta2)

                #so executa se o sw e a porta nao esta em estado
bloqueado ou sobrecarga
                if boa_carga and desligar_caminho!=novo_caminho:
```

```
                    core.openflow.connections[add_regras[x]].send(msg)
                    core.openflow.connections[add_regras[x]].send(msg2)

                    log.debug("AQUI Switch n: \%s" \% add_regras[x])
                    log.debug("AQUI Porta1: \%s porta2 \%s",porta1, porta2)

        #so executa se o sw e a porta nao
esta em estado bloqueado ou sobrecarga
        if boa_carga and desligar_caminho!=novo_caminho:
            #### Modificar a saida do primeiro switch adjacente aos
switches q tiveram novas regras###
            porta1=20
            porta2=20

            msg = of.ofp_flow_mod()
            msg2 = of.ofp_flow_mod()

            msg.command = of.OFPFC_MODIFY
            msg.match.dl_src = EthAddr(host1)

            msg2.command = of.OFPFC_MODIFY
            msg2.match.dl_dst = EthAddr(host1)


            for x in lista_links[mod1]:
                if x[1]==novo_caminho[1]:
                    porta1=x[2]
            porta2=lista_hosts[host1][1]
            msg.actions.append(of.ofp_action_output(port=porta1))
            msg2.actions.append(of.ofp_action_output(port=porta2))

            core.openflow.connections[mod1].send(msg)
            core.openflow.connections[mod1].send(msg2)



            log.debug("AQUI Switch n: \%s" \% mod1)
            log.debug("AQUI Porta1: \%s porta2 \%s",porta1, porta2)
```

```
            #### Modificar a saida do ultimo switch adjacente aos
switches q tiveram novas regras###
            porta1=20
            porta2=20

            msg = of.ofp_flow_mod()
            msg2 = of.ofp_flow_mod()

            msg.command = of.OFPFC_MODIFY
            msg.match.dl_src = EthAddr(host1)

            msg2.command = of.OFPFC_MODIFY
            msg2.match.dl_dst = EthAddr(host1)

            porta1=lista_hosts[host2][1]

            for x in lista_links[mod2]:
                if x[1]==novo_caminho[-2]:
                    porta2=x[2]

            msg.actions.append(of.ofp_action_output(port=porta1))
            msg2.actions.append(of.ofp_action_output(port=porta2))
            core.openflow.connections[mod2].send(msg)
            core.openflow.connections[mod2].send(msg2)

            log.debug("AQUI Switch n: \%s" \% mod2)
            log.debug("AQUI Porta1: \%s porta2 \%s",porta1, porta2)


            ##Desligar regras inuteis apos a migracao

            for x in range(0, len(desligar)):
                msg3 = of.ofp_flow_mod()
                msg3.command = of.OFPFC_DELETE
                msg3.match.dl_src = EthAddr(host1)
                core.openflow.connections[desligar[x]].send(msg3)

                msg3 = of.ofp_flow_mod()
                msg3.command = of.OFPFC_DELETE
```

```
                msg3.match.dl_src = EthAddr(host2)
                core.openflow.connections[desligar[x]].send(msg3)
```