COPPE/UFRJ

ECO-ALOC: ALOCAÇÃO ENERGETICAMENTE EFICIENTE DE RECURSOS PARA ROTEADORES DE SOFTWARE EM CLUSTER

Carlo Fragni

Dissertação de Mestrado apresentada ao Programa de Pós-graduação em Engenharia Elétrica, COPPE, da Universidade Federal do Rio de Janeiro, como parte dos requisitos necessários à obtenção do título de Mestre em Engenharia Elétrica.

Orientador: Luís Henrique Maciel Kosmalski Costa

Rio de Janeiro
Março de 2011

ECO-ALOC: ALOCAÇÃO ENERGETICAMENTE EFICIENTE DE RECURSOS PARA ROTEADORES DE SOFTWARE EM CLUSTER

Carlo Fragni

DISSERTAÇÃO SUBMETIDA AO CORPO DOCENTE DO INSTITUTO ALBERTO LUIZ COIMBRA DE PÓS-GRADUAÇÃO E PESQUISA DE ENGENHARIA (COPPE) DA UNIVERSIDADE FEDERAL DO RIO DE JANEIRO COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE MESTRE EM CIÊNCIAS EM ENGENHARIA ELÉTRICA.

Examinada por:

Prof. Luís Henrique Maciel Kosmalski Costa, Dr.

Prof. Aloysio de Castro Pinto Pedroza, Dr.

Profa. Luci Pirmez, D.Sc.

RIO DE JANEIRO, RJ – BRASIL
MARÇO DE 2011

Fragni, Carlo

ECO-ALOC: Alocação Energeticamente Eficiente de Recursos para Roteadores de Software em Cluster/Carlo Fragni. – Rio de Janeiro: UFRJ/COPPE, 2011.

XIV, 93 p.: il.; 29, 7cm.

Orientador: Luís Henrique Maciel Kosmalski Costa

Dissertação (mestrado) – UFRJ/COPPE/Programa de Engenharia Elétrica, 2011.

Referências Bibliográficas: p. 88 – 93.

1. Sistemas de Comunicação. 2. Redes Verdes. 3. Virtualização de Redes. 4. Gerenciamento de Recursos Virtuais. 5. Roteadores de Software. 6. Internet do Futuro. I. Costa, Luís Henrique Maciel Kosmalski. II. Universidade Federal do Rio de Janeiro, COPPE, Programa de Engenharia Elétrica. III. Título.

Dedicado à minha família.

Agradecimentos

Agradeço a todos que me ajudaram a chegar até aqui.

Em primeiro lugar aos meus pais, que sempre me deram amor, me passaram meus valores e me formaram como ser humano sempre me dando o melhor que podiam.

À Lívia, que sempre consegue arrancar-me um sorriso e me presenteou com milhares de momentos felizes.

À minha família pelo carinho, ajuda e incentivo ao longo de toda minha vida.

Aos meus amigos, sempre ao meu lado e com quem compartilho diversas boas lembranças.

Ao meu orientador Luís Henrique pela amizade, apoio e pela infindável paciência e presteza na minha formação do mestrado, que espero algum dia poder compensar.

Aos professores Aloysio Pedroza e Luci Pirmez por terem aceitado o convite de participarem da banca examinadora.

Aos amigos Marcelo, Felipe Grael, Rodrigo Rodovalho, Lyno, Miguel, Igor e Natália pela amizade e grande ajuda ao longo de todo o mestrado.

A todo o grupo do GTA que me acolheu ao longo de todo o mestrado, com quem compartilhei horas de discussão em inúmeros assuntos durante os almoços e lanches, dos mais complexos aos mais engraçados, que sempre me apoiou e onde fiz diversos bons amigos.

Ao Professor José Gabriel R. C. Gomes por gentilmente ceder seu tempo para um curso expresso de Otimização.

A todos os meus professores que contribuíram com seus conhecimentos para a minha formação.

À CAPES, CNPq, FAPERJ e FINEP/FUNTTEL pelo financiamento desta pesquisa.

Resumo da Dissertação apresentada à COPPE/UFRJ como parte dos requisitos necessários para a obtenção do grau de Mestre em Ciências (M.Sc.)

ECO-ALOC: ALOCAÇÃO ENERGETICAMENTE EFICIENTE DE RECURSOS PARA ROTEADORES DE SOFTWARE EM CLUSTER

Carlo Fragni

Março/2011

Orientador: Luís Henrique Maciel Kosmalski Costa

Programa: Engenharia Elétrica

A ossificação da Internet despertou a necessidade na comunidade científica de desenvolver uma nova arquitetura para a Internet do Futuro e diversas iniciativas lançam o uso de virtualização de redes em roteadores de software baseados na plataforma de hardware x86 como tendência. Os roteadores de software, entretanto, sofrem com problemas de desempenho. Surgem então os roteadores de software em cluster, uma abordagem prática para obter a flexibilidade dos roteadores de software e superar seus problemas de escalabilidade. Além disso, os roteadores de software em cluster abrem novas possibilidades para a economia de energia, quesito que deverá inegavelmente estar presente na Internet do Futuro, tanto por questões de sustentabilidade ambiental quanto por fatores econômicos. Este trabalho propõe o ECO-ALOC, um mecanismo de alocação de recursos energeticamente eficiente para roteadores de software baseados em cluster. O ECO-ALOC é formado por dois módulos: o primeiro provê economia de energia de alta granularidade através do controle da freqüência de operação das CPUs, enquanto o segundo provê economia de energia a longo prazo através da migração de redes virtuais para consolidar a carga e desativar servidores ociosos dentro do cluster. É desenvolvido um modelo de roteador de software em cluster e um simulador baseado no modelo desenvolvido para avaliação da proposta. Para enriquecer a avaliação, também desenvolve-se um mecanismo alternativo ao ECO baseado em uma técnica de otimização. Os mecanismos propostos são avaliados utilizando dados baseados em tráfego real provenientes de enlaces de *backbone* da CAIDA e da RNP. Também é proposto o ECOv2, versão alternativa do ECO para melhorar a utilização de buffer. Os resultados mostram que o ECO-ALOC provê até 93% de economia de energia em um cenário de baixa carga.

Abstract of Dissertation presented to COPPE/UFRJ as a partial fulfillment of the requirements for the degree of Master of Science (M.Sc.)

ECO-ALOC: ENERGY EFFICIENT RESOURCE ALOCATION FOR CLUSTER-BASED SOFTWARE ROUTERS

Carlo Fragni

March/2011

Advisor: Luís Henrique Maciel Kosmalski Costa

Department: Electrical Engineering

The Internet ossification has raised in the scientific community the need to develop a new architecture for the Future Internet and many initiatives launch the usage of network virtualization over x86-platform-based software routers as a trend. Software routers, however, have performance issues. Therefore, cluster-based software routers raise providing software router flexibility and overcoming their performance issues. Moreover, cluster-based software routers enable new approaches for energy savings, an inevitable feature on the Future Internet both for economical and environmental matters. This work proposes ECO-ALOC, an energy-efficient resource allocation mechanism for cluster-based software routers. ECO-ALOC has two modules: the first module provides fine-grained energy consumption control by switching CPU operation frequencies, while the second module provides long-term power savings by using virtual network migration to consolidate the load and shut idle servers down inside the cluster. A cluster-based software router model is developed, and also a simulator based on the developed model in order to evaluate the proposed mechanism. To improve the evaluation, an alternative mechanism to ECO is developed based on an optimization technique. The proposed mechanisms are evaluated using data based on CAIDA and RNP links real backbone traffic. It is also proposed ECOv2, an alternative ECO version to improve buffer usage. Results show that ECO-ALOC provides up to 93% energy saving on a low load scenario.

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Abreviaturas

ALOC	*Automatic Load Organizer for Cluster*, p. 5
CABO	*Concurrent Architectures are Better than One*, p. 8
CAIDA	*Cooperative Association for Internet Data Analysis*, p. 44
CPU	*Central Processing Unit*, p. 3
DPM	*Distributed Power Management*, p. 12
ECOv2	*Efficient/Cautious/Oversaving*, p. 78
ECO	*Efficient/Cautious/Otiose*, p. 5
FTP	*File Transfer Protocol*, p. 45
GPU	*Graphics Processing Unit*, p. 10
HTTPS	*Hypertext Transfer Protocol Secure*, p. 45
HTTP	*Hypertext Transfer Protocol*, p. 45
IDT	*IDle Threshold*, p. 33
IP	*Internet Protocol*, p. 7
ISP	*Internet Service Provider*, p. 44
MDL	Mecanismo de Dinâmica Lenta, p. 5
MDR	Mecanismo de Dinâmica Rápida, p. 5
MG	Minas Gerais, p. 65
NAPI	*New Aplication Programing Interface*, p. 14
NAT	*Network Address Translation*, p. 7
OVT	*OVerload Threashold*, p. 32

QoS *Quality of Service*, p. 8

RAM *Random Access Memory*, p. 19

RBA *Role-Based Architecture*, p. 8

RNP Rede Nacional de ensino e Pesquisa, p. 45

RV Rede Virtual, p. 27

SA *Simulated Annealing*, p. 55

SLA *Service Level Agreement*, p. 3

SP São Paulo, p. 65

TI Tecnologia da Informação, p. 85

WMA *Window Moving Average*, p. 29

Capítulo 1

Introdução

A Internet é um grande sucesso. Desde a sua concepção, passou de uma pequena rede de testes para a maior rede de comunicação do mundo, unindo pessoas através de todo o mundo. Apesar de todo esse sucesso, a Internet encontra-se "ossificada", visto que não é possível criar novos serviços ou inovação no núcleo da rede de forma fácil [2–4]. Para continuar com o desenvolvimento de novos serviços e não ser freada pelas limitações da Internet atual, a comunidade científica está desenvolvendo diversas iniciativas [5–8] para construir uma nova arquitetura para a Internet do Futuro que esteja apta a suportar os requisitos dos serviços atuais e dos que irão existir no futuro. Dentre as diversas iniciativas para a construção de uma nova arquitetura para a Internet, existem dois tipos de abordagens: a purista [9, 10] e a pluralista [11, 12]. A abordagem purista propõe novas arquiteturas para a Internet em que existe apenas uma rede genérica capaz de lidar com todos os requisitos de todos os serviços. A abordagem pluralista, por outro lado, propõe novas arquiteturas onde há a coexistência de múltiplas redes, cada uma com uma pilha de protocolos própria e específica para atender os requisitos de um serviço ou tipo de serviço.

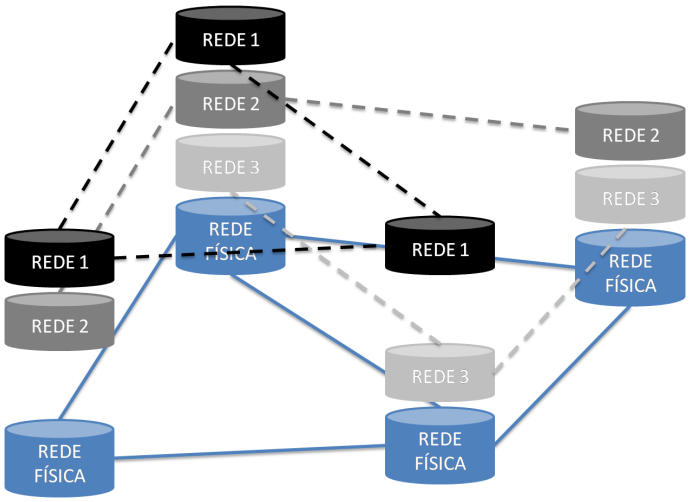


Figura 1.1: Exemplo de arquitetura pluralista.

1.1 Motivação

Um exemplo de arquitetura pluralista é ilustrado na figura 1.1. Para obter múltiplas pilhas de protocolos em paralelo, diversas iniciativas de arquiteturas pluralistas utilizam a virtualização [8, 13–17], uma técnica que permite o compartilhamento de recursos computacionais [18]. A técnica de virtualização divide um ambiente computacional real em diversos ambientes computacionais virtuais, que são isolados entre si e interagem com a camada computacional subjacente da mesma maneira que seria esperada ao interagir com um ambiente computacional não virtualizado. Uma comparação entre um ambiente virtualizado e um ambiente não virtualizado é mostrada na figura 1.2. Do lado esquerdo da figura encontra-se um ambiente computacional tradicional, composto pelo hardware (representada por um computador pessoal), sistema operacional (representado pelo Linux) e aplicações (representados pelo Skype, Firefox e Gimp). Do lado direito da figura encontra-se um ambiente computacional virtualizado, onde com a adição de uma camada de virtualização (representado pelo VMware) sobre o hardware é possível executar múltiplos sistemas operacionais ao mesmo tempo (representados por Linux, Windows e OS X), cada um com suas próprias aplicações.

No contexto de arquiteturas pluralistas que utilizem virtualização, os recursos de um roteador (processador, memória, disco, filas, banda, etc) podem ser considerados o ambiente computacional a ser virtualizado. Um conjunto de roteadores e enlaces virtuais é chamado de *rede virtual*. Dessa forma, utilizando a técnica de virtualização, arquiteturas pluralistas são capazes de possuírem múltiplas redes virtuais concorrentes, cada uma utilizando uma pilha de protocolos própria, compartilhando um mesmo substrato de infraestrutura física de rede, como ilustrado na Figura 1.1. Na Figura 1.1, cada cor representa uma rede diferente, com uma pilha de protocolos própria, e todas as redes virtuais compartilham os recursos do substrato comum de infraestrutura de rede, representado em azul.

O compartilhamento de recursos do substrato físico introduz uma série de desafios interessantes. Quando os recursos do substrato físico tornam-se escassos, há a necessidade de distribuir recursos entre as redes virtuais de forma a atendê-las segundo critérios que podem levar em conta aspectos de justiça, como o uso justo de recursos como banda ou processamento, ou critérios que levem em conta políticas econômicas, como fornecer mais recursos para uma rede virtual ou outra dependendo do quanto cobra-se para prover a infraestrutura de rede e do *Service Level Agreement* (SLA) estabelecido. Uma outra questão é o que fazer com os roteadores virtuais quando a máquina física onde eles estão executando precisa ser subitamente desligada. Embora as ferramentas de virtualização forneçam suporte à migração de máquinas virtuais, a realocação de um roteador virtual para outra máquina física

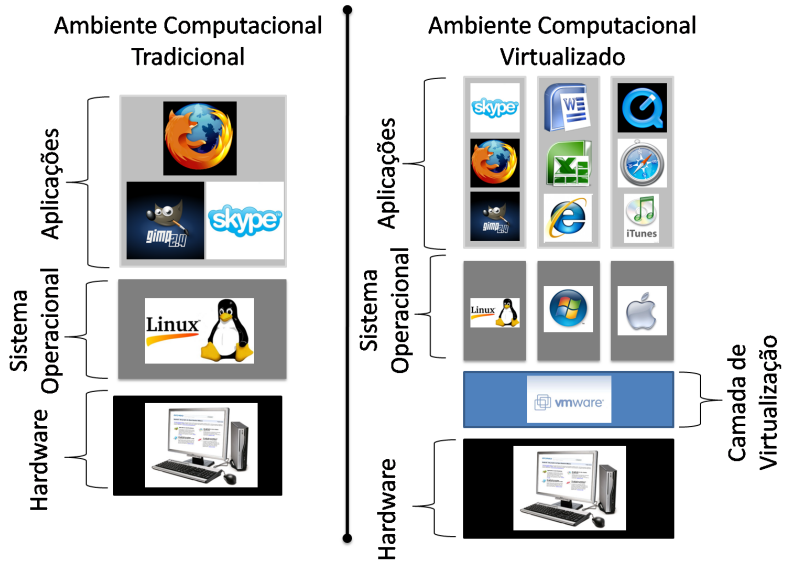


Figura 1.2: Exemplo de ambientes computacionais tradicional e virtualizado.

pode ser uma tarefa penosa, visto que a máquina de destino deve garantir o mesmo nível de conectividade que a máquina anterior fornecia, desde quesitos como capacidade do enlace até quesitos como alcançabilidade de destinos, e outros recursos como CPU ou memória.

Uma outra questão para a evolução das propostas de Internet do Futuro é a adequação dos roteadores aos novos requisitos, em especial a adição de flexibilidade aos roteadores. Os roteadores convencionais priorizam desempenho, mas são baseados em plataformas específicas de hardware e em software proprietário, prejudicando a experimentação de novas funcionalidades na rede. Como alternativa, surgem os roteadores de software, roteadores baseados em hardware convencional x86 e em sistemas operacionais convencionais, geralmente Linux, que possuem as vantagens de serem mais baratos, por utilizarem hardware comum, e de serem fáceis de programar, dada a ampla oferta de ambientes de desenvolvimento e mão de obra qualificada para o desenvolvimento na plataforma x86. Embora possuam essas vantagens, os roteadores de software possuem problemas de desempenho e não atingem taxas tão altas de encaminhamento de pacotes quanto os roteadores baseados em hardware dedicado [19]. Uma alternativa melhor é a utilização de roteadores de software em cluster, visto que ao agregar hardware convencional em clusters consegue-se superar as limitações de desempenho e manter a facilidade para programar novos protocolos e funcionalidades [20].

Os roteadores de software em cluster provêem às redes virtuais uma infraestrutura adequada de rede, mas seu uso gera o problema da distribuição dos recursos entre as redes virtuais. Visto que o consumo energético tornou-se um fator limitante para a escalabilidade e crescimento da Internet [21], outra vantagem dos roteado-

res de software em cluster é a possibilidade de gerenciamento mais eficiente do gasto energético através do desligamento e religamento dos servidores do cluster de acordo com as demandas de tráfego. Isso torna-se de suma importância frente ao fato das redes de telecomunicações já serem, há algum tempo, grandes consumidoras de energia e tenderem a crescer ainda mais seus consumos. Para ilustrar, já em 2002, os gastos energéticos dos EUA com telecomunicações estavam estimados entre 5 e 24 TWh/ano [22] enquanto a Telecom Italia gastou mais de 2 TWh em 2006 [23], aproximadamente 1% do total da demanda energética italiana. As projeções futuras também são alarmantes, segundo artigo recente na Scientific American [24], a rede global de telecomunicações produz anualmente 250 milhões de toneladas de dióxido de carbono, equivalente a aproximadamente a emissão de carbono de 50 milhões de carros (valor que corresponde a 20% dos carros dos EUA). Para absorver essa quantidade de carbono, seria necessária uma floresta do tamanho do Reino Unido. A questão do consumo energético da rede é tão grande que foi formado um consórcio chamado Green Touch, formado por governos, empresas e pesquisadores para lidar com a questão. O objetivo do consórcio é chegar mais próximo ao limite teórico do gasto energético para a transmissão de dados. Segundo uma página do site oficial do consórcio [25], um usuário consome 25 W para a transmissão de seus dados utilizando equipamentos de estado da arte em telecomunicações, enquanto o limite teórico seria de 1 mW, ou 25.000 vezes menos. O consórcio objetiva atingir a meta de cortar em 1000 vezes o gasto energético com redes de telecomunicação em um universo de 5 anos. Um outro estudo [26], indica que, caso o consumo médio diário de mídia de uma pessoa não se modifique e a tendência do conteúdo de vídeo ir para a Internet e ser consumida em forma de *stream* se mantenha, em 2030 o consumo energético para fornecer mídia para a população mundial será equivalente a 1.175 GW. Dessa forma, é fácil verificar que qualquer que seja a proposta de arquitetura de Internet do Futuro, é imprescindível que esta se preocupe com o consumo energético, tanto por questões econômicas quanto por questões ambientais.

1.2 Objetivo

Conforme explicitado na seção anterior, a Internet do Futuro está sendo desenvolvida e a questão energética está no centro das preocupações de sustentabilidade para o planeta. Visando atender a necessidade de um mecanismo de alocação de recursos para arquiteturas pluralistas de Internet do Futuro que se baseiem em roteadores de software em cluster e tendo a questão energética em mente, neste trabalho propõe-se o ECO-ALOC, um mecanismo de alocação de recursos energeticamente eficiente para roteadores de software em cluster. O ECO-ALOC possui um Mecanismo de Dinâmica Rápida (MDR) para lidar, em alta granularidade, com a

economia de energia dentro de cada servidor que compõe o cluster. O ECO-ALOC também possui um Mecanismo de Dinâmica Lenta (MDL) para lidar com a economia de energia a longo prazo dentro do cluster como um todo. O MDL também é responsável pela distribuição das redes virtuais entre os servidores que compõem o cluster. O MDR proposto, chamado *Efficient/Cautious/Otiose* (ECO), consiste em uma heurística para ajustar o tempo que a CPU gasta em três estados: *Efficient*, que provê a melhor eficiência energética em termos de watts por pacote encaminhado, *Cautious*, que provê a capacidade máxima de encaminhamento de pacotes, e *Otiose*, que provê o consumo energético mais baixo. A idéia básica é otimizar a alocação de recursos entre as redes virtuais de forma a atender as demandas de tráfego e reduzir o consumo energético da CPU. Como MDL, propõe-se o mecanismo *Automatic Load Organizer for Cluster* (ALOC). ALOC migra roteadores virtuais de servidores sobrecarregados, consolida os roteadores virtuais nos servidores mais eficientes do ponto de vista energético, e desliga servidores ociosos para economizar energia.

Além da proposta do ECO-ALOC, outra contribuições deste trabalho são: a identificação de fatores relevantes para o consumo energético de roteadores de software em cluster e de parâmetros configuráveis para seu gerenciamento energético, a investigação de roteadores de software em cluster como plataforma flexível para economizar energia em virtualização de redes e o desenvolvimento de um simulador para o estudo do problema de alocação de recursos e economia de energia em roteadores de software em cluster.

Para avaliar o ECO-ALOC, é desenvolvido um simulador de roteador de software em cluster. Também é desenvolvido um MDR alternativo baseado na técnica *Simulated Annealing* de otimização não-linear para auxiliar na avaliação do desempenho do ECO. Os resultados mostram que, sob tráfego baseado em dados reais de roteadores do *backbone*, o mecanismo proposto consegue reduzir o gasto energético em até 93% sem aumentar a perda de pacotes. No cenário de pior caso, o mecanismo proposto foi capaz de praticamente eliminar a perda de pacotes, economizar 48% da energia consumida e reduzir em 50% o uso do buffer quando comparado ao cluster sem nenhum mecanismo de gerenciamento de recursos ou economia de energia.

1.3 Organização da Dissertação

Este trabalho está organizado como descrito a seguir. O Capítulo 2 apresenta o estado da arte e apresenta os trabalhos relacionados a este. O Capítulo 3 apresenta o ambiente do roteador de software em cluster. Nele, discutem-se os modelos de arquitetura de roteadores de software em cluster, além de analisar os fatores de consumo energético e apresentar formas de configurar o desempenho e o consumo energético dos roteadores de software em cluster. O Capítulo 4 descreve o ECO-ALOC, meca-

nismo de alocação energeticamente eficiente de recursos para roteadores de software em cluster. O Capítulo 5 apresenta, em detalhes, o modelo do problema de alocação de recursos em roteadores de software em cluster e descreve o simulador desenvolvido para representar esse modelo e avaliar os mecanismos propostos. O Capítulo 6 apresenta um mecanismo de alocação de recursos alternativo, que controla os estados da CPU baseado em técnicas de otimização e que foi desenvolvido para melhor avaliar o ECO. O Capítulo 7 apresenta a avaliação dos mecanismos desenvolvidos. O Capítulo 8 apresenta uma segunda versão do ECO desenvolvida para amenizar o uso de buffers em detrimento do consumo de energia. Finalmente o Capítulo 9 conclui este trabalho e apresenta os trabalhos futuros.

Capítulo 2

Trabalhos Relacionados

Diversas questões permeiam o problema abordado neste trabalho.

A motivação inicial deste trabalho está no interesse da comunidade científica em desenvolver a Internet do Futuro. A internet atual foi desenvolvida a partir da ARPANET durante a década de 1970 e foi projetada seguindo o princípio de um núcleo de rede simples e transparente com a inteligência concentrada na borda da rede. Dessa forma, o núcleo da rede implementa somente a pilha de rede até a camada de roteamento, sendo as camadas de transporte até aplicação implementadas somente nos sistemas finais da rede. Essa escolha de projeto garantiu à Internet menor custo de implantação do núcleo e flexibilidade à rede para atender aos requisitos das aplicações da época e de aplicações futuras, possibilitando o crescimento e sucesso da Internet. Entretanto, novos requisitos foram surgindo ao longo dos anos e o núcleo da rede foi sofrendo "remendos" para atendê-los. A escassez de endereços IP foi remediada com a implantação do NAT (*Network Address Translation*) permitindo que diversos equipamentos acessassem a rede utilizando apenas um endereço IP global válido, mas que quebra o princípio de conectividade entre todas as estações da rede. O surgimento dos *Firewalls* também fere a conectividade da rede, mas tornou-se imprescindível dada a necessidade de proteger redes corporativas de acessos indesejáveis. O surgimento desses e outros "remendos" ossificou o núcleo da rede, tornando difícil a inovação no núcleo da rede e dificultando o atendimento a novos requisitos que vão surgindo [4], como a mobilidade das estações finais, o suporte a *multihoming* ou suporte a mecanismos de segurança no núcleo da rede para prevenir ataques de negação de serviço [27].

A necessidade de uma arquitetura de rede que suporte os requisitos das aplicações atuais e continue flexível para o atendimento aos novos requisitos que surgirem no futuro motiva o surgimento de diversos trabalhos para o desenvolvimento da arquitetura da Internet do Futuro. Pode-se distinguir a existência de duas linhas de pesquisa dentre as propostas de arquitetura para a Internet do Futuro [28]: a linha das abordagens puristas e a linha das abordagens pluralistas.

A primeira linha é a das abordagens puristas. Essa linha tem como ponto em comum a premissa de que deve haver um núcleo de rede monolítico e genérico desenvolvido para atender a todos os possíveis requisitos da rede. Dentre as propostas puristas encontra-se o RBA (*Role-Based Architecture* [29], que propõe uma arquitetura de rede que abandona o conceito de camadas de rede e propõe o uso de módulos denominados *roles* (papéis). Os *roles* são módulos com funções determinadas e específicas que podem ser interconectados entre si para prover o processamento de dados desejado. Esses módulos exercem todo o tipo de processamento que possa ser necessário dentro da rede, indo desde tarefas comuns na arquitetura atual, como encaminhamento de um pacote para o próximo salto, até atividades atribuídas hoje a *middleboxes*, como a filtragem de pacotes. Dessa forma, cada nó da rede pode possuir uma seqüência de módulos própria, processando os pacotes de forma diferenciada. Pacotes de aplicações que requeiram segurança podem passar por módulos de criptografia, enquanto pacotes de serviços de tempo real podem ser encaminhados com maior prioridade.

A segunda linha é a das abordagens pluralistas. As propostas pluralistas têm em comum a idéia de que o núcleo da rede deve estar preparado para suportar múltiplas pilhas de protocolos de rede em paralelo. Assim, a Internet seria formada por múltiplas redes concorrentes, cada uma com características próprias para atender aos requisitos específicos de um tipo de aplicação ou serviço. Dessa forma, dados de aplicações que demandem segurança, como o tráfego para acesso a um site bancário, utilizariam uma rede com suporte a diversos mecanismos de segurança, como autenticação de usuário ou criptografia, enquanto dados de uma aplicação de tempo real, como teleconferência ou jogos on-line, utilizariam outra rede com suporte a QoS, por exemplo. Dentre as propostas de arquiteturas pluralistas encontra-se CABO(*Concurrent Architectures are Better than One*) [8]. CABO propõe o uso de virtualização para separar o serviço prestado pelos ISPs em duas tarefas: o gerenciamento da infraestrutura de redes e a oferta de serviços. A infraestrutura de redes é virtualizada e os roteadores passam a suportar diversos roteadores virtuais, cada qual com sua pilha de protocolos de rede e gerenciamento independente dos outros, viabilizando a existência de diversas redes virtuais em paralelo. Esse modelo permite que um ISP possua uma rede virtual e passe a controlar todos os roteadores que fazem parte do caminho fim-a-fim de um determinado serviço, de forma a viabilizar a implantação de recursos como segurança ou QoS de forma efetiva.

A virtualização é um conceito base para o desenvolvimento de arquiteturas pluralistas, sendo explorado em diversos outros trabalhos. O conceito de virtualização de redes abrange a idéia de prover, em algum ponto das camadas de rede, uma camada de abstração que gera diversos ambientes isolados para acesso à infraestrutura de rede. O suporte à virtualização de redes está sendo adicionado dentro dos

equipamentos de rede, desde roteadores e comutadores dedicados convencionais até novos equipamentos de rede sendo propostos. Para ilustrar, a série E de roteadores da Juniper, por exemplo, virtualiza a rede a nível de roteador, possuindo suporte para até mil tabelas de encaminhamento em paralelo [30]. Com isso, essa série de roteadores permite que classes de tráfego diferentes sejam tratadas de maneira individual e isolada, de forma a atender, por exemplo, dois ISPs diferentes, mantendo a qualidade de serviço demandados por cada um. Embora esses roteadores tenham suporte limitado a tecnologias existentes hoje, já representam um passo em direção ao modelo proposto por CABO.

As primeiras iniciativas de virtualização de redes surgiram a partir do uso mais tradicional das ferramentas de virtualização, utilizando computadores baseados na plataforma x86 e ferramentas de virtualização inicialmente desenvolvidas para a consolidação de servidores, e não utilizando hardware dedicado. Essa escolha, embora possibilite maior flexibilidade devido à facilidade de desenvolvimento na plataforma x86, impôs inicialmente alguns entraves. Um dos aspectos mais interessantes da virtualização de redes é a possibilidade de migrar roteadores virtuais de um roteador para outro, possibilitando, por exemplo, melhor balanceamento de carga e facilidade na manutenção de roteadores. Entretanto, as ferramentas de virtualização para plataforma x86 foram desenvolvidas com o uso datancenters em mente para a consolidação de servidores, de forma que os servidores de origem e destino da migração da máquina virtual encontram-se na mesma rede local, e essas ferramentas não provinham suporte para migração entre máquinas que se encontrem em subredes distintas. Dessa forma, desenvolveram-se técnicas para expandir a migração para destinos fora da rede local, como visto em [31]. Uma dessas técnicas é a divisão do roteador virtual em dois planos: o plano de controle, responsável pela execução do protocolo de roteamento, e o plano de dados, que encaminha os pacotes de acordo com uma tabela configurada pelo plano de controle. Em [16] desenvolve-se um protótipo de roteador de software com suporte a múltiplas redes virtuais utilizando a ferramenta de virtualização de computadores Xen [32] e o *framework* Click [33] para a construção modular de roteadores de software. A utilização do Xen permite a criação de diversos roteadores virtuais que executam os protocolos de roteamento, enquanto que no domínio de gerenciamento do Xen que possui acesso direto ao hardware de rede, o Domínio 0, utiliza-se o Click para a geração de um plano de encaminhamento de dados que encaminha o tráfego de cada rede virtual isoladamente baseado nas tabelas de roteamento configuradas pelos roteadores virtuais.

Uma outra abordagem para a virtualização de redes é o Openflow [34], uma arquitetura para virtualização de redes a nível de comutador. O Openflow separa o tráfego de rede através de uma tabela para classificação de fluxos que combina diversos critérios para definir o destino dos dados trafegados, como endereços de

origem e destino da camada de enlace, portas e endereços de origem e destino da camada de transporte ou campos dentro dos cabeçalhos que indicam a prioridade dos dados. Além do comutador programável que encaminha dados rapidamente através da tabela de classificação de fluxos, a arquitetura do Openflow conta com um nó controlador. O nó controlador é responsável pela configuração das entradas nas tabelas de classificação de fluxos dos comutadores Openflow, além de processar os dados encaminhados pelos comutadores Openflow quando esses não correspondem a nenhum critério definido na tabela do comutador. O Openflow está disponível em diversas versões, havendo imagens de Openflow para roteadores wireless convencionais, pacotes para instalação em computadores baseados em x86 e até alguns modelos de comutadores de hardware dedicado com suporte a Openflow [35].

Para melhorar a flexibilidade dos roteadores, independente da arquitetura de Internet a ser adotada, surge uma nova classe de roteadores, os chamados roteadores de software. Esse novo tipo de roteador não mais é baseado em hardware dedicado e firmware proprietário, mas sim em hardware convencional baseado na plataforma x86, comum em computadores pessoais e servidores. Essa classe recente de roteadores, executa sistemas operacionais convencionais, geralmente Linux, e provê flexibilidade no encaminhamento de pacotes, podendo incorporar além da função de roteador, outras funções de *middleboxes* como *firewalls* ou *proxys*. Além da flexibilidade, os roteadores de software apresentam as vantagens de baixo custo, visto que utilizam hardware convencional amplamente disponível, e de serem fáceis de programar, visto que há uma ampla gama de ferramentas de desenvolvimento para a plataforma x86 e de pessoas capacitadas para o desenvolvimento de programas para essa plataforma.

Embora o uso de hardware convencional adicione flexibilidade aos roteadores a baixo custo, também possui suas desvantagens. Os roteadores de software possuem problemas de desempenho em relação aos roteadores baseados em hardware dedicado [19]. Para superar esta limitação, surgem diversas propostas. Em [36] combina-se placas de rede programáveis do tipo NetFPGA [37] aos computadores convencionais x86 para absorver as tarefas básicas de compartilhamento de acesso à interface de rede de forma a garantir o isolamento entre as redes virtuais e permitir a utilização do processamento da CPU para as tarefas de encaminhamento de pacotes acessando as tabelas de roteamento dos roteadores virtuais presentes. Em [38] é utilizada uma abordagem que utiliza o alto poder de processamento em paralelo fornecido pelas GPUs para o processamento dos pacotes a serem encaminhados.

As propostas de combinar algum tipo de hardware dedicado de alto desempenho aos computadores baseados em x86 aumentem as taxas de encaminhamento alcançadas, mas ainda assim é necessário o uso de outra estratégia para atingir taxas de encaminhamento de pacotes compatíveis com as demandadas por tráfego

de *backbone*. Dessa maneira, surge a idéia de construir roteadores de software em cluster de servidores, o que permite aos roteadores de software lidar com altas taxas de encaminhamento de pacotes. Dobrescu *et al.* [19] apresentam Routebricks, uma proposta de arquitetura de roteadores de software em cluster. No trabalho, os autores analisam a viabilidade do uso do roteador de software em cluster proposto como roteador para o *backbone*. O trabalho verifica diversas formas de interconectar os servidores utilizando apenas as suas portas de rede. Utilizando conexão direta entre todos os servidores para cluster pequenos e interconexão com uma topologia em borboleta generalizada para clusters maiores, Routebriks consegue escalar para um alto número de portas externas e, por utilizar apenas os servidores convencionais sem a necessidade de equipamentos extras, possibilita a construção barata de roteadores de software em cluster.

Uma outra proposta para a construção de roteadores de software em cluster é o Flowstream [15]. Nesse trabalho, apresenta-se uma outra arquitetura para a construção de roteadores de software em cluster onde combina-se um comutador Openflow ao grupo de servidores convencionais x86 que compõe o cluster. Embora aumente o custo do roteador em cluster, a adição do comutador programável Openflow aumenta a flexibilidade da plataforma. Em Flowstream, os pacotes chegam e saem do comutador Openflow, podendo ser redirecionados para qualquer servidor do cluster para processamentos específicos exercidos por *middleboxes* ou para processamentos genéricos como roteamento.

Embora ainda não foram identificados roteadores de software em cluster utilizando virtualização de redes, pode-se inspirar em soluções para a distribuição da carga de processamento das redes virtuais dentro do cluster considerando um problema de alocação de recursos análogo: o problema de balanceamento de carga em datacenters virtualizados. Em [39], Wood *et al.* propõe um mecanismo de balanceamento de carga para datacenters denominado Sandpiper. O Sandpiper consiste em um mecanismo que detecta servidores sobrecarregados baseados em uma métrica definida pelos autores como "volume" do servidor. O volume do servidor é composto pelo inverso do produto da quantidade disponível de memória, processador e banda, de forma que quando algum desses recursos estiver escasso, a métrica de volume adquire altos valores. Uma vez detectado um servidor sobrecarregado, é efetuada a escolha de que máquina virtual migrar. Essa é feita baseada no volume da máquina virtual dividido pela quantidade de memória que ela utiliza. A escolha é justificada pelos autores para obter maior eficiência na migração, visto que a principal etapa da migração é a cópia de memória da máquina virtual do servidor de origem para o servidor de destino e a relação (volume)/(memória utilizada) maximiza a quantidade de volume retirado do servidor sobrecarregado por esforço efetuado em migrar a máquina virtual. O servidor de destino escolhido para a migração é o

servidor com menor volume dentre os disponíveis no datacenter. Embora esta idéia tenha se mostrado eficiente para garantir o SLA das máquinas virtuais no datacenter, mostra-se uma má opção do ponto de vista energético. A técnica utilizada pelo *Sandpiper* tende a igualar a carga entre os servidores, visto que o servidor de destino das migrações é o que apresenta menor volume de carga no momento, e embora isso seja desejável do ponto de vista de desempenho, é indesejável do ponto de vista energético, já que a maior eficiência energética é atingida quando o servidor é utilizado ao máximo e a distribuição igualitária de carga faz os servidores operarem em uma zona de baixa eficiência energética [40].

O problema de alocação de recursos em datacenters também possui trabalhos que consideram a economia de energia como ponto central. Em [41], o problema de distribuição de cargas entre os servidores de um cluster de forma é resolvido explorando o recurso do desligamento de servidores para a obtenção de economia de energia. O mecanismo proposto pelos autores é baseado em uma formulação de problema de otimização que utiliza o processo de decisão de Markov com restrições para explora o desligamento de servidores até que haja tarefas em quantidade suficiente para compensar a sobrecarga gerada por ligar e desligar um servidor. Embora eficiente, essa técnica causa o adiamento do processamento da carga, que embora não seja um problema no contexto de fazendas de servidores, para o qual o mecanismo foi proposto, inviabiliza sua exploração em ambientes de rede. O uso desse mecanismo em ambientes de rede geraria grande quantidade de pacotes descartados pelo enchimento dos buffers e aumentos impraticáveis na latência dos pacotes armazenados em buffer, gerando diversos transtornos para as aplicações utilizando a rede.

A questão energética em ambientes virtualizados é crucial. A demanda por mecanismos de gerenciamento de energia tanto pela questão de sustentabilidade quanto pela questão do gasto monetário com energia fez com que as principais ferramentas de virtualização incluíssem mecanismos de economia de energia. A solução de gerenciamento de ambientes virtualizados *VMware vSphere* possui um mecanismo denominado *Distributed Power Management* (DPM) para economizar energia em datacenters. O DPM atua migrando máquinas virtuais e desligando servidores de acordo com políticas definidas pelo usuário mantendo o SLA dos serviços do datacenter gerenciado. O Xen Server 5.6 conta com um recurso de gerenciamento de energia denominado *Automated Power Management* enquanto o HyperV R2 da Microsoft conta com um mecanismo de economia de energia baseado no gerenciamento da energia dos processadores [42].

A questão de economia de energia em datacenters ainda conta com outras soluções mais ortodoxas. Em [43], os autores propõem a economia de energia através da eliminação do sistema de ar condicionado. Para isso, os autores propõem que

os servidores sejam postos em ambientes externos em locais de baixa temperatura, utilizando apenas uma tenda para cobri-los. A proposta foi avaliada utilizando equipamentos de hardware convencional e uma tenda desenvolvida pelos autores de forma a proteger o equipamento das intempéries ao mesmo tempo em que garante a circulação de ar e temperatura necessárias para o funcionamento dos equipamentos.

Soluções para economia de energia em ambientes de rede também tornam-se cada vez mais comuns. Nedevschi *et al.* [22] exploram a idéia de armazenar pacotes nos buffers das interfaces de rede para permitir o uso de estados de mais baixo consumo energético e de adaptar a taxa de transmissão das interfaces de rede para economizar energia. Para modelar o gasto energético foi utilizado um modelo energético simplificado, onde o dispositivo de rede encontra-se sempre ou no estado de repouso ou no estado de adaptação de taxa de transmissão. Dessa forma, definiu-se o gasto energético do dispositivo como sendo

$$E = P_s.T_s + P_r.T_r \tag{2.1}$$

onde E é o consumo energético, P_s o gasto energético em repouso, T_s o tempo em que o dispositivo fica em repouso, P_r o gasto energético em adaptação de taxa de transmissão e T_r o tempo em que o dispositivo fica em adaptação de taxa de transmissão. Para modelar o custo de transições entre os estados, considerou-se um custo expresso por um tempo em que o dispositivo não efetua nenhum tipo de tarefa útil. Também considerou-se, para simplificar o modelo, um único estado de adaptação de taxa de transmissão. Utilizando o artifício de repouso, o trabalho propõe um método chamando *buffer&burst*, onde pacotes são acumulados por B ms enquanto o roteador está em repouso e depois desses B ms desperta-se o roteador e envia-se todos os pacotes pendentes. Ao finalizar o encaminhamento dos pacotes da rajada, o roteador volta a entrar em repouso por B ms. Utilizando a adaptação de taxa, o trabalho propõe outro modelo, onde o dispositivo pode transmitir em N taxas diferentes. O custo da mudança de taxa é modelado novamente por um intervalo de tempo onde o dispositivo não opera. O controle de taxa de transmissão então é efetuado de forma que sejam evitadas mudanças na taxa de transmissão, pois essas são custosas e podem causar perdas e aumentos na latência. O controle é efetuado utilizando dois limiares. Quando o número de pacotes no buffer de entrada dividido pela taxa de transmissão for maior que o atraso aceitável, ou, quando o tempo para um aumento da taxa de transmissão tornar-se grande o suficiente para acarretar em perdas, o mecanismo aumenta a taxa de transmissão. Quando o buffer estiver vazio e a taxa de chegada de pacotes estimada seja menor do que a taxa utilizada, diminui-se a taxa de transmissão. A solução proposta é interessante é pode ser combinada a outros mecanismos de economia de energia utilizados na política de

alocação de recursos de processamento de pacotes do roteador.

Em [44], propõe-se economizar energia em enlaces do *backbone* de alta capacidade, compostos pelo agrupamento de cabos de enlaces de menor capacidade. A economia é atingida desligando-se uma parte dos enlaces de menor capacidade de acordo com a utilização. A decisão de quais cabos desligar é feita a partir de um algoritmo baseado em otimização que minimiza o número de cabos necessários em cada enlace para transportar os dados demandados pela rede. Para evitar o aumento da latência no transporte dos dados, o problema de otimização é formulado incluindo um custo para a redução da capacidade de um grupo de cabos abaixo da capacidade estimada como necessária. O custo embutido é proporcional ao aumento no caminho que os dados que seriam trafegados pelo enlace sofrem por passarem por outros enlaces da rede. Para exemplificar, caso um enlace de 10 Gb formado por dez cabos de 1 Gb tenha a demanda estimada em 6,3 Gb, ao optar por adequar a capacidade do enlace para 6Gb desligando-se quatro cabos, a função custo contabiliza um custo associado ao redirecionamento dos 0,3 Gb excedentes por caminhos alternativos mais longos. A avaliação efetuada mostra grande potencial de economia de energia, mas peca ao não demonstrar os efeitos negativos na rede, como aumento de latência, causados pela aplicação do mecanismo proposto.

Dentro da questão de economia de energia em ambientes de rede, Bolla *et al.* [23] analisam o consumo energético e desempenho de encaminhamento de diversos roteadores de software. No estudo são utilizados computadores baseados em diversas plataformas de processadores Intel e AMD com diversas velocidades de barramento. O artigo faz a análise da vazão atingida, consumo energético e latência obtida ao alterar o número de núcleos utilizados nos processadores e a freqüência de operação dos núcleos. O artigo avalia o modelo de encaminhamento do Linux baseado na implementação NAPI (*New Aplication Programming Interface* verificando o impacto no consumo energético e desempenho ao mudar a freqüência de operação dos núcleos da CPU, o número de núcleos da CPU e a quantidade de pacotes enviados. A implementação do NAPI trata o encaminhamento de pacotes via interrupção até um certo limiar e, após este limiar, passa a encaminhar pacotes fazendo *pooling* periodicamente. Ao fim dessas avaliações é proposto um modelo de gasto energético para os roteadores de software dependendo da freqüência de operação.

Baseado neste trabalho, Bolla *et al.* [45] exploram a troca de estados da CPU para economizar energia em roteadores de software baseados em hardware x86. É definida uma função custo para a escolha de qual estado de CPU utilizar no roteador de software para um intervalo de tempo de 10 minutos baseado na média, variância e demanda de picos medidos em intervalos anteriores. A função custo é avaliada utilizando um algoritmo de força bruta. Embora o artigo apresente resultados considerados satisfatórios na economia de energia, a baixa freqüência com que o estado

da CPU é governado torna a abordagem pouco agressiva na economia de energia. Infelizmente, a escolha por otimização usando força bruta demanda grande quantidade de tempo para sua execução e inviabiliza o controle da CPU com freqüência de execução na ordem de grandeza de frações de segundo.

Em [46], estuda-se o problema da economia de energia em redes de datacenters. As redes de datacenters são projetadas para atender a demanda de pico, geralmente muito superior à demanda de tráfego encontrada, e possuem redundâncias, evitando ponto único de falha. Essas escolhas de projeto fazem com que essas redes sejam compostas por um grande número de dispositivos de rede, em geral operando sob baixa carga, gerando grande desperdício de energia. Dessa forma, o autor propõe um roteamento ciente do gasto energético. A idéia do mecanismo proposto é, dada a redundância da rede, desligar dispositivos de rede até diminuir a vazão total da rede abaixo de um determinado limiar. O mecanismo funciona de forma iterativa. A cada rodada, o mecanismo calcula o roteamento entre os servidores dada a topologia da rede do datacenter e a matriz de tráfego entre os servidores. Após calcular o roteamento, o mecanismo elimina da topologia os dispositivos de rede que carregam menos tráfego, tomando o cuidado de não retirar dispositivos de rede considerados críticos, como aqueles que particionariam a topologia caso fossem eliminados. Após decidir quais dispositivos desligar, o mecanismo calcula a vazão total da nova topologia. Caso a vazão da nova topologia em relação à vazão inicial caia abaixo do limiar definido, o processo de eliminação está concluído, caso contrário, inicia-se uma nova iteração.

Em [47], propõe-se uma arquitetura de cluster para servir aplicações web com consumo energético proporcional ao número de requisições. Para isso, os autores propõem o uso de um cluster formado por plataformas heterogêneas de hardware e o uso de uma política específica de balanceamento de carga que suspende servidores desnecessários para atender a demanda. O cluster proposto é formado por servidores convencionais, dispositivos baseados em processadores ATOM x86 para dispositivos móveis de baixo consumo e em dispositivos embarcados baseados em processadores ARM. Os servidores possuem alta capacidade de processamento, alto consumo energético, altos tempos de transição entre os estados de suspensão e ativo e apresentam melhor eficiência energética que outras plataformas quando utilizados sob carga máxima. Os dispositivos baseados em ATOM possuem baixo consumo e baixos tempos de transição entre os estados ativo e suspenso. Os dispositivos baseados em ARM são de baixíssimo consumo, possuem a menor capacidade de processamento e praticamente não variam o gasto energético entre seu uso com carga máxima ou ocioso. A proposta é prever periodicamente a carga para o próximo intervalo de tempo e adequar a capacidade de processamento do cluster à demanda estimada. Para atender a demanda estimada, busca-se utilizar preferencialmente os

servidores, por apresentar melhor eficiência energética, e utilizam-se os outros tipos de dispositivos para atender picos na demanda, visto que os servidores possuem alto tempo de transição para o estado operacional. O esquema proposto é avaliado servindo páginas web de uma cópia da Wikipedia e, nos testes efetuados, apresenta a característica de proporcionalidade do consumo energético à carga.

O problema de alocação da banda disponível nos enlaces externos do roteador entre as redes virtuais é uma questão central na construção de plataformas pluralistas para a Internet do Futuro. Dentro deste contexto, DaVinci [6] propõe um mecanismo de alocação dinâmica de banda entre as redes virtuais baseado em otimização linear. O mecanismo proposto suporta a divisão do tráfego entre diversos caminhos e consegue operar utilizando informações obtidas dos enlaces locais do roteador. Como ponto negativo, DaVinci não explora a possibilidade de migração de redes virtuais que é uma das possibilidades mais interessantes das arquiteturas pluralistas para a redistribuição dos recursos da rede. Outro ponto negativo de DaVinci é ignorar a questão energética dentro do mecanismo de alocação de banda.

Finalmente, uma questão importante para o desenvolvimento deste trabalho é a questão da avaliação da proposta. Dada a impossibilidade de testar o mecanismo proposto em um sistema real, opta-se por simular um roteador de software em cluster. Para isso, é de suma importância a carga oferecida pelas redes virtuais e para gerá-las opta-se por separar classes de tráfego a partir de dados reais de tráfego. Existem diversos métodos de separação de tráfego e nenhum obtém taxas de erro desprezíveis, sendo ainda uma área de pesquisa que apresenta muitos desafios. Um primeiro método para a classificação consiste na análise da porta de destino de cada pacote analisado. Esse método apresenta boa acurácia para aplicações conhecidas com portas bem definidas, como web ou ssh, mas falha bastante quando há mau uso das portas por parte das aplicações ou quando a aplicação não possui portas muito bem definidas, como é o caso de aplicações de troca de arquivo par-a-par. Uma outra técnica é a classificação baseada na inspeção dos dados do pacote, onde a aplicação ou serviço são determinados por assinaturas características nos dados. Essa técnica apresenta alta acurácia para aplicações conhecidas, mas é ineficaz ao analisar pacotes de aplicações novas. Outra técnica possível é a análise baseada no comportamento da estação final. Nessa técnica analisa-se o padrão de portas e endereços de destino dos pacotes de uma estação e compara-se o padrão com as assinaturas disponíveis na base de dados. Esta técnica possui a vantagem de poder ser utilizada mesmo com tráfego criptografado, deficiência da técnica baseada na inspeção dos dados dos pacotes, mas sofre o mesmo problema de não identificar tráfegos de aplicações/serviços novos ainda não disponíveis na base de dados. Um outro tipo de técnica é baseado na análise de fluxos no tráfego. Existem duas variantes deste tipo de técnica, uma baseada em técnicas de aprendizagem de máquina supervisionada e outra em

aprendizagem de máquina não supervisionada. A utilização de técnicas de aprendizado supervisionado requer prévio treinamento do mecanismo de inteligência com vasto conjunto de fluxos, possui a vantagem de classificar corretamente fluxos com diferenças sutis e permite a classificação do fluxo imediatamente após seu término, mas geralmente possui dificuldade em classificar fluxos de aplicações novas. Já um mecanismo de aprendizado não supervisionado basicamente opera agrupando fluxos com características similares. Dessa forma não requer treinamento e é mais generalista, conseguindo classificar fluxos de aplicações novas. Entretanto, apresenta as desvantagens de falta de precisão na classificação e a necessidade do processamento *off-line* visto que esse opera comparando os fluxos e precisa de um grande número para efetuar uma boa classificação.

Os trabalhos acima mostrados, embora de diferentes áreas e abordando diferentes problemas, atacam aspectos importantes para a construção de uma plataforma de Internet do Futuro com a preocupação de minimizar o consumo energético por motivos ambientais e econômicos. O uso de roteadores de software em cluster provê uma plataforma flexível e com capacidade de processamento escalável, que, quando combinada com a virtualização de redes possibilita a existência de múltiplas redes virtuais independentes, cada uma com características próprias. Essa combinação também provê recursos para o gerenciamento de energia, visto que os servidores que compõem o cluster podem ser desligados e religados de acordo com a demanda utilizando a migração de redes virtuais para redistribuir a carga. As idéias de balanceamento de carga e economia de energia em datacenters tradicionais podem ser utilizadas para inspirar mecanismos para o problema análogo no ambiente de redes e soluções para economia de energia nos enlaces podem ser usadas como inspiração para economizar energia dentro do roteador de software em cluster.

Capítulo 3

O Ambiente do Roteador de Software em Cluster

Este capítulo apresenta o ambiente do Roteador de Software em Cluster. Inicialmente, analisam-se os fatores de consumo energético de um roteador de software em cluster e apresentam-se maneiras de controlar o desempenho e o consumo energético do mesmo. Em seguida, apresentam-se dois modelos de roteadores de software em cluster e justifica-se a escolha de um para o escopo deste trabalho.

3.1 Fatores de Consumo Energético

Para a construção de um roteador de software baseado em cluster que seja energeticamente eficiente, o primeiro passo é identificar quais são os componentes dos hardwares convencionais que demandam mais energia e avaliar as possíveis ações para diminuir seu consumo energético.

Os componentes mais utilizados por roteador de software baseado em hardware convencional, nas tarefas associadas ao encaminhamento de pacotes, são a CPU, o adaptador de rede, e a memória RAM. A CPU é, de longe, o componente que mais consome energia e foi identificado como gargalo em roteadores de software baseados em cluster [19]. Para ilustrar, um servidor convencional da Dell equipado com 2 CPUs Xeon consome 420W sob carga máxima[1] e 220 W quando ocioso [48]. As CPUs Xeon consomem aproximadamente 130 W sob carga máxima [49] e respondem pela maior parte da variação do gasto energético do servidor entre os estados de ocioso e sob carga máxima, significando que a CPU é um componente chave para a economia energética. O adaptador de rede possui um gasto energético relativamente baixo, aproximadamente 10 W [50], e por isso pode ser inicialmente ignorado na con-

[1] Gasto energético sob carga máxima medido utilizando o *benchmark* de CPU SANDRA Dhrystone.
(http://www.sisoftware.net/?d=qa&f=ben_cpu&l=en&a=)

fecção de um mecanismo de economia de energia. O consumo energético da RAM não pode ser diretamente controlado, pois ainda não conta com recursos de gerenciamento de desempenho e consumo energético da mesma maneira que a CPU. O resto do gasto energético do servidor remete à placa-mãe, fonte, sistema de resfriamento, e periféricos. Visto que os consumos desses componentes não podem ser controlados em alta granularidade, isto é, através da seleção de diversos estados de operação que ofereçam diferentes níveis de consumo energético e desempenho, pode-se gerenciar o gasto dos mesmos como um bloco. O gasto energético desse bloco pode ser controlado, em baixa granularidade, colocando o servidor em estado de *standby* para economizar energia, visto que nesse estado o servidor consome aproximadamente 15 W, e, quando houver a necessidade de utilizar a capacidade de processamento do servidor, religá-lo através da rede local utilizando o recurso de *wake on LAN*) [51].

Os servidores atuais possuem uma propriedade de não-proporcionalidade do gasto energético em relação a sua carga. Isto é, um servidor atual que está utilizando x% de sua capacidade não consome x% de energia em relação ao seu consumo sob carga máxima. O fato do gasto energético de um servidor não ser proporcional é considerado um sério entrave para a eficiência energética dos clusters. Embora os processadores estejam cada vez mais com consumo energético diretamente proporcional ao seu uso, os demais componentes do sistema ainda não acompanham esta tendência, sendo a memória RAM e o disco rígido os componentes mais cruciais para atingir esta meta [40]. Visto que um servidor ocioso consome em torno de 50% de seu gasto energético máximo [52], considerando um ponto de vista preocupado prioritariamente com o gasto energético, é melhor ter um servidor operando em 90% de sua capacidade e outro em *standby* do que dois servidores operando em 45% da capacidade.

Combinando os possíveis estados de freqüências/voltagens de operação dos núcleos de duas CPUs Xeon, Bolla *et al.* [45], estudaram o gasto energético da CPU em cada possível estado e sua capacidade de encaminhamento de pacotes. O trabalho considera 35 estados diferentes para a CPU quando combinados os estados de todos seus núcleos, número de estados suficientes para prover um bom controle sobre a capacidade de processamento do servidor e seu gasto energético. As medidas feitas no trabalho foram o gasto energético da CPU em cada estado, tanto quando ativa quanto quando ociosa. Além do consumo em cada estado da CPU, o trabalho também inclui a taxa máxima de encaminhamento de pacotes obtida no estado em questão.

Observando as curvas de gasto energético da CPU quando ativa e de capacidade de encaminhamento de pacotes em cada estado, obtidas combinando os dados sobre a CPU de [45] e o consumo base de um servidor retirado de [48], mostradas nas Figuras 3.1 e 3.2 respectivamente, pode-se observar uma relação quase linear na

capacidade de encaminhamento de pacotes e no gasto energético da CPU ativa ao variar o estado da CPU. Dado que a troca de estado da CPU dura poucos μs, o impacto de trocar de estado na CPU pode ser negligenciado quando utilizando esse artifício em mecanismos de controle que operem na escala de tempo de ms. Uma CPU Xeon da série 5500, por exemplo, demora no máximo 2 μs para efetuar a troca de estado [49]. Considerando um enlace de 10 Gb/s, no máximo 2685 B de dados deixariam de ser processados durante essa troca de estados, quantidade de dados facilmente absorvida pelos buffers sem grandes impactos.

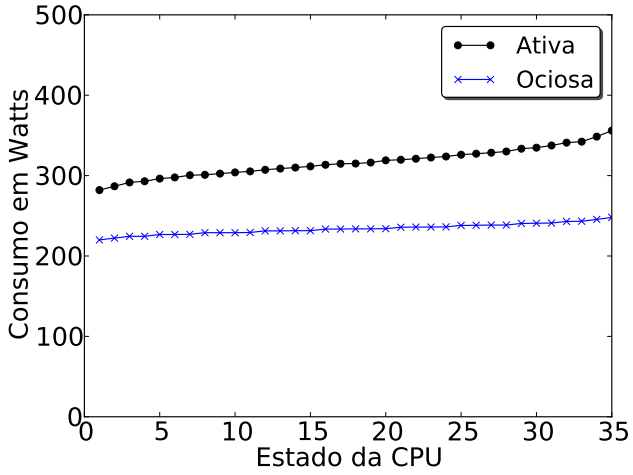


Figura 3.1: Consumo energético do servidor para cada estado da CPU quando ativa ou ociosa.

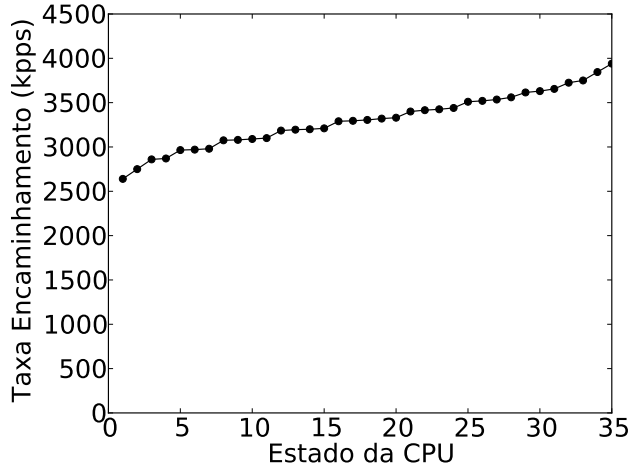


Figura 3.2: Taxa máxima de encaminhamento da CPU em cada estado.

A virtualização provê outras possibilidades para economizar energia em um roteador de software baseado em cluster, como a migração de máquinas virtuais e o controle do tamanho dos buffers de recepção.

A migração de roteadores virtuais possibilita balancear a carga e também pode ser usada para controlar os gastos energéticos [53]. A migração de roteadores virtuais permite consolidar a carga em um menor número de servidores dentro do cluster em períodos de baixa demanda, tornando possível o desligamento dos servidores ociosos para maximizar a economia de energia. Por outro lado, durante períodos de pico, é possível obter melhor distribuição da carga entre os servidores do cluster.

O controle do tamanho de buffers pode ser utilizado para melhorar o desempenho ou economia de energia. Aumentar os buffers permite o uso de um estado de CPU com capacidade inferior à demanda de pico e menor gasto energético, visto que o buffer absorve os picos da demanda. Por outro lado, diminuir o tamanho dos buffers demanda estados de CPU com maior desempenho para atender os picos de demanda, aumentando o consumo energético mas diminuindo a latência. O tamanho do buffer pode ser facilmente alterado dentro da ferramenta de virtualização e demora frações de segundo para ser configurado. Na implementação dos buffers de recepção de pacotes no Linux, há filas individuais de recepção para cada interface de rede, seguidas dentro do núcleo do sistema operacional por filas conhecidas como *backlog queues* que armazenam os pacotes antes do seu processamento, uma para cada CPU. Para efeito de simplicidade na modelagem dos buffers envolvidos na recepção de pacotes, neste trabalho será considerado que todos os pacotes recebidos vão para um único buffer antes de seu processamento. No modelo considerado neste trabalho, para ser conservador, considera-se que esse buffer unificado possui capacidade de armazenamento equivalente à de uma única fila de recepção de interface de rede.

Considerando os fatores de consumo energético e as ações identificadas para seu controle, verifica-se que as ações de controle de energia em roteadores de software em cluster possuem escalas de tempo para serem utilizadas com ordens de grandeza diferentes. Enquanto a configuração do estado em uso da CPU pode ser efetuado diversas vezes em um segundo, ações como a migração de um roteador virtual ou o desligamento de um servidor ocorrem na escala de tempo de segundos, ou até minutos. Dessa forma é necessário que o mecanismo de gerenciamento de recursos saiba diferenciar as escalas de tempo corretamente e harmonizar as ações de controle para que uma não atrapalhe a outra e obtenha-se o resultado desejado.

3.2 Modelo Básico de Roteador de Software em Cluster

O presente trabalho explora a arquitetura dos roteadores de software em cluster como uma forma de atingir dois objetivos: atender as demandas de encaminha-

mento das redes virtuais associadas ao roteador e economizar energia. Nessa seção apresentam-se dois modelos diferentes de roteadores de software em cluster identificados na literatura. O primeiro modelo é exemplificado na Figura 3.3 e representa a idéia básica do Routebricks [19]. Na figura, observa-se um roteador de software em cluster formado por seis servidores convencionais, representados pelas caixas na borda da nuvem, cada um responsável por um dos enlaces externos, representados pelas setas na face de cada servidor voltada para a parte externa do cluster. As linhas sem ponta no interior do cluster representam as interconexões entre os servidores que formam o cluster.

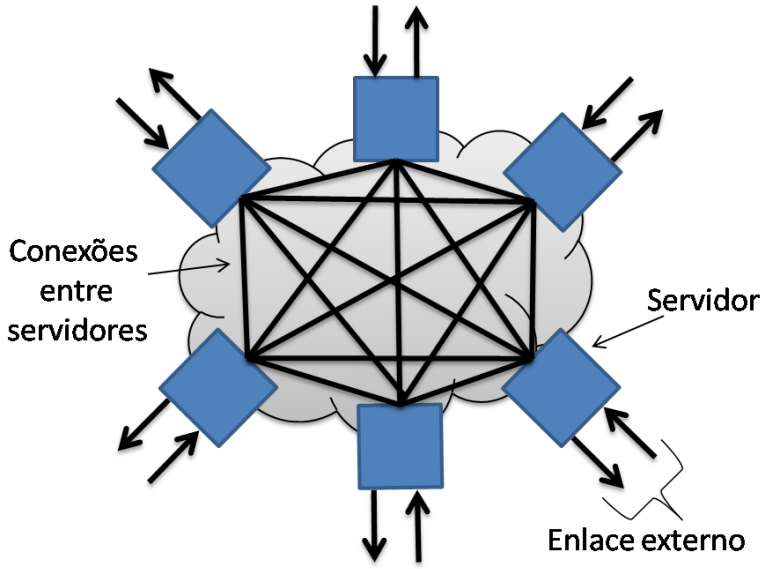


Figura 3.3: Modelo proposto por Routebricks.

A arquitetura apresentada na Figura 3.3 possibilita aumentar a capacidade de encaminhamento de pacotes dos roteadores de software, utilizando um cluster de servidores para desempenhar o papel de roteador de software, possibilitando escalar a capacidade de encaminhamento aumentando o número de servidores que compõe o cluster. Por utilizar as próprias interfaces de rede dos servidores para conexão direta com os outros servidores que compõem o cluster, esta proposta elimina a necessidade de utilizar equipamentos como comutadores para a interconexão dos servidores, tornando essa solução para construir roteadores de software uma solução mais barata. Por outro lado, essa solução para construção de roteadores de software em cluster prejudica a economia de energia. O desligamento de servidores para a economia de energia em situações de baixa demanda é um importante recurso para economizar energia, conforme mencionado na seção anterior, entretanto, não é possível desligar um servidor para economizar energia nessa solução, visto que isto significaria desconectar a porta externa do roteador pela qual o servidor desligado

é responsável.

Por esta razão, outra arquitetura é considerada neste trabalho. A arquitetura considerada possibilita o desligamento de servidores sem diminuir a disponibilidade das portas externas. A arquitetura considerada é ilustrada na Figura 3.4 que representa a idéia básica do Flowstream [15].

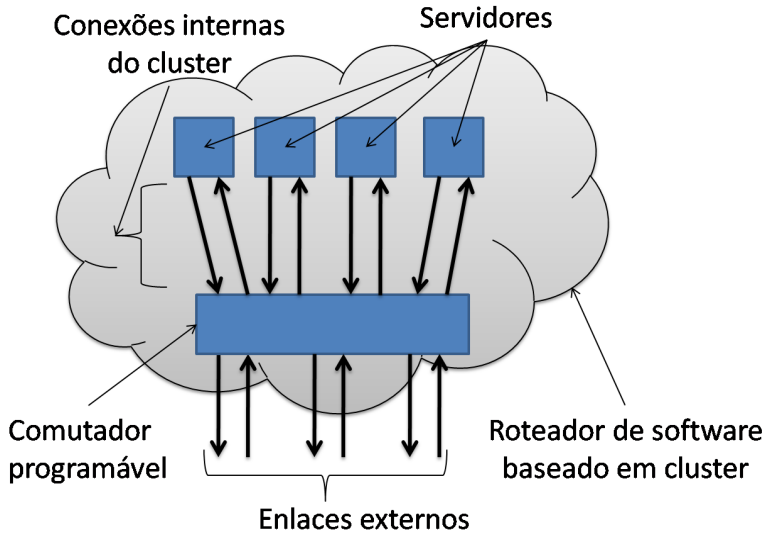


Figura 3.4: Modelo proposto por Flowstream.

Nessa arquitetura, as portas externas do roteador estão conectadas a um comutador programável do tipo Openflow, bem como todos os servidores que formam o cluster. O comutador programável é configurado para encaminhar os pacotes de cada rede virtual para o servidor onde se encontra o roteador virtual associado àquela rede virtual e, após o processamento do pacote, repassá-lo ao enlace externo apropriado. Essa camada de indireção inserida pelo comutador programável desacopla os servidores do cluster dos enlaces externos, possibilitando o desligamento de servidores do cluster. O redirecionamento do tráfego de uma rede virtual para um servidor específico não é uma tarefa difícil para um comutador Openflow pois, como mencionado no Capítulo 2, o comutador Openflow possui a capacidade de tratar individualmente cada fluxo que trafega pela rede, condição muito mais específica que a classificação de dados que trafegam por uma rede virtual. Para efetuar o desligamento de um servidor, basta migrar os roteadores virtuais que estejam utilizando seus recursos para outros servidores do cluster e, ao concluir as migrações, reconfigurar o comutador para redirecionar o tráfego das redes virtuais migradas para os respectivos servidores de destino das migrações sobrescrevendo as antigas entradas associadas ao tráfego dessas redes virtuais na tabela do comutador. Caso a carga de processamento das redes virtuais aumente, basta religar servidores desligados, migrar algumas redes virtuais para os mesmos e reconfigurar o comutador para redirecionar o tráfego apropriadamente sobrescrevendo as entradas da tabela

do comutador associadas às redes virtuais migradas.

Após verificar com mais detalhes as arquiteturas de roteadores de software em cluster existentes e verificar a organização do modelo considerado, pode-se passar para a descrição do mecanismo proposto de alocação energeticamente eficiente de recursos. O próximo capítulo apresenta o ECO-ALOC, mecanismo proposto neste trabalho.

Capítulo 4

O Mecanismo ECO-ALOC

Neste capítulo é apresentado o mecanismo proposto, o ECO-ALOC. O ECO-ALOC é um mecanismo de alocação energeticamente eficiente de recursos para roteadores de software em cluster.

O problema de alocação de recursos em um roteador de software em cluster pode ser dividido em duas partes: a alocação da banda disponível nos enlaces externos entre as redes virtuais, e a distribuição dos recursos necessários para o processamento dos pacotes para cada roteador virtual. O compartilhamento da banda entre as redes virtuais pode ser tratado analogamente ao problema de compartilhamento de banda entre diferentes classes de serviço e existem diversas propostas para a solução deste problema [6, 54–58]. Assim, este trabalho foca o problema do compartilhamento entre as redes virtuais dos recursos do cluster associados ao processamento dos pacotes. Esse problema pode ser subdividido em dois problemas com escopos diferentes, onde as ações de controle possíveis ocorrem em escalas de tempo diferentes. Assim, o mecanismo proposto é na verdade composto de dois mecanismos: um de dinâmica lenta e outro de dinâmica rápida. No escopo do cluster como um todo, deve-se decidir quais redes virtuais são alocadas em cada servidor. No presente trabalho, o Mecanismo de Dinâmica Lenta (MDL) é responsável por orquestrar a alocação de redes virtuais dentre os servidores do cluster, e o faz buscando distribuir as redes virtuais em um subconjunto dos servidores do cluster, de forma a possibilitar o desligamento de servidores para economizar energia. O nome do Mecanismo de Dinâmica Lenta justifica-se pelo mesmo utilizar ações de controle que ocorrem em uma escala de tempo lenta, que vai de vários segundos, para o caso da migração de redes virtuais, até poucos minutos, para o caso do religamento de servidores. No escopo individual de cada servidor, deve-se escolher qual é o estado de voltagem/freqüência de operação da CPU adequado para atender à demanda de processamento imposta pelo tráfego da rede. No presente trabalho, o Mecanismo de Dinâmica Rápida (MDR) é responsável pelo controle do estado da CPU utilizado em cada momento. Além de preocupar-se com o processamento demandado pelas

redes virtuais, o MDR também busca evitar o desperdício de capacidade de processamento, utilizando estados da CPU de menor consumo energético para economizar energia. O nome do Mecanismo de Dinâmica Rápida, da mesma maneira que o Mecanismo de Dinâmica Lenta, faz referência à escala de tempo de suas ações de controle, poucos *mus* para a alteração do estado da CPU.

O objetivo principal do mecanismo proposto, ECO-ALOC, é diminuir o consumo energético do roteador de software em cluster sem prejudicar sua tarefa de encaminhamento de pacotes. O mecanismo possui parâmetros configuráveis que permitem regular a agressividade na economia de energia, permitindo alterar o ponto de operação para priorizar latência ou economia de energia. Outro objetivo do mecanismo proposto é facilitar a manutenção do cluster, possibilitando a retirada de um servidor do cluster sem causar distúrbios nos serviços das redes virtuais. O último objetivo é possibilitar a adição de novos servidores no cluster de forma transparente para as redes virtuais.

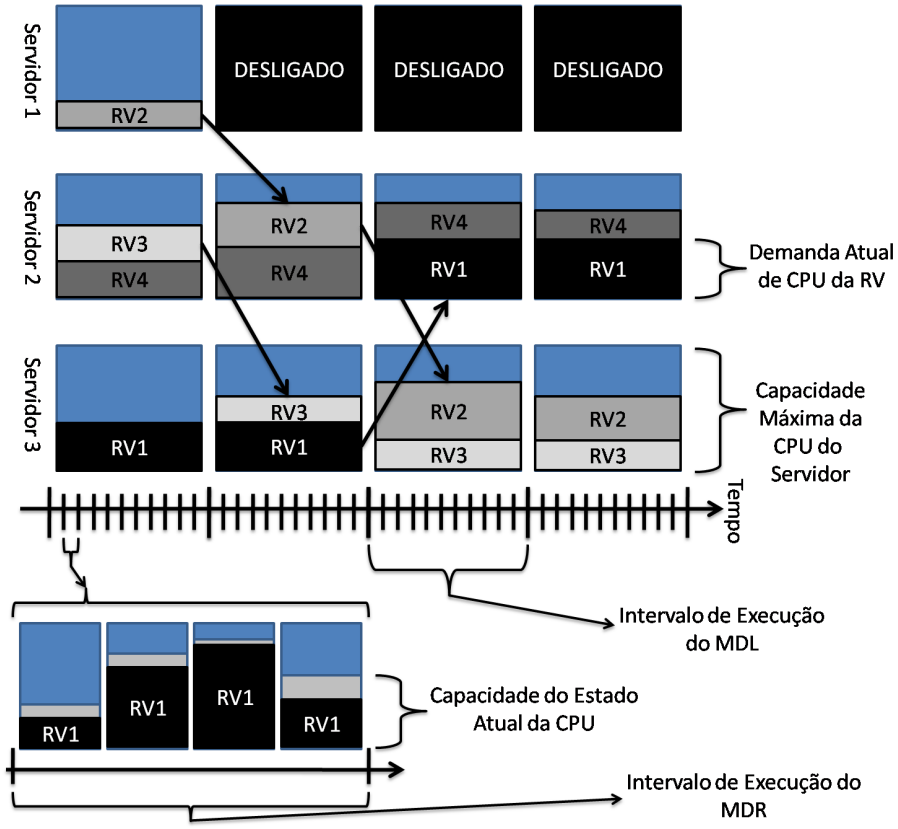


Figura 4.1: Exemplo dos escopos temporais do MDR e do MDL.

A Figura 4.1 exemplifica o escopo temporal e espacial da execução do MDL e do MDR, mostrando a diferença na escala de tempo da execução dos dois mecanismos e no escopo da ação dos dois, cluster como um todo para o MDL e servidores individualmente para o MDR.

Nesse exemplo, o MDL e o MDR estão controlando um cluster com 3 servidores

e 4 Redes Virtuais (RVs). O MDL é executado com menor periodicidade e atua no cluster como um todo, enquanto o MDR é freqüentemente executado dentro de cada servidor. O MDR planeja a seqüência de estados de operação da CPU a serem utilizados dentro de um pequeno intervalo de tempo de acordo com a estimativa de demanda de tráfego para o período, como mostrado na parte inferior da Figura 4.1. O MDL controla a alocação das redes virtuais entre os servidores do cluster e baseia-se nas demandas estimadas de tráfego das redes virtuais em uma escala de tempo maior. O controle efetuado pelo MDL desliga os servidores ociosos para economizar energia, conforme mostrado na parte superior da figura.

4.1 O Mecanismo de Dinâmica Rápida (MDR)

Esta seção descreve o Mecanismo de Dinâmica Rápida (MDR). O MDR orquestra os estados da CPU para economizar energia ao mesmo tempo em que atende a demanda de processamento das redes virtuais. De forma mais específica, o MDR determina quais estados da CPU devem ser utilizados no próximo intervalo de tempo, em que ordem e durante quanto tempo, definindo assim a seqüência de estados da CPU a ser utilizada dentro do intervalo de tempo para atender ao processamento demandado. O MDR não se preocupa com o consumo de memória RAM das redes virtuais, apenas com a demanda de CPU. A possível ação de controle para alocação de memória RAM é migração de roteadores virtuais para servidores com disponibilidade do recurso, dessa forma a preocupação com o consumo de memória recai sobre o MDL. Para efetuar o planejamento da seqüência de estados de CPU a serem utilizados durante um intervalo de forma a atender a demanda de processamento e economizar energia, propõe-se o uso do *Efficient/Cautious/Otiose (ECO)* como MDR.

O ECO foi inspirado na idéia de armazenar pacotes em buffer e enviar e na idéia de controlar o estado da CPU para economizar energia, propostos em [22] e [45] respectivamente e descritos com maiores detalhes no Capítulo 2. A premissa básica do ECO é de que pode-se atender a demanda de processamento das redes virtuais para um intervalo de tempo utilizando-se três estados da CPU, os estados *Efficient, Cautious* e *Otiose*. O estado *Efficient* é definido como o estado da CPU que apresenta o melhor desempenho energético no encaminhamento de pacotes. De maneira mais formal, é o estado da CPU que com o maior valor na relação de (capacidade de encaminhamento/gasto energético). O estado *Cautious* é o estado da CPU que possui a mais alta capacidade de encaminhamento de pacotes, isto é, o estado da CPU que permite ao servidor encaminhar pacotes na maior taxa possível. O estado *Otiose* é o estado da CPU que apresenta o mais baixo consumo energético quando a CPU está ociosa. No estado *Otiose* o servidor não encaminha

pacotes, gastando CPU somente para operações básicas de manutenção do Sistema Operacional.

A idéia chave do ECO é buscar maximizar a economia de energia utilizando como padrão o estado *Efficient* para encaminhamento de pacotes. Dessa forma, em situações em que a demanda de processamento de tráfego é baixa, o ECO planeja o uso do estado *Efficient* durante o tempo necessário para encaminhar a quantidade de pacotes estimada e planeja que a CPU passe o restante do tempo do intervalo no estado *Otiose*, de forma a economizar mais energia. Em situações de demanda mais acentuada, o ECO planeja o uso combinado dos estados *Efficient* e *Cautious* de forma a atender a demanda estimada para o intervalo. Em situações de demanda estimada superior à capacidade de encaminhamento do servidor, utiliza-se apenas o estado Cautious visto que é o estado que apresenta a maior taxa de encaminhamento de pacotes que o servidor suporta.

As heurísticas que orientam o funcionamento do MDR-ECO estão explicitadas no Algoritmo 1.

Algorithm 1 Heurísticas do MDR-ECO

Require: $T \in \mathbb{N}$, o tamanho do intervalo de tempo; e $D \in \mathbb{N}$, a demanda estimada de encaminhamento de pacotes dentro do intervalo de tempo;

Ensure: $t_e, t_c, t_o \in \mathbb{N}$, os intervalos de tempo gastos nos estados *Efficient, Cautious* e *Otiose* da CPU, respectivamente;

1: ———————————————————

2: Sejam $C_e, C_c \in \mathbb{N}$ as capacidades de encaminhamento de pacotes nos estados *Efficient* e *Cautious* da CPU, respectivamente;

3: $t_e, t_o, t_c \leftarrow 0$; // Inicialização

4: **if** $(D < C_e.T)$ **then**

5: // c_{eff} é suficiente para atender à demanda

6: $t_e \leftarrow \lceil D/C_e \rceil$;

7: $t_o \leftarrow T - t_e$; //A CPU pode gastar algum tempo ociosa

8: **else if** $(D > C_c.T)$ **then**

9: // A demanda é maior do que c_{caut} pode atender

10: $t_c \leftarrow T$;

11: **else**

12: // A demanda pode ser atendida combinando c_{eff} e c_{caut}

13: $t_e \leftarrow \lceil (C_c.T - D)/(C_c - C_e) \rceil$;

14: $t_c \leftarrow T - t_e$.

15: **end if**

Conforme observado no Algoritmo 1, o tempo a ser gasto pela CPU em cada estado é escolhido baseado na demanda de encaminhamento estimada para o intervalo de tempo, D, o tamanho do intervalo de tempo, T, e a capacidade de encaminhamento dos estados *Efficient* e *Cautious*, C_e e C_c, respectivamente. Como pode-se observar nas linhas 4-7 do Algoritmo 1, quando a demanda estimada é baixa, ou seja, a capacidade de encaminhamento do estado *Efficient* é suficiente para atendê-la, o ECO planeja gastar o tempo necessário para atender a demanda no estado *Efficient* e, para obter maior economia de energia, configura a CPU para o estado *Otiose*

durante o restante do intervalo de tempo. Caso a demanda estimada seja superior à capacidade máxima de encaminhamento da CPU, o ECO mantém a CPU no estado *Cautious* durante todo o intervalo de tempo para minimizar as perdas (linhas 8-10 do Algoritmo 1). O último caso compreende as situações onde a demanda estimada é inferior à capacidade de encaminhamento da CPU no estado *Cautious* mas superior à capacidade de encaminhamento que pode ser atingida utilizando-se apenas o estado *Efficient*. Nesse caso, o ECO utiliza-se dos dois estados para atender à demanda (linhas 11-15 do Algoritmo 1).

Um ponto importante para o funcionamento do ECO é como é feita a estimativa de demanda de encaminhamento de pacotes para o intervalo sendo planejado. É desejável que a estimativa esteja o mais próxima possível da realidade, pois erros na estimativa causam efeitos indesejáveis. Estimativas acima da demanda real fazem o mecanismo planejar maior capacidade de encaminhamento de pacotes que o necessário, desperdiçando energia e, em casos de falsa escassez de recursos, privando redes virtuais de utilização de recursos disponíveis. Por outro lado, estimativas abaixo da demanda real fazem com que haja escassez de recursos para encaminhar os pacotes das redes virtuais, aumentando o uso dos buffers e, em casos extremos, causando o descarte de pacotes. Por considerar menos danoso superestimar a demanda, utiliza-se um método de estimação baseado no método de média de janela deslizante (WMA - *Window Moving Average* [47]) que, confrontado com análise com dados obtidos de tráfego real, apresentou estimativas de demanda pouco acima da demanda real na grande maioria das estimativas. O método utilizado nesse trabalho consiste em utilizar o valor médio de demanda real medida no intervalo de tempo anterior e o valor do desvio padrão para o mesmo intervalo, somá-los e multiplicar pelo tamanho do intervalo de tempo. O método diferencia-se do método de WMA padrão por adicionar à media o valor de desvio padrão para tornar a estimativa, em geral, pouco superior ao valor real. Formalmente, o estimador utilizado é definido como

$$D = (m + d).T \tag{4.1}$$

onde D é a demanda estimada, m é a demanda média de pacotes medida no intervalo anterior de tempo, d é o desvio padrão das demandas de tempo do intervalo anterior e T é o tamanho do intervalo de tempo. O ECO também possui um fator multiplicando a demanda estimada, que pode ser usado para configurá-lo para ser mais agressivo na economia de energia ou mais cauteloso e priorizar o encaminhamento de pacotes.

O uso do estado *Otiose* em servidores com alguma rede virtual associada necessariamente implica no ocasional aumento da latência dos pacotes, visto que pacotes recebidos enquanto a CPU está no estado *Otiose* são armazenados no buffer. Como não é viável alterar o estado da CPU cada vez que um pacote é recebido devido

à sobrecarga associada a chaveamentos de estado da CPU que isso geraria, é necessário lidar com as demandas de tráfego em determinados intervalos de tempo. Dessa forma o ECO atua planejando o tempo gasto em cada estado da CPU no intervalo de tempo, de maneira a atender a média do tráfego daquele intervalo. Para minimizar o aumento de latência causado pelo uso do estado *Otiose*, ao controlar os estados da CPU durante um intervalo de tempo, o ECO alterna continuamente o estado da CPU entre os estados planejados para o intervalo de tempo, respeitando contudo o total de tempo planejado para ser gasto em cada estado da CPU.

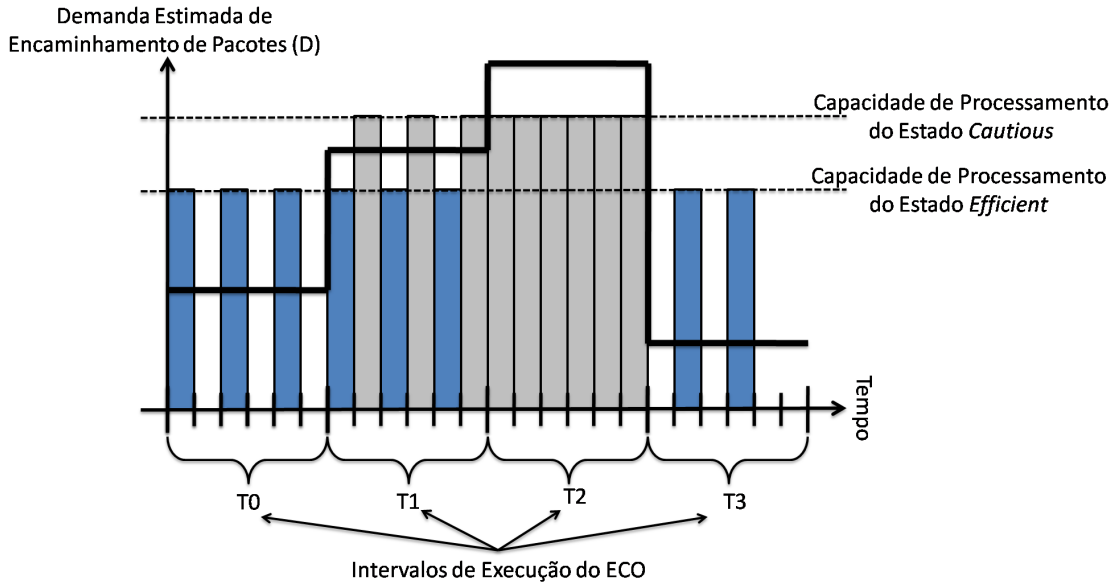


Figura 4.2: Execução do ECO.

A Figura 4.2 exemplifica a execução do ECO em quatro diferentes intervalos de tempo, representados por T0 - T3. As caixas mais baixas representam a CPU no estado *Efficient*, as caixas mais altas representam a CPU no estado *Cautious*, os espaços em branco representam a CPU no estado *Otiose* e a densa linha negra do gráfico representa a demanda estimada de encaminhamento de pacotes. As alturas das caixas representam a capacidade de encaminhamento da CPU em cada estado. Conforme anteriormente descrito, o estado *Otiose* é um estado de economia de energia e não encaminha pacotes. No intervalo de tempo T0 a demanda estimada está dentro da primeira condição do ECO, linhas 4-7 do Algoritmo 1, e a demanda estimada é atendida utilizando-se o tempo necessário para atender a demanda estimada no estado *Efficient* e o restante do tempo do intervalo no estado *Otiose*, de forma a economizar energia. No intervalo de tempo T1, a demanda estimada aumenta e encontra-se dentro da terceira condição do ECO, linhas 11-15 do Algoritmo 1, de forma que a demanda estimada é atendida combinando-se o uso do estado *Efficient* com o estado *Cautious*. No intervalo de tempo T2 demanda estimada novamente aumenta, ultrapassando a capacidade de encaminhamento do servidor e encontrando-se

dentro da segunda condição do ECO, linhas 8-10 do Algoritmo 1, de forma que o estado *Cautious* é utilizado durante todo o intervalo de tempo. Finalmente em T3 a demanda estimada sofre uma súbita diminuição e volta a estar dentro da primeira condição do ECO, sendo atendida pelos estado *Efficient* e *Otiose* combinados.

Apesar de ser intuitivamente mais simples não utilizar as heurísticas do ECO e, simplesmente ajustar periodicamente a CPU para o estado com capacidade de encaminhamento suficiente para cobrir o pico da demanda estimada, esta abordagem acaba sendo freqüentemente muito conservadora, provendo muito mais poder de processamento que o necessário e, dessa forma, desperdiçando energia. Isso deve-se ao fato do estado de CPU a ser escolhido ter de ser selecionado para atender a demanda de pico, de forma que na maior parte do intervalo de tempo, o estado da CPU possui capacidade de encaminhamento e consumo energético superiores ao necessário, causando desperdício de energia. Outro ponto importante é reforçar que o ECO utiliza no máximo dois estados da CPU por intervalo de tempo, visto que não é razoável utilizar o estado *Otiose* em situações de alta demanda ou o estado *Cautious* em períodos de baixa demanda.

O ECO é sensível ao tamanho do buffer utilizado. Um aumento no tamanho do buffer possibilita avaliar a demanda das redes virtuais com menor freqüência, provendo maior economia de energia em troca de aumento da latência. A diminuição do tamanho do buffer provoca os efeitos contrários.

É importante lembrar que a troca de freqüência/voltagem de operação da CPU demora poucos μs e, como o MDR troca poucas vezes o estado de operação da CPU em um período de dezenas ou até centenas de ms, o impacto da troca de estados pode ser negligenciado.

4.2 O Mecanismo de Dinâmica Lenta (MDL)

Esta seção descreve o Mecanismo de Dinâmica Lenta (MDL). O MDL atua no cluster como um todo e objetiva economizar energia minimizando a quantidade de servidores encaminhando pacotes das redes virtuais e desligando os servidores ociosos. Como MDL, propõe-se o uso do *Automatic Load Organizer for Cluster (ALOC)*.

O ALOC foi inspirado nas idéias de utilizar o recurso de migração em ambientes de redes virtuais para balanceamento de carga e o recurso de desligar servidores em clusters/datacenters para economia de energia, conforme utilizados em diversos trabalhos apresentados no Capítulo 2. A idéia básica do ALOC é consolidar as redes virtuais nos servidores mais eficientes do ponto de vista energético, desligando os servidores mais ineficientes para economizar energia e mantendo uma certa capacidade de processamento ociosa em servidores menos eficientes para a absorção de

eventuais variações súbitas na demanda de processamento.

O ALOC é periodicamente executado e possui três etapas para decidir as ações de controle a serem executadas. A primeira etapa consiste em migrar redes virtuais para desafogar servidores sobrecarregados. A segunda etapa busca consolidar o processamento das redes virtuais nos servidores mais eficientes, retirando redes virtuais dos servidores mais ineficientes. A terceira etapa consiste em desligar servidores ineficientes ociosos para a economia de energia e em manter ligados ou até mesmo religar servidores ociosos de forma a manter uma certa capacidade de processamento ociosa para absorver eventuais aumentos súbitos na demanda de processamento. A lógica do MDL-ALOC está descrita no Algoritmo 2.

Algorithm 2 MDL-ALOC

Require: Estado atual do cluster e demandas estimadas das redes virtuais;
Ensure: Planos para consolidar as redes virtuais, desligar e religar servidores;
1: ———————————————————
2: **for** cada servidor sobrecarregado no cluster **do**
3: *planoA* ← "migre redes virtuais até não estar mais sobrecarregado";
4: **end for**
5: **for** cada servidor s na lista de servidores ordenados por ineficiência energética **do**
6: **for** cada servidor r na lista de servidores ordenados por eficiência energética **do**
7: **if** $(s = r)$ **then**
8: Pare o loop interno;
9: **else**
10: *planoA* ← "migre redes virtuais para r sem sobrecarregá-lo";
11: **end if**
12: **end for**
13: **end for**
14: *planoB* ← "desligue os servidores ociosos ou religue servidores desligados de acordo com o IDT e com o estado hipotético do servidor após a execução do *planoA*";
15: Return *planoA* e *planoB*.

O primeiro passo, com o objetivo de eliminar a sobrecarga dos servidores, inicia-se na linha 2 do Algoritmo 2. Nesse passo o MDL-ALOC verifica se há servidores ligados com uso de CPU acima do limiar definido como *OVerload Threshold (OVT)*. Se houver, o MDL-ALOC cria um primeiro plano descrevendo migrações de redes virtuais de servidores sobrecarregados para outros com capacidade ociosa, garantindo que as migrações não deixem nenhum servidor de destino sobrecarregado, conforme mostrado na linha 3 do Algoritmo 2.

O segundo passo, que tem o objetivo de consolidar a carga nos servidores mais energeticamente eficientes do cluster, inicia-se na linha 5. A eficiência energética de um servidor é definida por um índice que considera os três estados da CPU utilizados pelo MDR-ECO. O índice de eficiência do servidor, I_e, é definido pela equação

$$I_e = \Big((P_e/C_e).(P_c/C_c).(P_o)\Big)^{-1} \tag{4.2}$$

onde P_e, P_c, e P_o são as potências de consumo com a CPU nos estados *Efficient*,

Cautious, e *Otiose*, respectivamente, enquanto C_e e C_c são as capacidades de encaminhamento de pacotes nos estados *Efficient* e *Cautious*, respectivamente. Nesse passo, o MDL-ALOC varre todos os servidores ligados do menos eficiente para o mais eficiente energeticamente (linha 5 no Algoritmo 2). Para cada servidor, o MDL-ALOC efetua um segundo plano buscando migrar as redes virtuais do servidor para servidores mais eficientes do ponto de vista energético (linhas 6-12 do Algoritmo 2). Ao final deste passo, o MDL-ALOC possui um plano com as redes virtuais consolidadas nos servidores mais eficientes, deixando os servidores mais ineficientes ociosos.

O terceiro passo possui o objetivo de desligar os servidores ociosos para maximizar a economia de energia, tomando o cuidado de manter uma certa capacidade de processamento ociosa para absorver aumentos súbitos na demanda, visto que o tempo para o religamento de um servidor é consideravelmente longo, na ordem de minutos [51]. Esse passo decide se todos os servidores ociosos devem ser desligados ou se um servidor ocioso deve ser deixado ligado para absorver um possível aumento inesperado na demanda de encaminhamento. No terceiro passo, o MDL-ALOC verifica o uso de CPU do último servidor com redes virtuais alocadas. Caso o uso de CPU esteja acima do limiar definido como *IDle Threshold (IDT)*, o MDL-ALOC mantém ligado o servidor ocioso de melhor eficiência energética e desliga os servidores ociosos restantes. Caso contrário, todos os servidores ociosos são desligados (linha 14 do Algoritmo 2).

Um exemplo de execução do ALOC encontra-se na Figura 4.3.

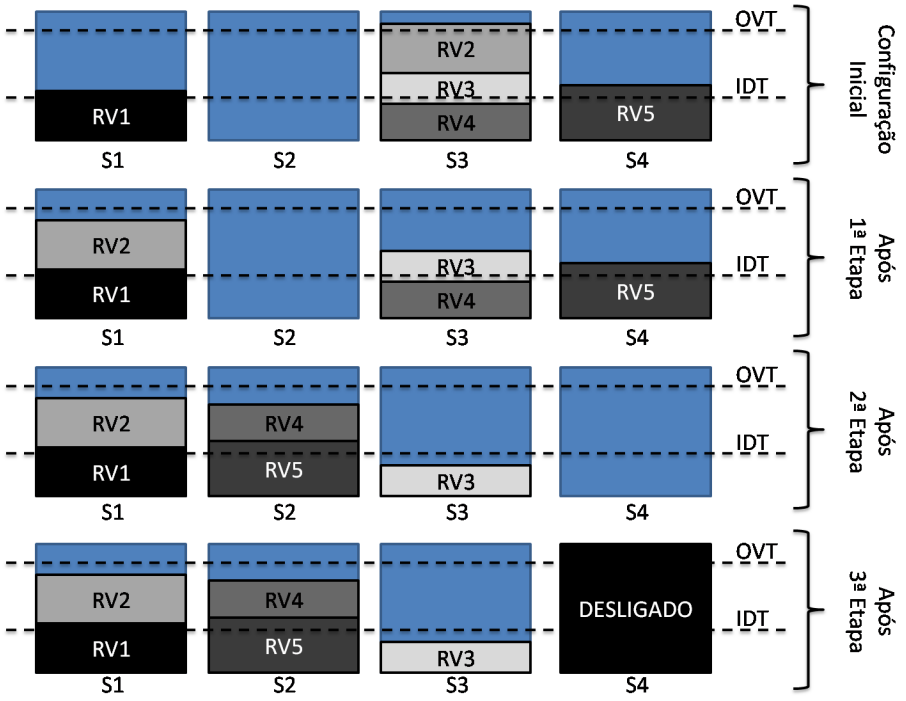


Figura 4.3: Exemplo de execução do ALOC.

No exemplo, o roteador de software em cluster é composto pelos servidores S1-S4 e possui as redes virtuais RV1-RV5. A altura das caixas representa a demanda de processamento das redes virtuais e a capacidade de encaminhamento dos servidores. Os servidores estão ordenados de acordo com a eficiência energética em ordem decrescente da esquerda para a direita. Na primeira etapa, o ALOC planeja migrar a RV2 de S3 para S1, visto que a carga de S3 está acima do *OVerload Threshold* (OVT). Na segunda etapa, o ALOC planeja a migração de RV5 e RV4 para S2, visto que S2 é mais eficiente energeticamente do que S3 e S4 além de possuir capacidade ociosa suficiente para atender essas redes virtuais. Na terceira etapa, o ALOC desliga S4 visto que a carga de S3 está abaixo do *IDle Threshold* (IDT).

Assim, como para o ECO, a demanda estimada para cada rede virtual é calculada utilizando-se o valor da demanda média de processamento para o intervalo de tempo anterior, somado ao valor do desvio padrão para o mesmo período e multiplicados pelo tamanho do intervalo de tempo.

É importante notar que o ALOC não efetua nenhuma migração de redes virtuais até o fim do planejamento de três etapas para evitar que migrações desnecessárias sejam efetuadas. O ALOC é também a parte do mecanismo proposto responsável pelo suporte à retirada de servidores do cluster de forma transparente. Para a retirada de um servidor do cluster, para manutenção por exemplo, basta configurar manualmente o índice de eficiência do servidor desejado para um valor muito baixo. Dessa forma, todas as redes virtuais deste servidor serão migradas para outros servidores e o ALOC irá desativar o servidor em questão. Para adicionar mais servidores ao cluster, basta informar ao ALOC sobre a inclusão dos novos servidores no cluster e esses serão automaticamente utilizados na próxima execução do MDL.

Após apresentar o ECO e ALOC, é importante verificar o funcionamento dos dois mecanismos para validar suas eficácias. Infelizmente não há implementações disponíveis de roteadores de software em cluster com suporte a redes virtuais, fazendo com que a avaliação dos mecanismos tenha que ser feita através de simulações. Como não foram encontrados simuladores adequados para a simulação do modelo de roteador de software em cluster considerado neste trabalho, optou-se por desenvolver um simulador próprio. O capítulo a seguir apresenta em detalhes o simulador desenvolvido, desde a concepção do modelo de simulação até a implementação do simulador correspondente ao modelo concebido.

Capítulo 5

Simulação do Roteador de Software em Cluster

Uma parte fundamental para avaliar e validar o trabalho desenvolvido é a simulação do mecanismo proposto em diversos cenários, variando tanto a carga quanto os recursos disponíveis no cluster. Para isto, dado que não foram identificados simuladores apropriados para o problema estudado, é necessária a construção de um simulador que modele apropriadamente o problema da distribuição de recursos do cluster relativos ao processamento das redes virtuais. Nesse capítulo é apresentado o modelo de simulação do cluster e são fornecidos detalhes sobre o simulador construído baseado neste modelo.

5.1 Dinâmica do Modelo do Simulador

O problema da alocação de recursos de um roteador de software em cluster pode ser dividido basicamente em duas partes, conforme anteriormente afirmado no Capítulo 4: alocação de banda dos enlaces externos do roteador entre as redes virtuais; e alocação dos recursos relativos ao processamento dos pacotes das redes virtuais. O segundo problema é o objeto de estudo deste trabalho e é modelado no simulador.

Primeiramente devem-se analisar as partes que constituem um roteador de software em cluster do modelo utilizado como referência e apresentado na Seção 3.2: os enlaces externos, o comutador programável, os enlaces internos e os servidores. Os enlaces externos bem como o comutador programável devem ser gerenciados pelo mecanismo de alocação de banda entre as redes virtuais e podem ser deixados de fora do modelo do problema de processamento dos pacotes. Os enlaces internos não devem ser um gargalo para o roteador de software em cluster, visto que a banda para comunicação entre os servidores tende a ser muito superior à disponível nos

enlaces externos. Mesmo para os casos extremos, em que os enlaces externos do roteador têm alta capacidade ou em que o roteador de software em cluster é composto por muitos servidores, a carência por banda na interconexão dos servidores do cluster pode ser suprida aplicando-se técnicas para a distribuição do tráfego interno do cluster, conforme explorado em Routebricks [19]. Em Routebricks, avalia-se algumas técnicas para interconectar os servidores do cluster de forma a atender a taxa requerida por cada porta externa. O artigo explora desde técnicas de interconexão mais simples, como conectar todos os servidores entre si formando um grafo completo, utilizada para o caso de roteadores de software em cluster compostos por poucos servidores, até técnicas mais sofisticadas que permitem a interconexão entre todos os servidores utilizando um número limitado de portas em cada servidor e escalando o cluster para um grande número de servidores, como a interconexão em borboleta [59] k-ária n-ária generalizada. Desta forma, pode-se considerar no modelo que a banda entre os servidores do cluster não é um gargalo e por isso não precisa ser considerada como recurso a ser gerenciado ao analisar o problema da alocação de recursos relativos ao processamento dos pacotes das redes virtuais. Assim, os únicos recursos a serem gerenciados são os recursos dos servidores que compõem o cluster, mais especificamente o processador e a memória RAM dos servidores, visto que estes são os recursos mais utilizados para o processamento dos pacotes.

Além dos recursos consumidos pelos servidores, é necessário definir os eventos que compõem a dinâmica do modelo. Os eventos estão associados às etapas do processamento dos pacotes, compreendidos entre a recepção dos pacotes na interface de rede do servidor e seu reenvio para o próximo salto, conforme ilustrados no diagrama de atividades da Figura 5.1.

O primeiro evento é o recebimento de um pacote em trânsito proveniente de uma rede virtual. Seguido a este evento está o de armazenar o pacote recebido no buffer unificado que recebe todos os pacotes provenientes das redes virtuais, conforme descrito anteriormente na Seção 3.1. Caso haja espaço disponível no buffer o pacote é armazenando e caso o buffer esteja cheio o pacote é descartado. O último evento modelado referente à tarefa básica de encaminhamento de pacotes é o processamento do pacote para a escolha do próximo salto.

Passada a modelagem dos eventos básicos para o encaminhamento de pacotes, deve-se adicionar à dinâmica do modelo os eventos que representam as particularidades do ambiente de cluster com redes virtuais. Para contemplar a migração de redes virtuais, é adicionado um evento de migração. A migração de redes virtuais considerada neste trabalho segue os moldes da migração de redes virtuais proposta em VROOM [53], descrita a seguir. No VROOM, as redes virtuais são divididas em duas partes: um plano de dados e um plano de controle. O plano de dados é responsável pelo encaminhamento de pacotes enquanto o plano de controle é responsável pela

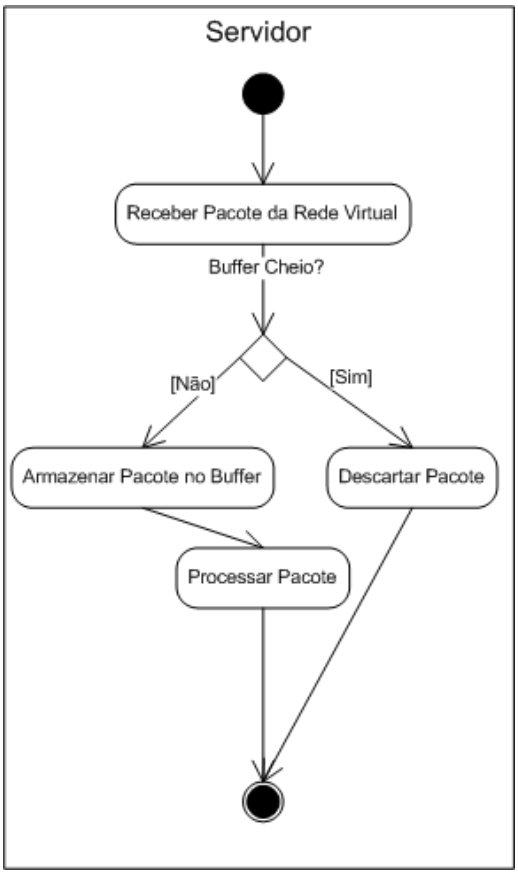


Figura 5.1: Diagrama de atividades do encaminhamento de pacotes.

troca de mensagens do protocolo de roteamento, bem como pela atualização da tabela de encaminhamento. A separação em dois planos permite a migração da rede virtual sem gerar distúrbios no tráfego da rede virtual sendo migrada. O processo de migração ocorre em três etapas: migração do plano de controle, clonagem do plano de dados e migração dos enlaces. A fase de migração do plano de controle transfere a imagem da máquina virtual com o roteador, contendo os arquivos binários, os arquivos de configuração e outros. Para viabilizar a migração do plano de controle sem interrupção do encaminhamento, primeiramente é estabelecido um túnel do servidor de origem para o servidor de destino para que passem as mensagens de controle do protocolo da rede virtual. Em seguida é iniciada a transferência de memória do servidor de origem para o servidor de destino, que acontece em duas partes. Na primeira parte, chamada de pré-cópia, o servidor de destino recebe uma cópia da memória idêntica à da máquina virtual contendo o plano de controle no momento em que o processo de migração foi iniciado. Como a máquina virtual continua em funcionamento durante essa cópia, diversas páginas de memória sofrem mudanças durante o processo de transferência da imagem, e o mecanismo de migração monitora quais são as páginas modificadas. Ao final da primeira cópia, o mecanismo de migração envia as páginas modificadas para o servidor de destino, e continua a

monitorar quais páginas de memória são alteradas pela máquina virtual em funcionamento. Este processo iterativo continua por mais algumas rodadas até que o número de páginas modificadas durante a transferência anterior esteja abaixo de um determinado limiar. Neste momento ocorre a segunda parte da transferência de memória, onde a máquina virtual com o plano de controle é brevemente paralisada para a transferência do resíduo de memória modificado durante a cópia de memória da iteração anterior. Terminada a cópia do resíduo de memória, inicia-se a etapa de clonagem do plano de dados que consiste no repovoamento da tabela de encaminhamento do roteador no servidor de destino. Como a cópia do plano de dados é um processo demorado, durante esta etapa o encaminhamento dos pacotes continua sendo feito pelo plano de dados no servidor de origem, controlado remotamente pelo plano de controle no servidor de destino através dos túneis estabelecidos no início do processo de migração. Ao final da cópia do plano de dados os enlaces da rede virtual são migrados e o servidor de destino assume completamente a responsabilidade pelo tráfego da rede virtual, liberando os recursos previamente utilizados no servidor de origem.

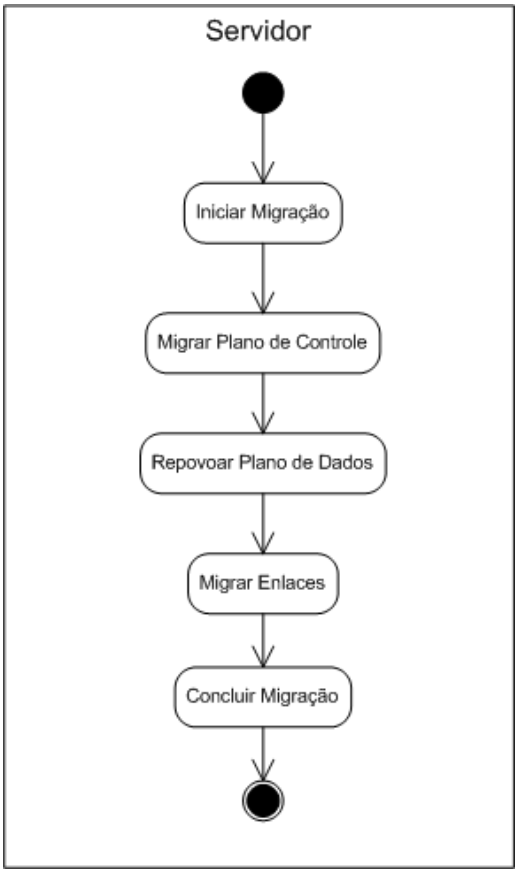
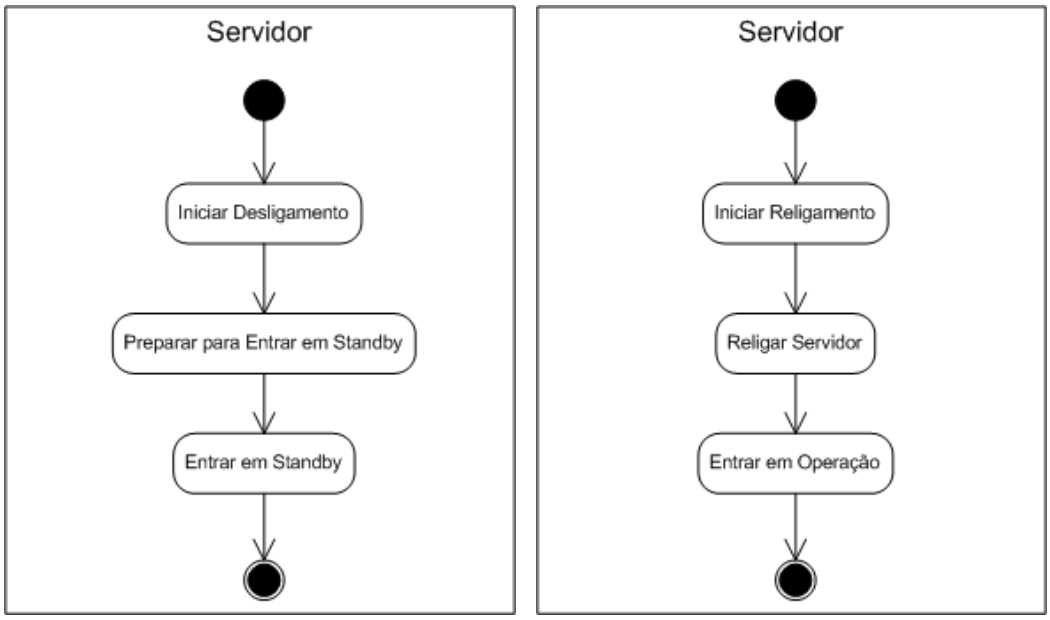


Figura 5.2: Etapas da migração de um roteador virtual no simulador.

Para modelar este processo de migração de uma rede virtual, o simulador gera sobrecargas impactando no consumo de memória do servidor de destino e em seu pro-

cessamento, visto que estes recursos são necessários para a cópia da máquina virtual e execução remota do plano de controle, e adicionando sobrecarga de processamento no servidor de origem, devido ao esforço para efetuar a migração, envolvendo tarefas como a cópia das páginas de memória e o controle de quais páginas de memória são modificadas durante a migração para posterior reenvio ao servidor de destino. Durante o processo de migração, a memória demandada pela rede virtual, bem como a carga de processamento necessária para o encaminhamento de pacotes da rede virtual, continuam a ser contabilizados no servidor de origem, visto que o plano de dados do servidor de origem é responsável pelo encaminhamento dos pacotes até o fim do processo de migração. As etapas modeladas no simulador que representam o processo de migração de um roteador virtual são exibidas no diagrama de atividades da Figura 5.2. As sobrecargas aplicadas devido ao processo de migração são baseadas nos valores apresentados em [53] e estão detalhados na Seção 7.1. Para verificar a correta alocação das redes virtuais quanto ao uso de memória, o simulador também modela a situação em que a memória RAM demandada pelos roteadores virtuais supera a memória que o servidor possui. Para isso, quando essa situação ocorre, o simulador termina a execução de roteadores virtuais até que o uso de memória esteja dentro da capacidade do servidor e gera eventos de terminação de roteador virtual para os roteadores virtuais terminados. O evento de terminação de roteador virtual serve para depurar o MDL. A não ser em situações de simulação mal definidas onde o consumo de memória das redes virtuais excede a memória disponível no cluster, a ocorrência de um evento de terminação de roteador virtual indica mal funcionamento do MDL na situação simulada, indicando que o MDL ordenou uma distribuição de roteadores virtuais no cluster incompatível com a memória disponível nos servidores.

A última particularidade a ser modelada é o desligamento e o religamento dos servidores. Um servidor somente pode ser desligado quando não possui redes virtuais sob sua responsabilidade para que não sejam causadas perdas de pacotes. O processo de desligamento dura o tempo necessário para o servidor entrar em estado de *standby*. Durante o desligamento o servidor não está hábil a processar pacotes, todo seu processamento está associado a tarefas relacionadas ao seu desligamento e considera-se que seu consumo energético é equivalente ao estado ativo da CPU com a mais baixa capacidade de processamento. Esta escolha na modelagem justifica-se pelo fato da maior parte das atividades do servidor ocorrer no disco rígido, salvando os dados importantes que se encontram em RAM para que não sejam perdidos após o desligamento. Uma vez concluído o processo de desligamento, o servidor mantém-se em *standby*, estado em que possui baixo consumo energético, apenas o suficiente para monitorar pedidos de religamento via rede local (*wake-on-LAN*). Para o religamento do servidor os efeitos na dinâmica do simulador são similares.



(a) Desligamento. (b) Religamento.

Figura 5.3: Etapas do desligamento/religamento de um servidor no simulador.

O simulador modela o religamento do servidor considerando que seu consumo é equivalente ao estado ativo da CPU com a mais baixa capacidade, visto que a principal tarefa é o restabelecimento do sistema operacional, carregando o kernel em memória a partir do disco rígido e iniciando os serviços. Durante o religamento o servidor não está apto a processar pacotes de nenhuma rede virtual, podendo fazê-lo apenas após o processo de religamento ser concluído e o recebimento de alguma rede virtual para a sua responsabilidade. As três etapas modeladas no simulador que representam o processo de desligamento/religamento de um servidor são exibidas nos diagramas de atividade da Figura 5.3. A duração dos processos de desligamento e religamento, bem como os consumos energéticos estão detalhados na Seção 7.1.

Concluída a descrição do modelo utilizado e da dinâmica deste modelo será apresentada em seguida a descrição do simulador desenvolvido.

5.2 O Simulador

Após a definição do modelo, passa-se para a construção do simulador. O simulador é desenvolvido na linguagem de programação python [60] e totaliza aproximadamente 7400 linhas de código fonte. A estrutura do simulador é basicamente dividida em quatro partes: uma parte central responsável pela execução das rodadas de simulação, uma parte responsável pela modelagem do cluster, uma parte responsável pela execução dos mecanismos de controle e uma parte responsável pelo armazenamento dos dados das rodadas de simulação. As quatro partes do simulador, bem

como um diagrama de classes simplificado das classes básicas que as constituem encontram-se na Figura 5.4.

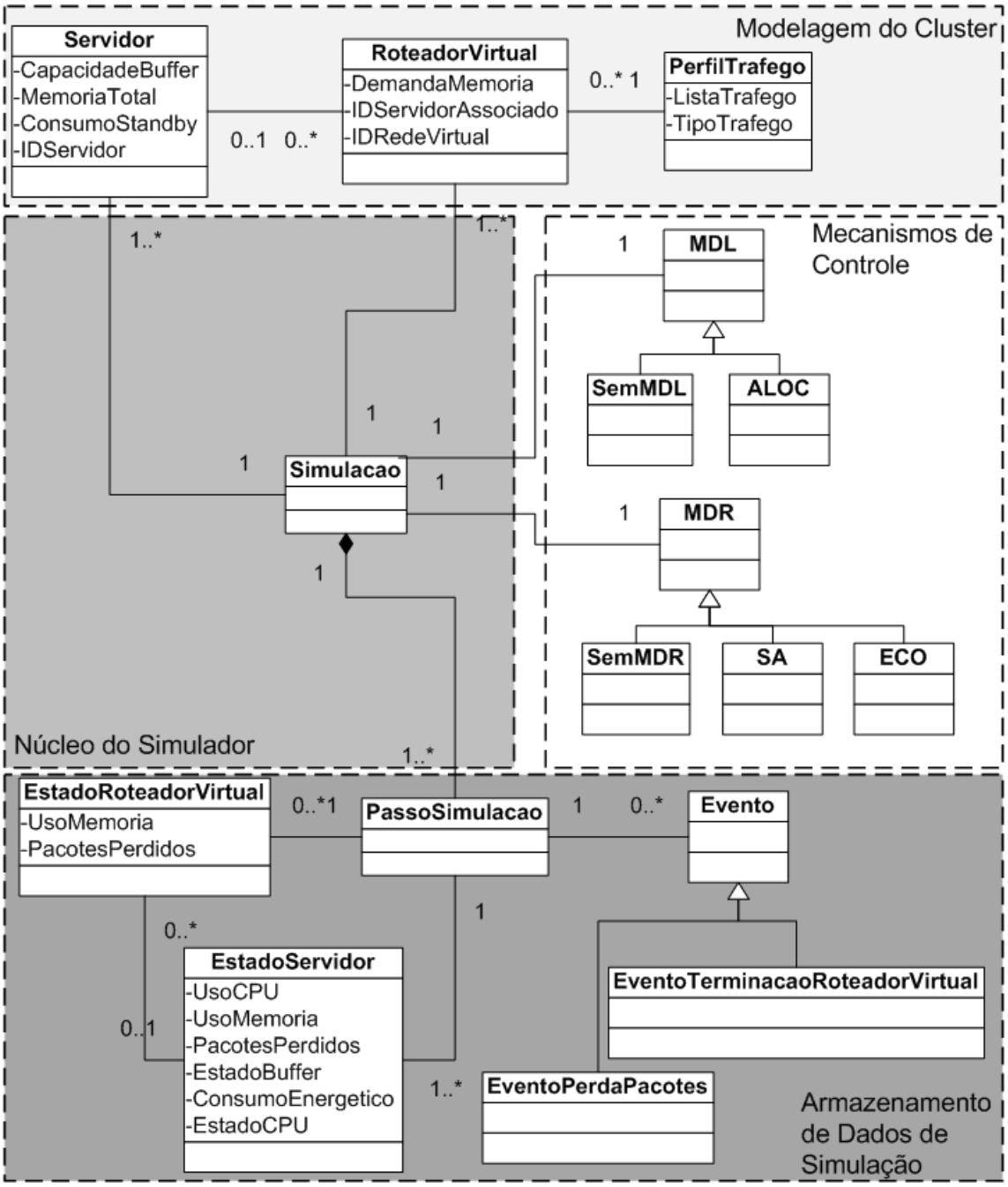


Figura 5.4: Diagrama de classes simplificado do simulador.

A parte central, o núcleo do simulador, é constituída pela classe Simulacao. Essa classe é responsável pela avaliação dos estados dos roteadores virtuais e dos servidores do cluster.

A parte do armazenamento dos dados de simulação é responsável por acumular os dados de todas as rodadas da simulação, permitindo a verificação de qualquer estado de qualquer roteador virtual ou servidor em qualquer passo simulado. Essa parte do

simulador é composta pelas classes de PassoSimulacao, EstadoServidor, EstadoRoteadorVirtual, Evento, EventoTerminacaoRoteadorVirtual e EventoPerdaPacotes, conforme pode-se verificar na Figura 5.4. A classe PassoSimulacao é responsável por centralizar o acesso a todos os dados de uma rodada de simulação, que estão armazenados nas demais classes dessa parte do simulador. A classe EstadoServidor é responsável por armazenar quais roteadores virtuais estão associados ao servidor no passo de simulação em questão, qual é o consumo energético do servidor, o estado utilizado na CPU, o uso de memória RAM, o estado do buffer de recepção de pacotes e a quantidade de pacotes que foram perdidos no servidor, que corresponde ao total de pacotes perdidos pelos roteadores virtuais associados ao servidor em questão. A classe EstadoRedeVirtual guarda o consumo de memória da rede virtual e o número de pacotes perdidos. A classe Evento armazena informações referentes a eventos ocorridos durante o passo de simulação e especializa-se em duas classes. A primeira classe, EventoPerdaPacotes, armazena informações específicas de perda de pacotes em um roteador virtual. Já a segunda classe, EventoTerminacaoRoteadorVirtual, armazena informações específicas de quando um roteador virtual é terminado por falta de RAM disponível no servidor a qual o roteador está associado.

A parte dos mecanismos de controle possui as classes com as implementações dos mecanismos de dinâmica lenta e rápida, bem como as classes com o comportamento padrão do cluster sem mecanismo de controle. Em cada simulação apenas uma classe está ativa dentre as especializações da classe de MDR e MDL. Para MDL estão disponíveis as especializações de SemMDL e ALOC, enquanto para a classe de MDR estão disponíveis as especializações de SemMDR, SA e ECO. Quando não há uso do MDL, a alocação de roteadores virtuais no cluster permanece estática durante toda a simulação, idêntica à distribuição inicial dos roteadores virtuais feita nos procedimentos de inicialização do simulador. A distribuição inicial dos roteadores virtuais é feita através de *round-robin* entre os servidores do cluster com recursos disponíveis para as demandas iniciais dos roteadores virtuais. Adotou-se a configuração inicial de *round-robin* por ser uma técnica comumente utilizada na distribuição de tarefas em sistemas distribuídos de larga escala [51]. Quando não há uso do MDR, o estado da CPU do servidor fica continuamente no estado de maior capacidade de encaminhamento de pacotes e de maior gasto energético. Essa escolha foi feita porque a alocação estática da CPU nesse estado fornece o melhor desempenho de encaminhamento e maior consumo energético possíveis, servindo como parâmetros limítrofes na comparação de economia de energia e encaminhamento de pacotes para o MDR proposto.

A última parte é a parte de modelagem do cluster. Esta parte contém as classes associadas aos dados de entrada e aos parâmetros utilizados durante a computação das rodadas do simulador, conforme mostrados na Tabela 5.1. A classe Servidor

Elemento modelado	**Dados de Entrada e Parâmetros de Simulação**
Servidor	Gasto energético em *standby*, capacidade máxima do buffer, memória RAM total, parâmetros da CPU, roteadores virtuais associados
Roteador Virtual	memória RAM demandada, identificador do servidor associado, identificador do perfil de tráfego associado
Perfil de Tráfego	Demandas de tráfego dos diversos perfis e memória RAM demandada pelos diversos perfis

Tabela 5.1: Parâmetros e dados de entrada dos módulos de modelagem do cluster.

possui os parâmetros da CPU (capacidade de encaminhamento de pacotes em cada estado e consumo energético do servidor para cada estado da CPU quando ela está ativa ou ociosa), memória RAM total do servidor, capacidade máxima de armazenamento do buffer de recepção, gasto energético do servidor no estado de *standby* e a relação dos roteadores virtuais associados ao servidor. A classe RoteadorVirtual possui a quantidade de memória RAM demandada pelo roteador virtual, a identificação de qual servidor é responsável pelo processamento dos seus pacotes e um perfil de tráfego associado. A classe PerfilTrafego armazena diversos tipos de perfis de tráfego (demanda de pacotes por passo de simulação) e o identificador do tipo de tráfego utilizado.

O simulador é baseado em passos de simulação equivalentes a um certo intervalo fixo de tempo, que pode ser configurado. A escolha por fazer o simulador baseado em passos que equivalem a intervalos fixos de tempo, ao invés de fazê-lo orientado a eventos, foi feita para simplificar a implementação de seus módulos, em especial dos módulos de alocação de recursos dos mecanismos propostos. No escopo deste trabalho o intervalo de tempo é utilizado com o valor de 10 ms, valor que apresenta resolução suficiente para avaliar e verificar os efeitos do controle do Mecanismo de Dinâmica Rápida na dinâmica do cluster. Dessa forma, o arquivo que descreve o padrão de tráfego de uma rede virtual é definido como um arquivo texto contendo um valor numérico por linha. Cada valor numérico representa a demanda de encaminhamento de pacotes durante o intervalo de tempo para a rede virtual. Para cada passo de simulação é lido um valor numérico dos arquivos que descrevem as demandas de tráfego e, passa-se para a linha seguinte de cada arquivo durante o cálculo do próximo passo de simulação. Os perfis de tráfego utilizados nas simulações são obtidos através de dados capturados de tráfego real. O processo de geração dos arquivos que descrevem as demandas de tráfego das redes virtuais é detalhado na subseção a seguir.

5.2.1 Arquivos de Tráfego das Redes Virtuais

Como mencionado na seção anterior, os perfis de tráfego das redes virtuais foram gerados a partir de dados reais. Foram aplicados três conjuntos de dados na geração dos perfis de tráfego de redes virtuais utilizados para testar os mecanismos deste trabalho, um obtido da CAIDA e os outros dois da RNP.

Tráfego Real - CAIDA

O primeiro conjunto de dados de tráfego real é obtido da CAIDA(*Cooperative Association for Internet Data Analysis*) [61], um grupo dedicado à análise do tráfego da Internet. Dentre as suas diversas iniciativas, a CAIDA possui um projeto de monitoramento passivo de tráfego da Internet em tempo real com diversos pontos de coleta. Dentre os pontos de coleta há alguns dentro de grandes provedores de acesso à Internet (*Internet Service Providers - ISPs*) nos Estados Unidos. Os dados coletados pela CAIDA são compartilhados entre pesquisadores de todo o mundo e utilizados em projetos que vão desde modelos de caracterização do tráfego da Internet até testes de protótipos de softwares projetados para mitigar danos causados por softwares maliciosos na Internet. Os dados cedidos pela CAIDA e utilizados nesse trabalho consistem em cabeçalhos anonimizados de pacotes capturados durante o período de uma hora em um ponto de coleta de um datacenter da Equinix [62] em Chicago contendo dados no formato da ferramenta de captura TCPDUMP [63], como tamanho do pacote e estampa de tempo com hora de chegada, por exemplo. O ponto de coleta está instalado em um enlace OC192 (9953 Mbps) do *backbone* de um provedor de acesso *Tier1* entre Chicago, Illinois e Seattle, Washington.

Para utilizar os dados da CAIDA no simulador é necessário, entretanto, modificar o formato dos dados e adaptá-los para o formato esperado pelo simulador. Para isso, é necessário utilizar o campo de estampa de tempo com a hora da chegada dos pacotes, convertendo os dados no formato de cabeçalhos de pacotes para um arquivo texto contendo em cada linha o número de pacotes demandados durante o intervalo de tempo equivalente a uma rodada de simulação. Como deseja-se gerar diversos perfis de tráfego para as redes virtuais simuladas é necessário escolher uma forma de classificar o tráfego dos dados disponíveis. Para isso, opta-se por gerar diversos perfis de tráfego considerando uma das possíveis tendências de organização das redes virtuais, onde há uma rede específica com uma pilha de protocolos para atender a um tipo de aplicação/serviço [64]. É necessária ainda a escolha de um método de caracterização de tráfego para efetuar a separação dos serviços nos dados obtidos da CAIDA. Existem diversos métodos, conforme citado no Capítulo 2. Visto que dentro do escopo deste trabalho o objetivo é gerar alguns perfis de tráfego para alguns serviços/aplicações disponíveis na Internet, dentre as opções consideradas, o

método escolhido é o de caracterização de tráfego pela porta de destino dos pacotes. A escolha deve-se ao fato desse método ser bastante utilizado, simples de aplicar, e boa acurácia para a classificação de aplicações conhecidas e bem comportadas [65], além de poder ser utilizado somente com a inspeção dos cabeçalhos dos pacotes individualmente, facilitando o trabalho de classificação. Os serviços escolhidos para a definição dos perfis de tráfego das redes virtuais são o HTTP, filtrando a porta de destino número 80, o HTTPS, filtrando a porta de destino número 443, o FTP, filtrando as portas de destino números 20 e 21, e o acesso a e-mail via SMTP, POP3 e IMAP, filtrando as portas de destino números 25,110 e 143.

Para efetuar testes com 24 horas de duração foram solicitados dados adicionais à CAIDA. Os dados recebidos, devido ao longo período de duração, não possuem a mesma resolução. Esses dados sobre 24 horas de tráfego consistem no valor médio de pacotes transmitidos a cada cinco minutos, e pertencem ao mesmo enlace dos dados de uma hora de duração. Para os arquivos de entrada do simulador foram sintetizados arquivos de demandas de pacotes com resolução de 10 ms e 24 horas de duração. Os arquivos foram sintetizados usando os perfis de HTTP, HTTPS, FTP e e-mail baseados nos dados de uma hora de duração e nos dados adicionais de 24 horas de duração. Os arquivos são sintetizados repetindo 24 vezes os valores de demandas dos arquivos de uma hora de duração e modulando os valores pelos valores de demanda média de 5 minutos disponíveis nos dados de 24 horas. Dessa forma, são obtidos arquivos com demandas que apresentam tanto as variações de demanda instantânea, que podem ser observadas microscopicamente, quanto as variações de uso médio do enlace de acordo com a hora do dia, que podem ser observadas macroscopicamente.

Tráfego Real - RNP

De forma a fazer uma análise mais completa, também foram utilizados dados provenientes de outra rede *backbone*, da Rede Nacional de Ensino e Pesquisa (RNP) [1] provenientes da rede Ipê nas simulações. A rede Ipê é uma rede de alta velocidade que provê conectividade a diversas universidades e instituições de pesquisa. A topologia da rede Ipê é reproduzida na Figura 5.5.

Os dados obtidos da RNP estão no formato NetFlow [66] e correspondem a dois enlaces de 10 Gbps da rede Ipê entre o Rio de Janeiro e São Paulo e entre o Rio de Janeiro e Belo Horizonte. Foram extraídos dados de demanda média de intervalos de 5 minutos para HTTP, HTTPS, FTP e e-mail com a duração total de 24 horas, no formato similar aos dados adicionais fornecidos pela CAIDA. A partir desses dados processados de 24 horas, foram gerados arquivos de entrada do simulador com 24 horas de duração e 10 ms de resolução utilizando a mesma técnica descrita para os dados da CAIDA. A escolha dos enlaces entre o Rio de Janeiro e São Paulo e entre o Rio de Janeiro e Belo Horizonte deve-se ao fato dos dois enlaces possuírem

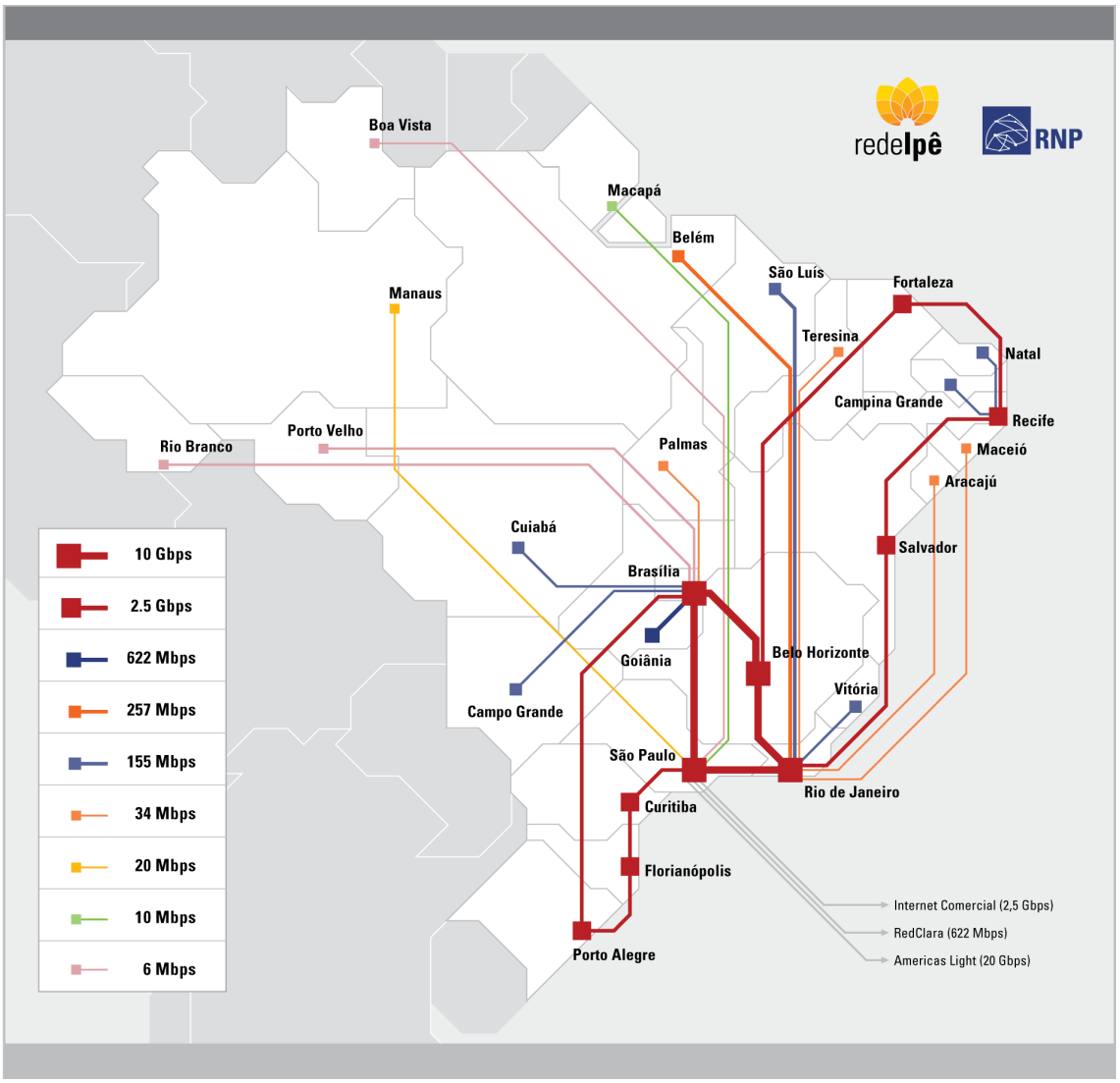


Figura 5.5: Topologia da Rede Ipê da RNP. Retirada do site oficial [1].

a mesma capacidade, mas com médias de utilização diferentes. Em geral, o enlace entre Rio de Janeiro e São Paulo opera com taxa de utilização muito acima do que o enlace entre Rio de Janeiro e Belo Horizonte. Para exemplificar, a Figura 5.6 mostra o perfil do tráfego de entrada e saída do enlace entre Rio de Janeiro e São Paulo durante 24 horas, enquanto a Figura 5.7 mostra o perfil de tráfego de entrada e saída do enlace entre Rio de Janeiro e Belo Horizonte para o mesmo período.

As figuras dos perfis de tráfego foram obtidas a partir de uma ferramenta on-line de monitoramento da rede Ipê [67] disponível no site da rede Ipê. A captura dessas imagens foi efetuada no dia 10 de fevereiro de 2011. Como pode-se observar nas figuras, enquanto a média do tráfego de entrada/saída do enlace com Belo Horizonte foi de 226,78 Mbps/366,20 Mbps, a média do tráfego de entrada/saída do enlace com São Paulo foi de 2,08 Gbps/870,20 Mbps, valores de utilização do enlace muito superiores aos apresentados no enlace entre Rio de Janeiro e Belo Horizonte.

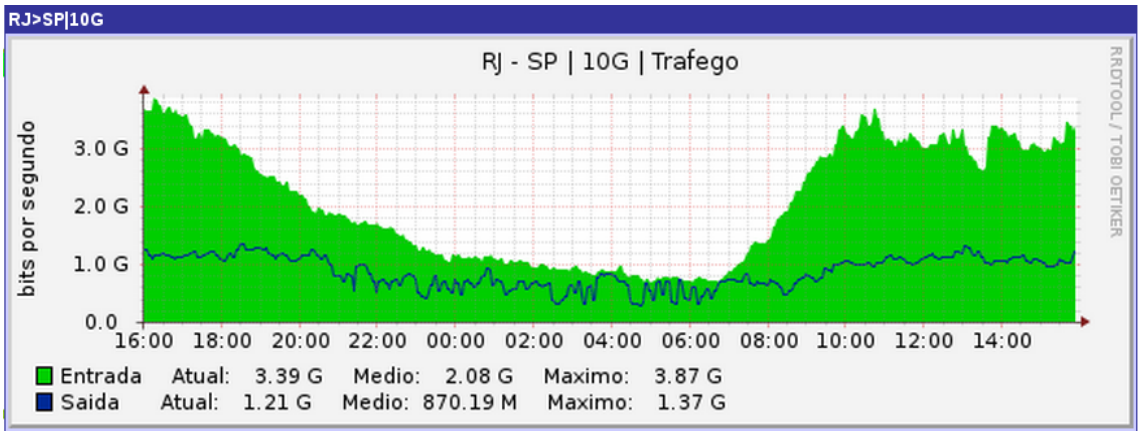


Figura 5.6: Exemplo da variação do tráfego de entrada e saída do enlace entre Rio de Janeiro e São Paulo no período de 24 horas.

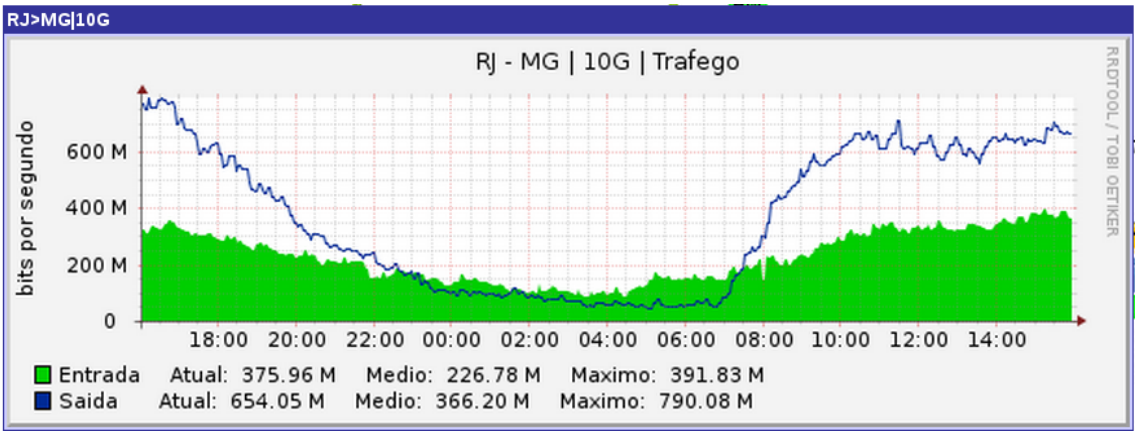


Figura 5.7: Exemplo da variação do tráfego de entrada e saída do enlace entre Rio de Janeiro e Belo Horizonte no período de 24 horas.

Após o detalhamento da geração dos perfis de tráfego das redes virtuais utilizados no simulador e da origem dos dados utilizados como base para a sua geração, o próximo passo para finalizar o detalhamento do simulador consiste na descrição do cálculo de uma rodada de simulação, apresentada na próxima subseção.

5.2.2 Cálculo de Uma Rodada de Simulação

Esta subseção detalha como são efetuados os cálculos das rodadas de simulação. Conforme mostrado no diagrama de atividades da Figura 5.8, o simulador passa por cinco etapas distintas para avaliar uma rodada de simulação: atualização de roteadores virtuais, tratamento das migrações, tratamento do desligamento/religamento dos servidores, cálculo do uso de recursos e execução dos mecanismos de alocação de recursos.

Na primeira etapa, o simulador verifica se ocorreu algum evento de falta de memória na rodada de simulação anterior. Caso algum evento deste tipo tenha

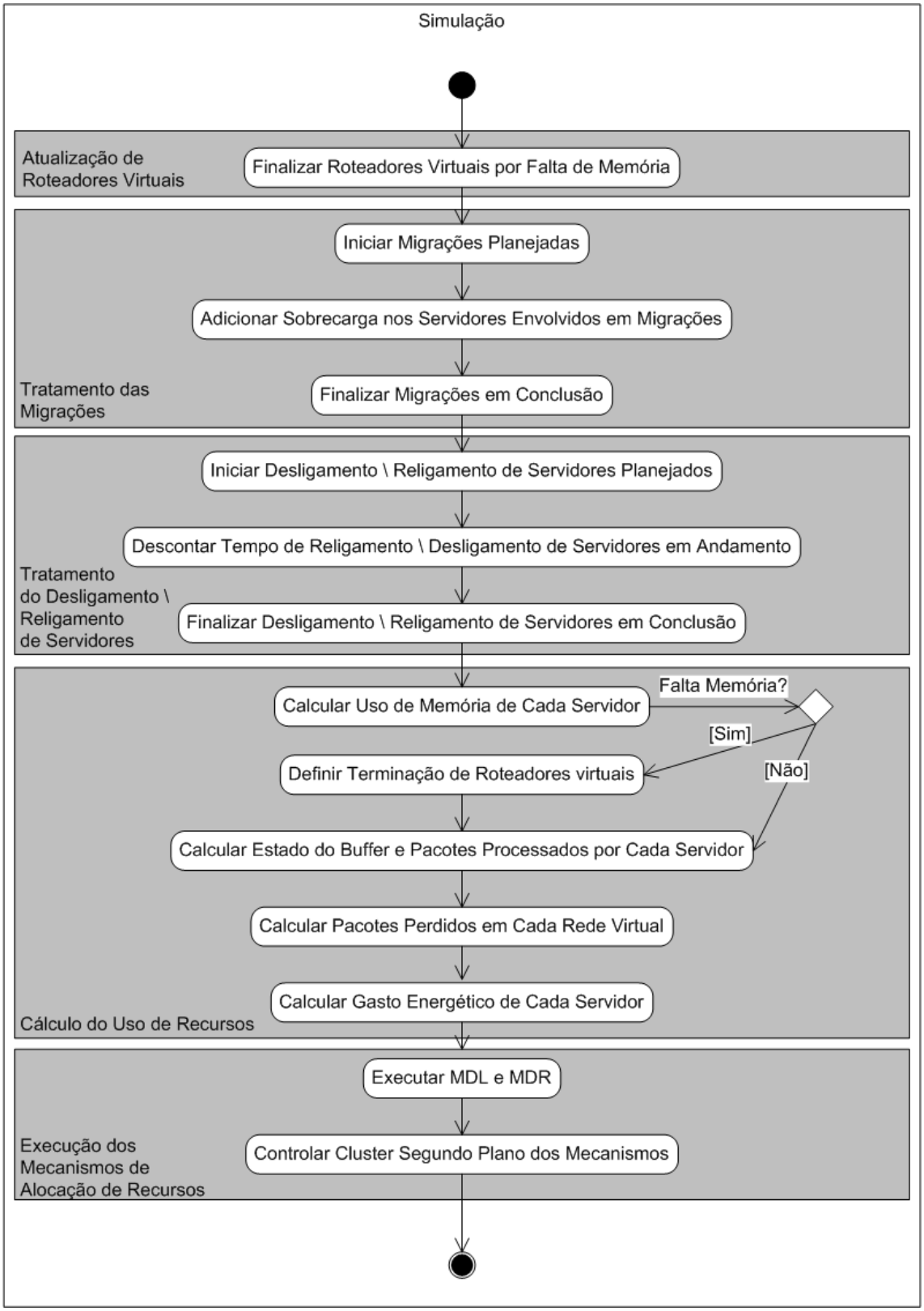


Figura 5.8: Detalhamento do cálculo de uma rodada de simulação.

ocorrido, o simulador encarrega-se de terminar a rede virtual determinada no evento de falta de memória pelo servidor carente do recurso. Embora este aspecto esteja modelado, não deve ser observado com os mecanismos de alocação de recursos fun-

cionando corretamente em um cluster bem dimensionado, visto que esses eventos indicam que os mecanismos de alocação de recursos não estão distribuindo corretamente as redes virtuais dentro do cluster ou que a simulação foi configurada com excesso de redes virtuais no cluster.

A segunda etapa do cálculo da rodada de simulação consiste no tratamento das migrações de redes virtuais. Esta etapa é iniciada verificando-se a existência de pedidos de migração pendentes, gerados pelo Mecanismo de Dinâmica Lenta na rodada de simulação anterior. Caso haja pedidos, estes são retirados de uma fila de pedidos pendentes e o simulador inclui as redes virtuais em uma lista de redes virtuais em migração. A lista de redes virtuais em migração contém as informações sobre as sobrecargas geradas nos servidores de origem e de destino, bem como a contagem do tempo necessário, em número de passos de simulação, para a conclusão da migração de cada rede virtual. Após tratar os pedidos pendentes de migração, o simulador varre a lista de redes virtuais em migração, adicionando as sobrecargas descritas nos estados da rodada de simulação dos servidores apropriados e decrementando o tempo restante de migração de cada rede virtual. A última parte da etapa consiste em retirar da lista de redes em migração as redes que atingem tempo restante de migração igual a zero, associando a rede virtual ao servidor de destino e concluindo assim o processo de migração.

A etapa seguinte consiste no tratamento do desligamento e religamento de servidores. A primeira parte desta etapa é a verificação de pedidos pendentes de desligamento/religamento. Assim como para os pedidos de migração, os pedidos de religamento ou desligamento de servidores são gerados na rodada de simulação anterior pelo Mecanismo de Dinâmica Lenta. Ao processar um pedido, o simulador verifica se o servidor já está no estado desejado, se já se encontra em processo de desligamento ou religamento, ou se possui alguma rede virtual associada (apenas no caso de pedido de desligamento). Caso o servidor se encontre em algum desses estados, o pedido não é retirado da lista de pedidos pendentes e não é processado até que o servidor saia do estado proibido ou até que o Mecanismo de Dinâmica Lenta sobrescreva o pedido. No caso do servidor estar em um estado adequado, o simulador retira o pedido da lista e adiciona o servidor em uma lista de servidores em processo de desligamento ou religamento, configurando o tempo para o fim do processo de religamento ou desligamento e colocando o consumo energético do servidor ao consumo equivalente ao estado ativo da CPU de menor consumo, conforme citado na Seção 5.1. Conforme citado na mesma seção, o servidor não está apto a processar pacotes durante o processo de religamento ou desligamento. É importante lembrar também que o tempo de desligamento do servidor é consideravelmente menor que o de religamento, por exemplo 10 segundos contra 110 segundos, respectivamente, conforme os dados extraídos de [51]. Em seguida, o simulador decrementa o tempo

de cada servidor na lista de servidores em processo de religamento ou desligamento. Os servidores que atingem tempo zero concluem o processo. Caso o servidor esteja em processo de desligamento, este entra em estado de *standby* consumindo baixa quantidade de energia. Caso o servidor esteja em processo de religamento, este continua no estado CPU com mais baixo consumo energético e continua nesse estado e ocioso até que sejam ordenadas alocações de redes virtuais nele pelo Mecanismo de Dinâmica Lenta.

A próxima etapa da simulação da rodada consiste no cálculo dos recursos. A primeira parte dessa etapa consiste em calcular o uso de memória de cada servidor. Para isso, contabiliza-se a memória necessária para efetuar as migrações em andamento e a memória demandada pelos roteadores virtuais associados ao servidor. Caso a quantidade total de memória supere a memória presente no servidor, o simulador gera eventos de terminação de roteadores virtuais até que a quantidade total de memória demandada esteja dentro da capacidade do servidor. Após calcular o uso de memória de cada servidor, passa-se ao cálculo do uso da CPU. Primeiramente calcula-se a quantidade de CPU gasta com sobrecargas geradas por migrações de redes virtuais em andamento nos servidores de origem e destino. Em seguida processam-se os pacotes das redes virtuais associadas aos servidores. O processamento é efetuado verificando os pacotes armazenados no buffer do servidor e a demanda de cada rede virtual para calcular o esforço de CPU necessário para processar os pacotes. Caso o uso da CPU atinja 100%, os pacotes subseqüentes processados na rodada passam a ser acumulados no buffer. Caso o uso do buffer alcance a capacidade máxima de armazenamento, os pacotes subseqüentes são descartados e são gerados eventos de perda de pacotes nas redes virtuais das quais os pacotes perdidos provêm. Após o processamento dos pacotes das redes virtuais, o simulador calcula o gasto energético de cada servidor. Os servidores que processaram pacotes consomem energia de acordo com o estado da CPU em que se encontravam. Os servidores em *standby* consomem uma baixa quantidade de energia específica para servidores nesse estado. Os servidores em processo de desligamento ou religamento consomem energia equivalente ao estado ativo de menor consumo energético da CPU. Por último, os servidores ociosos consomem energia conforme o estado ocioso de CPU em que se encontra.

A última etapa do cálculo de uma rodada de simulação consiste na execução dos mecanismos de alocação de recursos, caso haja algum mecanismo de alocação em uso. Primeiramente é executado o Mecanismo de Dinâmica Rápida, se houver algum em uso na simulação. A primeira parte da execução do Mecanismo de Dinâmica Rápida é a verificação de quantas rodadas se passaram desde o último planejamento efetuado para o servidor. Caso o número de rodadas tenha atingido o valor definido na sua configuração, o mecanismo efetua um novo planejamento para controlar o servidor

nas rodadas subseqüentes. Em seguida o mecanismo age no servidor conforme o planejado para a rodada corrente, configurando a CPU para o estado definido no plano. Após a execução do Mecanismo de Dinâmica Rápida em cada servidor, é executado o Mecanismo de Dinâmica Lenta, caso esteja em uso na simulação. Assim como para o Mecanismo de Dinâmica Rápida, verifica-se quantas rodadas passaram desde o último planejamento e, um novo plano para controlar o cluster nas rodadas subseqüentes é feito caso o número de rodadas passadas seja igual ao valor definido na configuração. Em seguida o Mecanismo de Dinâmica Lenta atua no cluster conforme definido no plano em vigência, migrando as redes virtuais e religando ou desligando os servidores necessários.

Após concluir a descrição detalhada do modelo da dinâmica do roteador de cluster e do simulador desenvolvido para executá-lo, passa-se para a definição do mecanismo de dinâmica rápida alternativo, desenvolvido para melhor avaliar o ECO. O próximo capítulo explora o uso de otimização para a construção desse mecanismo e detalha a lógica da formação de sua função custo.

Capítulo 6

O Mecanismo de Dinâmica Rápida Alternativo

Neste capítulo apresenta-se um Mecanismo de Dinâmica Rápida alternativo, baseado na técnica *Simulated Annealing* de otimização não-linear, que foi desenvolvido para melhor avaliar o desempenho do ECO. A escolha por utilizar otimização para o desenvolvimento do Mecanismo de Dinâmica Rápida alternativo advém do fato de ter sido observada a ampla utilização de mecanismos baseados em otimização para a solução de problemas de alocação de recursos, conforme pode-se verificar nos trabalhos relacionados apresentados no Capítulo 2.

6.1 Mecanismo de Dinâmica Rápida alternativo

Nesta seção apresenta-se um Mecanismo de Dinâmica Rápida (MDR) alternativo.

Utilizando-se uma configuração estática do estado da CPU, pode-se obter os resultados extremos do desempenho de encaminhamento ou de economia de energia, mas não os dois ao mesmo tempo. O resultado de melhor encaminhamento e pior consumo energético possíveis pode ser obtido colocando a CPU no estado de maior gasto energético e maior capacidade de encaminhamento. Colocando a CPU no estado de menor gasto energético obtém-se o melhor resultado quanto a consumo energético, porém o pior resultado quanto ao encaminhamento de pacotes.

Para enriquecer a avaliação de desempenho do MDR-ECO, opta-se pelo desenvolvimento de um MDR alternativo. Para isso, opta-se pelo desenvolvimento de uma abordagem baseada em otimização. Primeiramente, é necessário definir uma função de custo para modelar o problema da escolha de estados da CPU e economia de energia. Dessa forma, a função de custo deve apresentar um termo que modele a necessidade de prover poder de processamento suficiente para atender a demanda de encaminhamento. Visto que esse termo tende a maximizar a capacidade de encami-

nhamento da CPU e, por conseqüência, maximizar o gasto energético, é necessário também definir um termo para considerar o gasto energético. Com esses fatores em mente, define-se uma função de custo $\Phi(\mathbf{t})$, para ser minimizada. $\Phi(\mathbf{t})$, exibida na Equação 6.1, é a soma de três termos, cada um ponderado por um peso próprio (α, β, e γ), e depende de $\mathbf{t}$, o vetor de intervalos de tempo, em segundos, gasto em cada estado i da CPU, com $i \in \{1 \dots N\}$. N representa o número total de estados da CPU.

$$\Phi(\mathbf{t}) = \alpha.E(\mathbf{t}) + \beta.S(\mathbf{t}) + \gamma.R_{tempo}(\mathbf{t}) \tag{6.1}$$

$$E(\mathbf{t}) = \sum_{i=1}^{N} \Big(P_i^{ocioso}.t_i^{ocioso} + P_i^{ativo}.t_i^{ativo} \Big).E_{norm} \tag{6.2}$$

$$S(\mathbf{t}) = \left(\Big(\sum_{i=1}^{N} C_i.t_i^{ativo} \Big) - D \right)^{-1}.S_{norm} \tag{6.3}$$

$$R_{time}(\mathbf{t}) = \left(T - \sum_{i=1}^{N} (t_i^{ocioso} + t_i^{ativo}) \right)^{2}.R_{norm} \tag{6.4}$$

O primeiro termo, $E(\mathbf{t})$, é o termo responsável por modelar o consumo energético, em Joules, do servidor, e é apresentado na Equação 6.2. $E(\mathbf{t})$ contabiliza a energia gasta, para todos os N estados da CPU, pelo servidor enquanto ocioso, $P_i^{ocioso}.t_i^{ocioso}$, e enquanto encaminhando pacotes (ativo), $P_i^{ativo}.t_i^{ativo}$, onde P_i^{ocioso} e P_i^{ativo} são as potências, em Watts, do servidor quando ocioso e quanto ativo com a CPU no estado i, respectivamente. Como pode-se observar na Equação 6.2, $E(\mathbf{t})$ assume valores maiores quando aumenta-se o consumo energético do servidor e valores menores quando diminui-se o consumo energético do servidor.

O segundo termo, $S(\mathbf{t})$, é responsável por modelar a necessidade de prover a capacidade de processamento para atender a demanda de encaminhamento de pacotes. $S(\mathbf{t})$ modela a capacidade ociosa de processamento do servidor, e é definido como o inverso da diferença entre a soma da capacidade individual de encaminhamento de cada estado da CPU, $C_i.t_i^{ativo}$, em pacotes, e o total de demanda de tráfego das redes virtuais no servidor, D, em pacotes, como mostrado na Equação 6.3. A capacidade total de encaminhamento depende de quanto tempo foi gasto em um estado da CPU com o servidor ativo, t_i^{ativo}, e a capacidade de encaminhamento da CPU naquele estado, C_i, em pacotes por segundo. Conforme mostrado na Equação 6.3, $S(\mathbf{t})$ assume valores menores quando há maior capacidade de processamento ociosa e assume valores maiores quando diminui-se a capacidade de processamento do servidor.

O terceiro termo de $\Phi(\mathbf{t})$, $R_{tempo}(\mathbf{t})$, garante a consistência dos valores de tempo escolhidos que devem ser gastos em cada estado i da CPU, de forma que a soma

dos intervalos de tempo que a CPU gasta em cada estado corresponda ao tamanho do intervalo de tempo T. Conforme exibido na Equação 6.4 minimiza-se este termo ao utilizar intervalos de tempo que somem uma quantidade de tempo equivalente ao tamanho do intervalo de tempo sendo planejado, T, e maximiza-se esse termo quanto mais a soma dos intervalos de tempo escolhida divirja do valor de T.

As constantes em cada um dos termos de $\Phi(\mathbf{t})$, E_{norm}, S_{norm} e R_{norm}, são responsáveis por cancelar as unidades de cada termo, fazendo com que o valor da função de custo seja adimensional, e servem para normalizar cada termo da otimização, visto que cada termo possui originalmente valores com ordens de grandeza muito dispares, indo desde as dezenas de Watts de consumo energético do servidor até os milhões de pacotes que um servidor pode encaminhar.

Os pesos que multiplicam cada termo de $\Phi(\mathbf{t})$ servem para configurar a otimização. Aumentar α faz a otimização priorizar a economia de energia, enquanto aumentar β estimula o aumento de capacidade ociosa, melhorando o desempenho e aumentando o gasto energético. Finalmente, γ determina o quão restritiva é a otimização ao escolher os intervalos de tempo que compõe T.

6.1.1 Escolha do Algoritmo de Otimização

A função de custo definida é não-linear e, dessa forma, foram consideradas inicialmente as técnicas de força bruta e *Simulated Annealing* (SA) dentre as possíveis técnicas para resolvê-la. A técnica de força bruta provê os melhores resultados quando exaustivamente executada, mas apresenta tempo de convergência extremamente alto. A técnica de *Simulated annealing*, por outro lado, apresenta bons resultados com tempo de convergência bem menor. Para ilustrar, a Tabela 6.1 apresenta o tempo de execução e os resultados de desempenho obtidos tanto para a técnica de força bruta quanto para a técnica *Simulated Annealing* em simulações efetuadas. As simulações utilizam um servidor com oito redes virtuais utilizando perfis de tráfego baseados nos dados da CAIDA. Visto que a técnica de força bruta testa todas as possíveis configurações de CPU dentro de um intervalo de tempo T, o tempo de sua execução é proporcional a N^T, rapidamente levando a tempos de convergência inviáveis ao aumentar N ou T. Por esta razão, executam-se as simulações com o algoritmo de força bruta explorando três estados da CPU: o com maior capacidade de encaminhamento, o com melhor relação de energia gasta por pacote encaminhado, e o com menor consumo energético, os mesmos usados pelo MDR-ECO. Por esta mesma limitação de convergência, utiliza-se T com o valor de 10 passos de simulação. As simulações usando *Simulated Annealing* são executadas usando os mesmos três estados e, também, utilizando todos os 35 estados da CPU considerada. É interessante notar que o número de estados da CPU utilizados não aumenta notavelmente

Algoritmo de Otimização	**Número de Estados da CPU**	**Potência Média Consumida (W)**	**Uso Médio do Buffer (Pacotes)**	**Tempo de Execução da Simulação (Segundos)**
Força Bruta	3	325.4	12220.5	182
SA	3	329.1	9712.5	4
SA	35	306.8	16482.7	3
Sem MDR	1	356.0	0.0	3

Tabela 6.1: Desempenho das técnicas *Simulated Annealing* e força bruta.

o tempo de execução do algoritmo de *Simulated Annealing*. O aumento do número de estados da CPU possibilita maior economia de energia mas requer maior uso do buffer. O tempo de execução do algoritmo de *Simulated Annealing* depende dos parâmetros avaliados na Subseção 7.2. Também testa-se o comportamento do sistema sem uso de um Mecanismo de Dinâmica Rápida (MDR), com a CPU sempre no estado de maior poder de processamento e maior gasto energético. Visto que o melhor resultado para o algoritmo de *Simulated Annealing* é obtido usando todos os estados da CPU, sem acarretar em um aumento notável do tempo de execução, esta configuração é tomada como padrão para os próximos testes.

6.1.2 Controle dos Estados da CPU pelo MDR-SA

Assim como no caso do MDR-ECO, o algoritmo de planejamento decide quais estados da CPU serão utilizados e por quanto tempo serão utilizados dentro de um intervalo de tempo. Dessa forma, ainda resta definir como o tempo total a ser gasto em cada estado da CPU é distribuído dentro do intervalo de tempo planejado.

Ao controlar a CPU por um intervalo de tempo, o MDR-SA passa continuamente o tempo previsto em cada estado da CPU. O tempo a ser passado em cada estado condiz com os valores definidos no vetor $\mathbf{t}$, que contém o tempo total planejado pelo otimização para ser gasto em cada estado da CPU. Opta-se por não particionar o tempo gasto em cada estado da CPU, diferentemente do que é feito no MDR-ECO, devido ao grande número de estados da CPU envolvidos no plano de controle da CPU estabelecido pelo MDR-SA, 35 no caso da CPU considerada neste trabalho, ou 70 se for levado em consideração que cada estado da CPU possui duas entradas no vetor de tempo: uma para o tempo que passa no estado encaminhando pacotes e outra para o tempo que passa no estado ociosa, sem encaminhar pacotes. Como o intervalo de tempo onde o MDR atua é curto em relação ao número de estados utilizado pelo MDR-SA, na ordem de dezenas de rodadas de simulação nos testes feitos neste trabalho, a troca de estados da CPU ocorre naturalmente com alta freqüência. Os estados planejados para o controle da CPU são utilizados em ordem

decrescente de capacidade de encaminhamento.

Finalizada a descrição do MDR alternativo, pode-se passar para a avaliação do mecanismo proposto. No próximo capítulo são apresentados diversos testes para avaliar o desempenho do mecanismo proposto e seu comportamento frente a modificações nos valores de seus parâmetros.

Capítulo 7

Avaliação

Neste capítulo é apresentada a avaliação do mecanismo do ECO-ALOC proposto neste trabalho. Inicialmente, são apresentados os detalhes da configuração utilizada no simulador desenvolvido. Depois, apresentam-se testes preliminares com o intuito de compreender a sensibilidade dos mecanismos testados aos seus parâmetros e escolher os melhores parâmetros de configuração dos mecanismos para os testes subseqüentes. Após os testes preliminares, apresentam-se testes efetuados com os mecanismos utilizando dados baseados em tráfego real com 24 horas de duração, a fim de mostrar a eficiência do mecanismo proposto. Em seguida, é efetuado um estudo mais detalhado do comportamento do Mecanismo de Dinâmica Rápida proposto, analisando sua dinâmica em uma curta escala de tempo e comparando seu comportamento com o verificado no Mecanismo de Dinâmica Rápida alternativo.

7.1 Configuração do Simulador

Essa seção detalha os parâmetros utilizados no simulador de roteador de software em cluster desenvolvido para avaliar as técnicas propostas.

O simulador cria um cluster com o número desejado de servidores. Cada servidor possui uma CPU com um conjunto de estados com capacidade de encaminhamento e consumo energético distintos para quando a CPU está ativa ou ociosa. Os dados de capacidade de encaminhamento de pacotes e consumo do servidor de acordo com o estado de operação da CPU foram retirados de [45] e [48], e podem ser observados nas Figuras 3.2 e 3.1 respectivamente. Cada servidor também possui um buffer com capacidade de armazenamento de 125.000 pacotes, valor típico de buffer utilizado em enlaces do *backbone* de 2,5Gb/s considerando um tamanho médio de pacote de 500 bytes [68].

Para cada rede virtual, utiliza-se um perfil de demanda de encaminhamento de pacotes e quantidade de memória RAM necessária. Os perfis de carga são compostos por classes de tráfego HTTP, HTTPS, FTP e e-mail extraídas de dados de rotea-

dores do *backbone* da CAIDA [69] e da rede Ipê da RNP [1], conforme explicado na Seção 5.2.1.

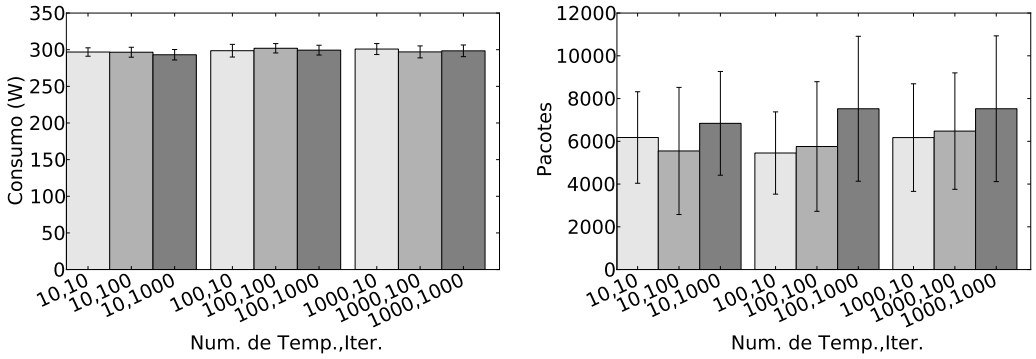
O simulador cria as redes virtuais escolhendo um perfil de tráfego dentre os quatro perfis disponíveis usando como critério o algoritmo *round-robin*. A distribuição inicial das redes virtuais entre os servidores do cluster também segue o mesmo critério. Cada passo de simulação corresponde a 10 ms de tempo e consiste em calcular o novo estado (CPU, memória, buffer e consumo energético) de cada servidor do cluster e executar os mecanismos de alocação de recursos. O consumo de CPU e memória é calculado de acordo com a demanda das redes virtuais e com as sobrecargas geradas pelas migrações de redes virtuais. As migrações de redes virtuais duram 3 s ao todo, geram sobrecarga na CPU dos servidores de origem e destino equivalente a 8% da capacidade da CPU, e alocam a quantidade de memória RAM requerida pelo roteador virtual no servidor de destino. Todos os parâmetros e sobrecargas da migração foram baseados em [53]. Para atualizar o estado dos buffers de cada servidor, considera-se que os pacotes que não puderam ser processados no passo de simulação ficam armazenados no buffer, e, caso o mesmo esteja cheio, os pacotes excedentes são descartados. O gasto energético de servidores ligados é calculado de acordo com o estado da CPU. A execução dos mecanismos de alocação de recursos envolve o planejamento e controle dos estados da CPU (dinâmica rápida); e as ações de longo prazo (dinâmica lenta), como a migração de roteadores virtuais, o desligamento ou religamento de servidores. Considera-se que um servidor demora 110 s para ser religado e 10 s para ser desligado, intervalos de tempo durante os quais o servidor não processa nenhum pacote e consome energia equivalente ao estado de CPU ativo de mais baixo consumo [51]. Após o desligamento, os servidores ficam em estado de *standby* consumindo 15 W.

7.2 Sintonia de Parâmetros do MDR-SA

Esta seção apresenta maiores detalhes do funcionamento do algoritmo *Simulated annealing* (SA) de otimização não-linear e apresenta testes efetuados para escolher o valor dos parâmetros utilizados na execução da otimização. O SA é uma técnica de otimização que perturba as variáveis de entrada, definidas na função de custo do MDR-SA como o vetor N-dimensional $\mathbf{t}$ de intervalos de tempo gastos em cada estado da CPU, com o objetivo de encontrar o valor mínimo para a função de custo definida, $\Phi(\mathbf{t})$. O SA é um algoritmo iterativo que inicia com uma configuração arbitrária das variáveis de entrada e, a cada iteração, adiciona uma perturbação arbitrária à configuração anterior, avaliando se o efeito da perturbação reduz o erro de convergência. O algoritmo SA possui dois parâmetros. O primeiro, denominado temperatura, controla a probabilidade de aceitar uma nova configuração para as

variáveis de entrada quando a configuração testada acarreta em aumento do erro de convergência. A temperatura é alta no início da execução, para evitar convergir para um mínimo local, e decai ao longo da execução do algoritmo. O segundo parâmetro é o número de iterações, que define o número de configurações testadas em cada temperatura.

Para testar o valor de parâmetros a serem adotados, simulam-se cinco servidores com duas redes virtuais em cada um. Cada simulação é efetuada durante 400 passos, com o MDR-SA sendo executado a cada 20 passos, e utilizando o tráfego das redes virtuais baseado nos dados da CAIDA. Tanto o número de temperaturas quanto o número de iterações assumem os valores 10, 100 e 1000, levando a 9 configurações diferentes. Para cada par (temperatura, iteração), as simulações são executadas vinte vezes, número de experimentos suficiente para obter desvio padrão inferior a 10% no valor médio de potência consumida. Não há perdas de pacotes nos experimentos efetuados. A potência média gasta por servidor e a quantidade média de pacotes nos buffers de cada servidor são mostradas na Figura 7.1.



(a) Potência média consumida.

(b) Uso médio de buffer.

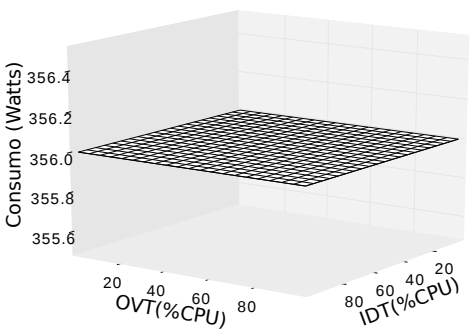
Figura 7.1: Sintonia de parâmetros do MDR-SA.

Os resultados mostram que o MDR-SA foi capaz de economizar quase 20% de energia quando comparado ao consumo de 356 W dos servidores sem o uso de mecanismos de alocação de recursos. Todas as configurações de (temperatura, iteração) praticamente não usaram os buffers, utilizando em média em torno de 5% da capacidade máxima de 125.000 pacotes. Como resultados similares de desempenho foram obtidos para todas as configurações testadas, a configuração de (temperatura, iteração) usando os valores (10, 10) é utilizada para os próximos testes, pois esta configuração é a que exige menor esforço computacional.

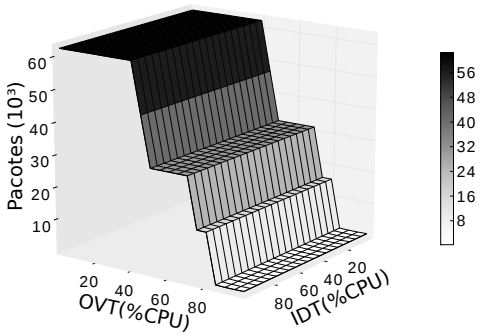
7.3 Sintonia de Parâmetros do MDL-ALOC

Esta seção apresenta os testes efetuados para analisar o impacto da configuração dos parâmetros do ALOC em seu desempenho. Para isso, são efetuadas simulações variando o número de servidores no cluster entre 4, 8 e 16 e com 40 redes virtuais no cluster. As simulações são efetuadas com 150.000 passos de duração e o ALOC é configurado com os valores de OVT e IDT de 0% até 100% com passos de 5% em 5%. O MDL-ALOC é executado a cada 30.000 passos e os perfis de tráfego das redes virtuais são baseados nos dados da CAIDA. O número de servidores no cluster simula três situações diferentes. O cluster no cenário com 4 servidores está sobrecarregado. O cenário com 8 servidores simula um cluster com distribuição bastante heterogênea de carga, alguns servidores estão com alta carga devido a redes virtuais carregando tráfego HTTP e outros servidores apresentam baixa carga proveniente de redes virtuais com tráfego de e-mail, FTP e HTTPS. A configuração com 16 servidores apresenta um cenário onde há grande capacidade ociosa de processamento.

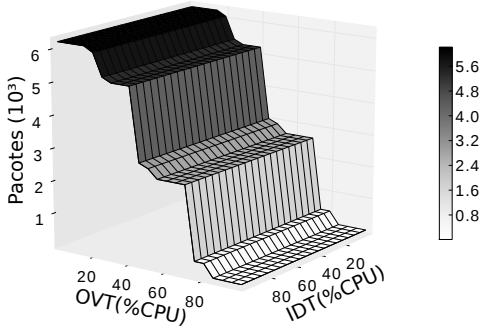
Os resultados para os cenários com 4, 8 e 16 servidores são exibidos nas Figuras 7.2, 7.3 e 7.4 respectivamente. Em geral, aumentar os valores de OVT e IDT diminui o gasto energético conforme esperado.



(a) Potência média consumida.



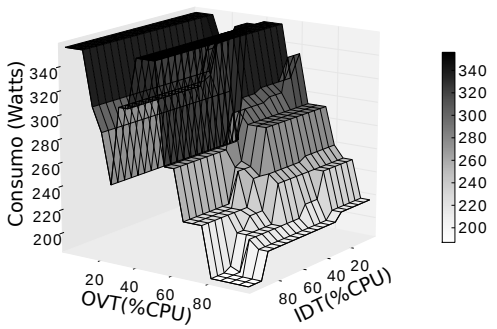
(b) Uso médio de buffer.



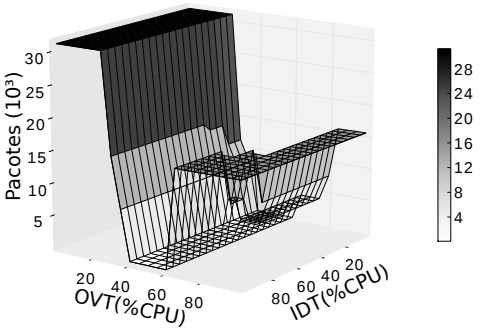
(c) Perda média de pacotes.

Figura 7.2: Sintonia de parâmetros do MDL com 4 servidores no cluster.

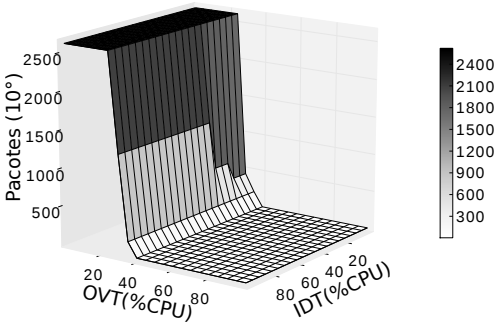
No cenário exibido na Figura 7.2 com 4 servidores, não há capacidade ociosa de processamento no cluster suficiente para permitir o desligamento de nenhum servidor. Dessa forma, o MDL-ALOC não pode economizar energia e o valor do parâmetro IDT não influencia nesse cenário, como pode ser observado na Figura 7.2(a). Entretanto, conforme o valor do limiar de sobrecarga (OVT) aumenta, o MDL-ALOC ganha maior margem de manobra para melhorar a alocação das redes virtuais entre os servidores, continuamente diminuindo o uso do buffer e a perda de pacotes, como pode ser observado nas Figuras 7.2(b) e 7.2(c). Os melhores resultados nesse cenário de sobrecarga foram obtidos para as configurações com 100% de OVT, visto que esse valor de OVT permite ao ALOC maior liberdade no gerenciamento da distribuição das redes virtuais.



(a) Potência média consumida.



(b) Uso médio de buffer.

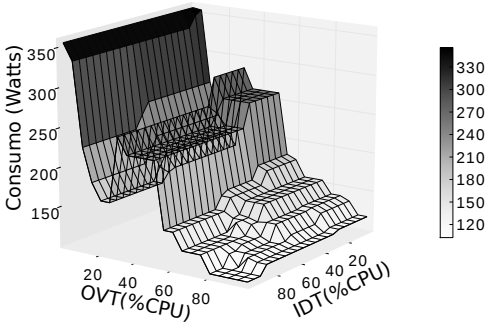


(c) Perda média de pacotes.

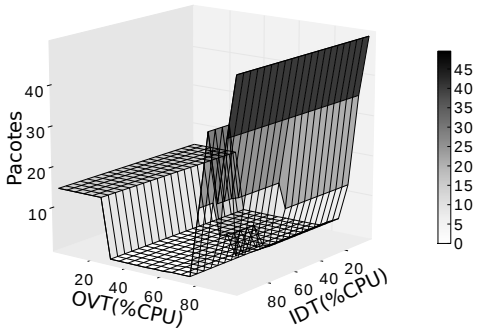
Figura 7.3: Sintonia de parâmetros do MDL com 8 servidores no cluster.

No cenário com 8 servidores, exibido na Figura 7.3, pode-se observar que o parâmetro IDT passa a influenciar no comportamento do ALOC, visto que nesse cenário existe capacidade de processamento ociosa suficiente para efetuar o desligamento de servidores. Como pode-se observar na Figura 7.3(a), valores de OVT acima de 40% possibilitam alta consolidação das redes virtuais, gerando maior economia de energia. O parâmetro IDT é altamente dependente do valor do OVT,

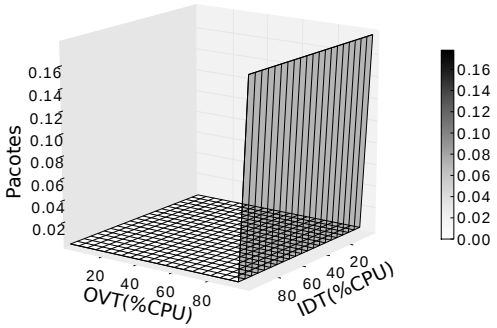
visto que o OVT controla o nível de consolidação das redes virtuais, e, dessa forma, o número de servidores ociosos. Altos valores de IDT possibilitam mais freqüentemente o desligamento de servidores ociosos, provendo maior economia de energia, mas reduzindo o poder de reação do cluster a aumentos no tráfego.



(a) Potência média consumida.



(b) Uso médio de buffer.



(c) Perda média de pacotes.

Figura 7.4: Sintonia de parâmetros do MDL com 16 servidores no cluster.

Quanto ao uso médio do buffer, mostrado na Figura 7.3(b), valores de OVT acima de 20% permitem melhor realocação das redes virtuais de servidores sobrecarregados, diminuindo assim o uso do buffer. Como previamente explicitado, a distribuição inicial das redes virtuais é feita utilizando *round-robin*. Visto que os números de servidores nos testes efetuados são múltiplos do número de perfis de tráfego utilizados (quatro: HTTP, HTTPS, FTP e e-mail), todos os servidores recebem inicialmente apenas um tipo de tráfego. Neste cenário, dois servidores recebem somente tráfego HTTP, outros dois somente HTTPS, e assim por diante com FTP e e-mail. Posto que o volume médio de tráfego HTTP é mais de 20 vezes maior que os tráfegos HTTPS, FTP e de e-mail, quando o valor do OVT é inferior a 30%, o MDL-ALOC não consegue migrar redes virtuais com HTTP, explicando a queda inicial de gasto energético para valores de OVT abaixo de 25%. Com OVT entre 30% e 40%, o MDL-ALOC passa a migrar redes virtuais HTTP, conseguindo retirar

a sobrecarga dos servidores nos quais estavam inicialmente concentradas. Apesar do aumento no gasto energético, retirar a sobrecarga desses servidores gera grande redução no uso do buffer. O último comportamento notável no uso do buffer ocorre para os valores de OVT e IDT acima de 60% e 50%, respectivamente. Acima desses valores o uso do buffer aumenta por causa da baixa capacidade ociosa nos servidores ligados, insuficiente para lidar com eventuais picos na demanda.

Quanto à perda de pacotes, exibida na Figura 7.3(c), ela só ocorre nos servidores que estão inicialmente sobrecarregados com redes HTTP, até que o menor valor de OVT que permite migrar redes HTTP é atingido. Após esse valor ser atingido, o MDL-ALOC é capaz de eliminar a sobrecarga nesses servidores. O MDL-ALOC consegue economizar 48% de energia quando utilizando OVT e IDT iguais a 100% nesse cenário com 8 servidores.

No cenário com 16 servidores, exibido na Figura 7.4, as tendências de desempenho do ALOC são similares às observadas no cenário com 8 servidores. As principais diferenças estão no fato de não haver perda de pacotes na configuração inicial e que o MDL-ALOC consegue migrar redes virtuais HTTP para valores mais baixos de OVT devido à maior disponibilidade de recursos livres, o que desloca a anomalia na diminuição do gasto energético para valores próximos a 20% de OVT. Nesse cenário o MDL-ALOC conseguiu economizar 71% de energia para os valores de OVT e IDT iguais a 100%.

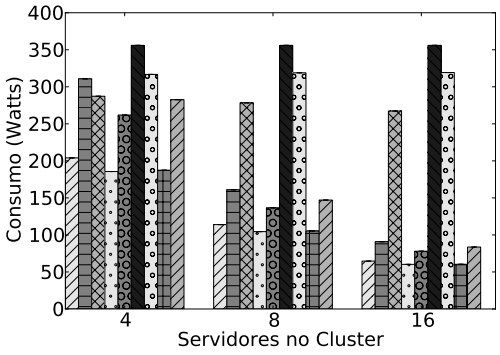
Observando os resultados apresentados nos três cenários, decide-se utilizar duas configurações distintas para os próximos testes com o MDL-ALOC. A primeira configuração, utilizando os valores de (100% OVT , 100% IDT), é escolhida por ter se apresentado como a mais agressiva na economia de energia. A segunda configuração, utilizando os valores de (60%OVT, 65% IDT), é escolhida por apresentar boa economia de energia com baixo uso de buffer.

7.4 Avaliação do ECO-ALOC

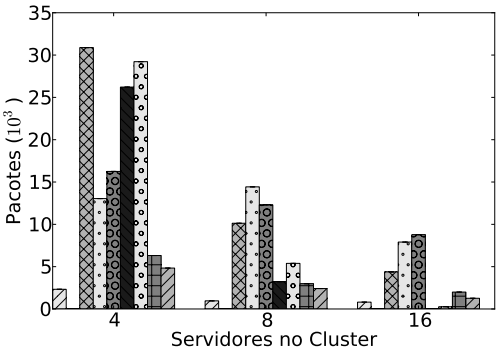
Nesta seção avaliam-se os mecanismos propostos MDR-ECO e MDL-ALOC, além do mecanismo alternativo MDR-SA tanto individualmente quanto combinados. O número de servidores no cluster simula três situações diferentes, as mesmas simuladas na Seção 7.3. Para testar um cenário com o cluster sobrecarregado utiliza-se apenas 4 servidores. Para um cenário com servidores inicialmente sobrecarregados com redes virtuais HTTP e servidores com baixa carga de redes HTTPS, FTP e de e-mail, utiliza-se um cluster com 8 servidores. No cenário com 16 servidores testam-se os mecanismos em um cluster com abundância de recursos ociosos. As simulações utilizam 40 redes virtuais e duram 8.600.000 rodadas, equivalente a 24 horas. Para os perfis de tráfego das redes virtuais efetuam-se testes usando tanto os

dados provenientes da CAIDA quanto da RNP. Os dados da RNP utilizados vêm de dois enlaces de 10 Gbps: do enlace entre o Rio de Janeiro e São Paulo e do enlace entre Rio de Janeiro e Belo Horizonte.

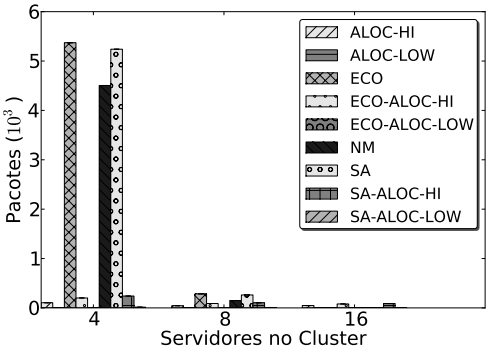
Os resultados dos testes estão nas Figuras 7.5, 7.6 e 7.7 . Nos gráficos apresentados, a sigla NM referencia os testes efetuados sem nenhum mecanismo de alocação de recursos e com a CPU de todos os servidores configurada no estado de maior capacidade de processamento e gasto energético. Por simplicidade as siglas "MDR" e "MDL" são omitidas na identificação dos mecanismos dentro dos gráficos. As identificações com "HI" e "LOW" se referem aos testes efetuados com o MDL-ALOC utilizando as configurações de (100% OVT, 100% IDT) e (60% OVT, 65% IDT), respectivamente.



(a) Potência média consumida - CAIDA.



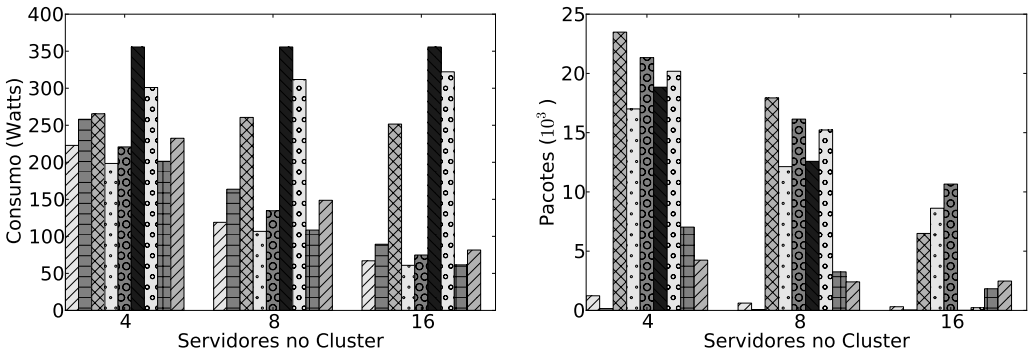
(b) Uso médio de buffer - CAIDA.



(c) Perda média de pacotes - CAIDA.

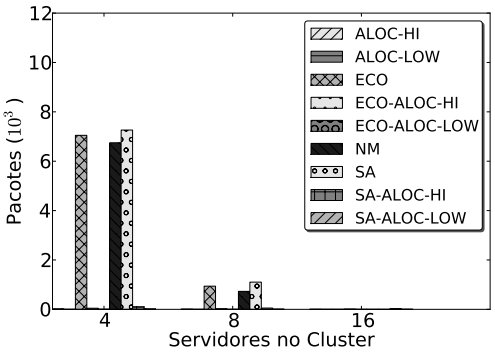
Figura 7.5: Avaliação dos mecanismos usando tráfego baseado nos dados da CAIDA.

Considerando as simulações feitas com os dados da CAIDA, exibidas na Figura 7.5, os resultados com 4 servidores no cluster mostram que o MDR-ECO consegue economizar mais energia do que o MDR-SA, mecanismo baseado na ferramenta de otimização, quando comparados com o uso de nenhum mecanismo, 19% e 11% respectivamente. O MDR-ECO utiliza 5% mais o buffer do que o MDR-SA e perde 2% a mais de pacotes que o MDR-SA nesse cenário de pior caso com o cluster so-



(a) Potência média consumida.

(b) Uso médio de buffer.

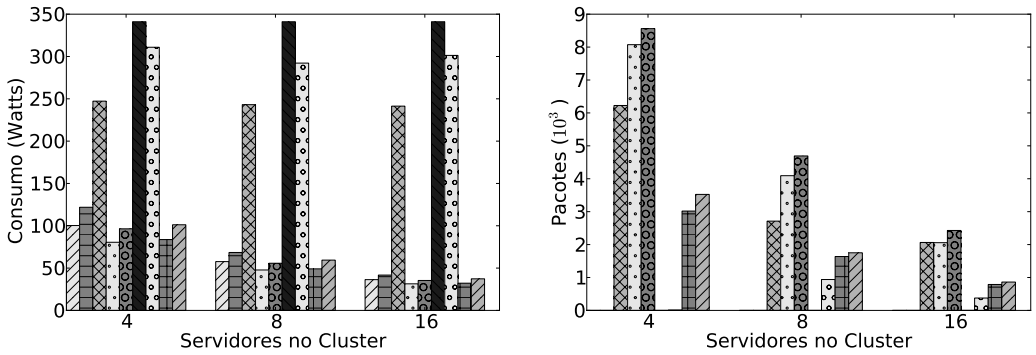


(c) Perda média de pacotes.

Figura 7.6: Avaliação dos mecanismos usando tráfego baseado nos dados da RNP RJ-SP.

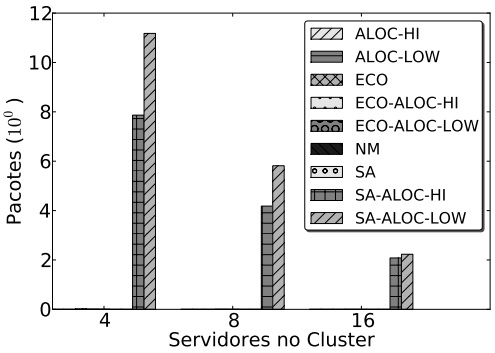
brecarregado. Comparando com o uso de nenhum mecanismo, tanto o ECO quanto o SA perdem em torno de 19% a mais de pacotes. Isso se deve ao fato de que tanto o ECO quanto o SA basearem suas escolhas em uma demanda estimada, que, nesse cenário de sobrecarga, gerou mais perdas do que usando nenhum mecanismo e deixando a CPU na capacidade máxima de encaminhamento o tempo todo. O uso individual do MDL-ALOC foi capaz de redistribuir a carga dentro do cluster, diminuindo o consumo energético para ambas as configurações LOW e HI. Além disso, o uso do MDL-ALOC conseguiu reduzir drasticamente o uso do buffer e praticamente eliminar as perdas de pacotes. Combinar os MDRs com o MDL-ALOC continua provendo quase nenhuma perda de pacote, mas consegue aumentar a economia de energia em troca de maior uso do buffer. Finalmente, o ECO-ALOC usando a configuração agressiva de (100% OVT, 100% IDT) praticamente elimina a perda de pacotes, economiza 48% de energia e usa 50% menos o buffer em comparação a não usar nenhum mecanismo.

Na configuração com 8 servidores, as duas configurações do MDL-ALOC continuam redistribuindo a carga de maneira apropriada. A configuração HI de (OVT, IDT) economizou 30% a mais de energia que a configuração LOW, mas a confi-



(a) Potência média consumida.

(b) Uso médio de buffer.



(c) Perda média de pacotes.

Figura 7.7: Avaliação dos mecanismos usando tráfego baseado nos dados da RNP RJ-MG.

guração HI utiliza freqüentemente o buffer enquanto a configuração LOW quase não o utiliza. As duas técnicas de MDR foram capazes de reduzir o consumo energético em relação a não usar nenhum mecanismo, mas o custo foi aumento na utilização do buffer e um pequeno aumento na perda de pacotes nos servidores sobrecarregados com tráfego HTTP. O ECO-ALOC conseguiu reduzir ainda mais o consumo de energia em relação ao ALOC sozinho ao custo de usar o buffer mais freqüentemente. O ECO-ALOC também economizou mais energia do que a combinação SA-ALOC, utilizando mais o buffer para fazê-lo.

Finalmente, com 16 servidores o cluster possui abundante quantidade de recursos e não há servidores sobrecarregados na configuração inicial. O MDL-ALOC consegue consolidar apropriadamente as redes virtuais em um número menor de servidores, economizando entre 42% e 46% de energia nas configurações individuais e quando combinado ao ECO e ao SA em relação às mesmas combinações de mecanismos no cenário com 8 servidores no cluster. Através da alta consolidação das redes virtuais e da utilização do buffer, ECO-ALOC-HI conseguiu economizar 83% de energia e o ECO-ALOC-LOW 78% em relação a não usar nenhum mecanismo.

Considerando as simulações baseadas nos dados do *backbone* da Rede Nacional

de Ensino e Pesquisa (RNP) para os enlaces entre Rio de Janeiro e São Paulo, exibidas na Figura 7.6, e entre Rio de Janeiro e Belo Horizonte, exibidas na Figura 7.7, é possível observar que o MDL-ALOC continua eficiente na reorganização da carga dentro do cluster, tanto para eliminar a sobrecarga dos servidores quanto para economizar energia. A troca básica entre economia de energia e uso de buffer ainda pode ser observada ao modificar a configuração de (OVT, IDT) ou ao se combinar o ALOC com os MDRs. No cenário com 16 servidores, ECO-ALOC-HI é capaz de economizar 83% de energia para o enlace com São Paulo e 93% para o enlace com Belo Horizonte, que possuir menor taxa de utilização que o enlace com São Paulo.

Os testes desta Seção mostram que o mecanismo proposto ECO-ALOC é capaz de lidar bem com a reorganização da carga dentro do cluster em situações de sobrecarga ao mesmo tempo em que consegue obter boas taxas de economia de energia quando o tráfego está baixo. Os parâmetros de OVT e IDT produzem uma configuração com maior ou menor efetividade dependendo da situação do cluster e, por isso, merecem uma investigação sobre formas de controlá-los dinamicamente.

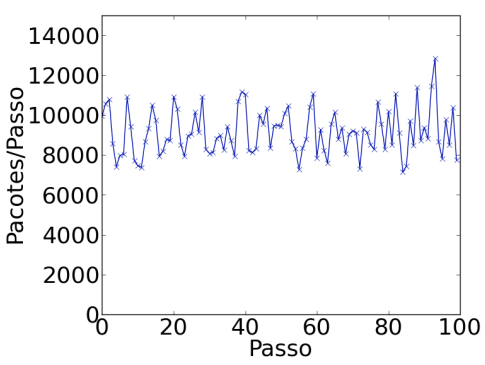
7.5 Análise dos Mecanismos de Dinâmica Rápida em Curta Escala de Tempo

Essa seção tem como objetivo analisar o comportamento dos Mecanismos de Dinâmica Rápida em curta escala de tempo para verificar como os Mecanismos de Dinâmica Rápida comportam-se dentro de seus intervalos de execução quanto ao uso do buffer, ao encaminhamento de pacotes e ao consumo energético. Para isso, os Mecanismos de Dinâmica Rápida são executados em curtas simulações de cem rodadas utilizando apenas um servidor no cluster e com apenas um roteador virtual com tráfego baseado nos dados da CAIDA. A curta duração das simulações e o fato de haver apenas um servidor nas simulações dispensa a execução do Mecanismo de Dinâmica Lenta. O uso de apenas um roteador virtual coloca o servidor em um cenário com baixa carga. Para avaliar o comportamento dos Mecanismos de Dinâmica Rápida em situações com cargas mais acentuadas, também são efetuadas simulações em que os dados de taxa de pacotes demandados por passo de simulação são multiplicados por um fator, em dois outros cenários diferentes: em um o tráfego é escalado por um fator de duas vezes, tornando-o mais intenso mas ainda abaixo da capacidade do servidor, no outro o tráfego é escalado por um fator de quatro vezes, de forma a sobrecarregar o servidor durante os picos. Em uma segunda etapa de análises, o número de rodadas utilizado para efetuar o planejamento do Mecanismo de Dinâmica Rápida é variado para observar as mudanças no comportamento do Mecanismo de Dinâmica Rápida e o impacto no consumo energético e no encami-

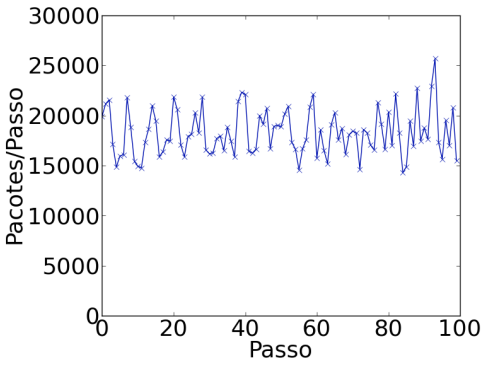
nhamento do servidor.

7.5.1 Simulações Variando a Intensidade do Tráfego

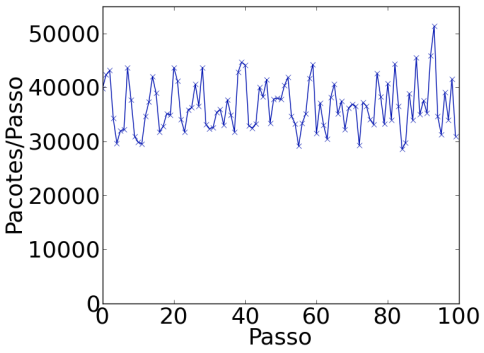
Nesta subseção são apresentadas as simulações utilizando uma rede virtual com perfil de tráfego HTTP retirado dos dados de tráfego da CAIDA e com os perfis de tráfego escalados por um fator de duas vezes e por um fator de quatro vezes. O perfil de tráfego original demanda uma pequena parte da capacidade de processamento do servidor, fazendo com que este cenário sirva para analisar o comportamento dos Mecanismos de Dinâmica rápida em um cenário com baixa carga. O cenário com o tráfego escalado por um fator de duas vezes apresenta uma carga intermediária para o servidor, enquanto o cenário com tráfego escalado por um fator de quatro vezes apresenta carga intensa para o servidor. A demanda de tráfego da rede virtual durante os cem passos de simulação para o tráfego normal e para os tráfegos escalados é mostrada na Figura 7.8.



(a) Tráfego normal.



(b) Tráfego escalado 2x.

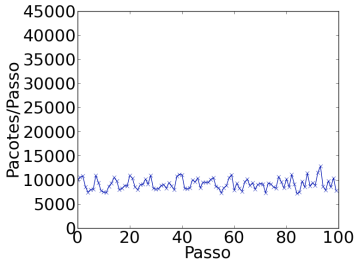


(c) Tráfego escalado 4x.

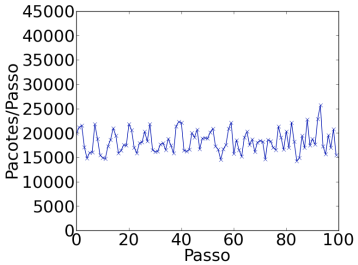
Figura 7.8: Perfis de tráfego utilizados.

As primeiras simulações efetuadas não utilizam nenhum Mecanismo de Dinâmica Rápida, isto é, nessas simulações o servidor é mantido todo o tempo no estado da CPU de maior capacidade de encaminhamento e maior consumo energético. Essas

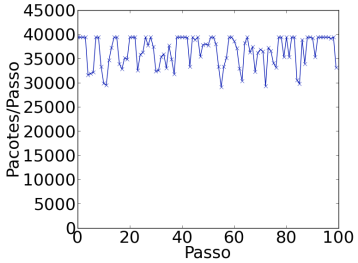
simulações servem como referencial para o limite superior de desempenho de encaminhamento e de máximo energético. Os resultados das simulações sem uso de MDR estão na Figura 7.9.



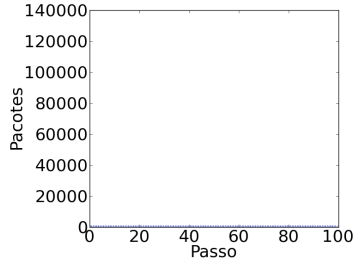
(a) Pacotes encaminhados - Tráfego normal.



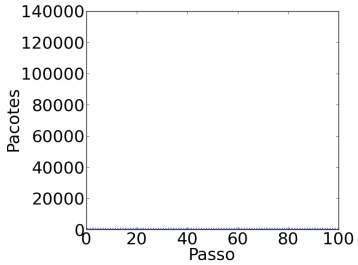
(b) Pacotes encaminhados - Tráfego escalado 2x.



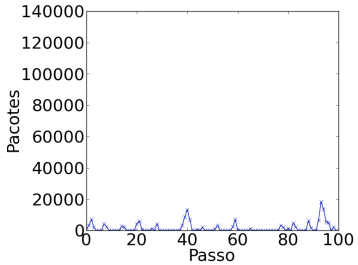
(c) Pacotes encaminhados - Tráfego escalado 4x.



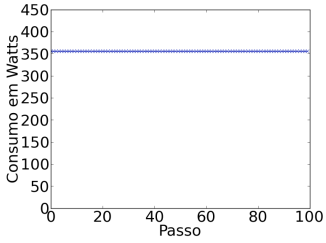
(d) Uso de buffer - Tráfego normal.



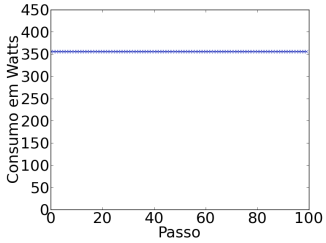
(e) Uso de buffer - Tráfego escalado 2x.



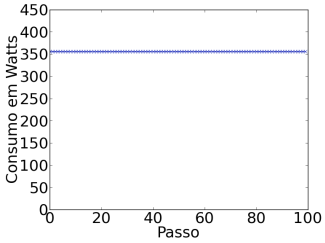
(f) Uso de buffer - Tráfego escalado 4x.



(g) Consumo energético - Tráfego normal.



(h) Consumo energético - Tráfego escalado 2x.



(i) Consumo energético - Tráfego escalado 4x.

Figura 7.9: Simulações sem uso de MDR.

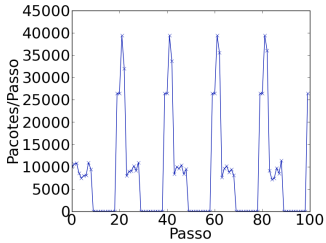
Conforme esperado, utilizar a CPU sempre em seu estado de maior consumo e capacidade de encaminhamento faz com que os pacotes sejam encaminhados imedi-

atamente sempre que a quantidade de pacotes demandados é inferior à capacidade máxima de encaminhamento do servidor. Dessa forma, o buffer do servidor encontra-se sempre vazio para os testes com o tráfego normal e com o tráfego escalado por um fator de duas vezes. Com o tráfego escalado por um fator de quatro vezes, há momentos em que ocorrem picos de demanda superiores à capacidade máxima da CPU, fazendo com que o buffer tenha que ser utilizado para absorver a demanda excedente. Também é esperado o fato do consumo energético do servidor ser constante e ser o consumo máximo visto que não há variação no estado da CPU, mantendo-se sempre no estado de maior capacidade de encaminhamento e de consumo.

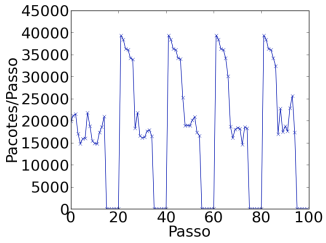
As próximas simulações são efetuadas com o MDR-SA e estão exibidas na Figura 7.10. O MDR-SA é configurado para ser executado em intervalos de 20 rodadas.

Como pode-se observar na Figura 7.10, o MDR-SA apresenta em suas curvas de encaminhamento de pacotes e de consumo energético um comportamento de onda dente-de-serra. Isso se deve à maneira como os estados da CPU são alocados dentro do intervalo de tempo, começando com os estados de maior capacidade de processamento e terminando com os estados de mais baixa capacidade de processamento, conforme descrito na Subseção 6.1.2. Esse comportamento também se reflete nas curvas de utilização do buffer: enquanto o MDR-SA mantém a CPU em estados de economia de energia, onde não há encaminhamento de pacotes, o buffer passa a acumular os pacotes que chegam da rede virtual, e, enquanto o MDR-SA mantém a CPU em estados de mais alto consumo energético, o buffer é esvaziado encaminhando os pacotes acumulados. Nos cenários de simulação com tráfego normal e tráfego escalado por um fator de duas vezes, a otimização executada pelo MDR-SA consegue obter configurações de estados da CPU a serem utilizados que conseguem economizar energia e ao mesmo tempo não deixar o buffer tender à saturação. O mesmo não ocorre no cenário com tráfego intenso, onde o buffer torna-se cada vez mais cheio e satura por diversas vezes. Isso ocorre porque o algoritmo de otimização do MDR-SA está configurado para um compromisso entre a economia de energia e provimento de capacidade de processamento que lida bem com cargas baixas e intermediárias constantes, mas que por evitar usar os estados de mais alto gasto da CPU por períodos prolongados, se apoia na utilização do buffer para absorver eventuais picos de demanda. Como o cenário com tráfego escalado por um fator de quatro vezes apresenta sempre carga intensa, a configuração utilizada do algoritmo de otimização acaba acumulando pacotes no buffer até sua saturação. Para situações de demanda intensa mais prolongada, é necessário reconfigurar o parâmetro β da função de custo do algoritmo de otimização para valores mais altos, fazendo com que o algoritmo de otimização priorize mais o encaminhamento de pacotes em detrimento da economia de energia.

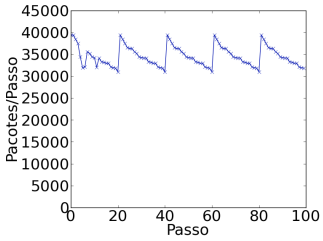
Finalmente são efetuadas as simulações com o MDR-ECO, exibidas na Fi-



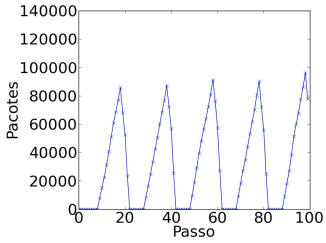
(a) Pacotes encaminhados - Tráfego normal.



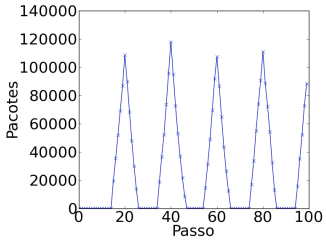
(b) Pacotes encaminhados - Tráfego escalado 2x.



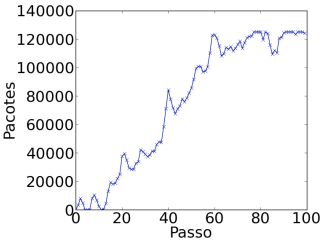
(c) Pacotes encaminhados - Tráfego escalado 4x.



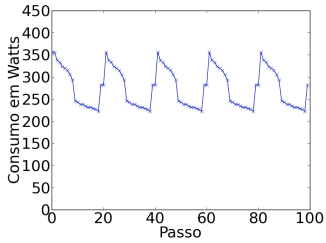
(d) Uso de buffer - Tráfego normal.



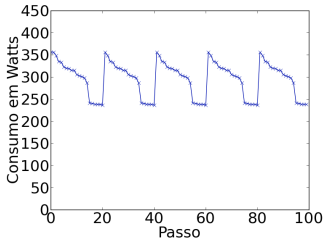
(e) Uso de buffer - Tráfego escalado 2x.



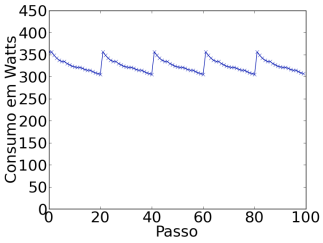
(f) Uso de buffer - Tráfego escalado 4x.



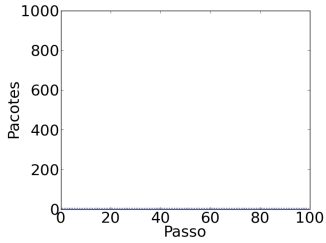
(g) Consumo energético - Tráfego normal.



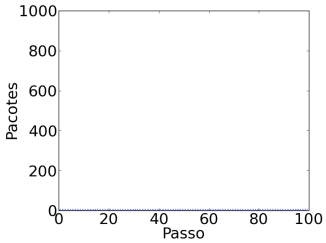
(h) Consumo energético - Tráfego escalado 2x.



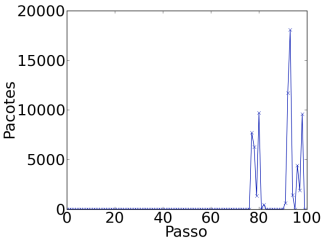
(i) Consumo energético - Tráfego escalado 4x.



(j) Perdas de pacotes - Tráfego normal



(k) Perdas de pacotes - Tráfego escalado 2x.

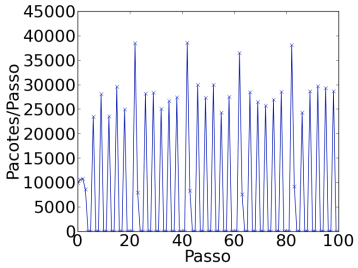


(l) Perdas de pacotes - Tráfego escalado 4x.

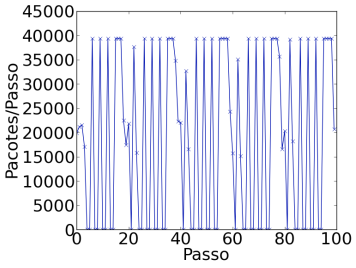
Figura 7.10: Simulações com o uso do MDR-SA.

gura 7.11. Nessas simulações, o ECO também é executado em intervalos de 20 passos.

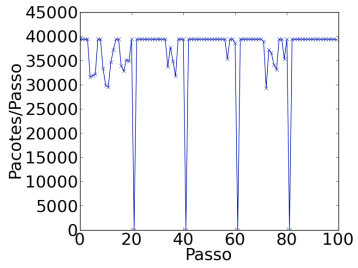
Como pode-se observar na Figura 7.11, o MDR-ECO apresenta nas curvas de encaminhamento de pacotes uma seqüência de picos de envio de pacotes seguidos de momentos sem encaminhamento nenhum. Isso se deve à política de distribuição do tempo a ser gasto nos estados planejados, conforme descrito na Seção 4.1, que alterna continuamente a CPU entre os estados planejados, respeitando o tempo to-



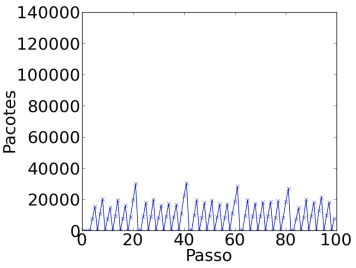
(a) Pacotes encaminhados - Tráfego normal.



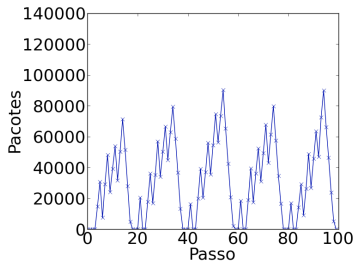
(b) Pacotes encaminhados - Tráfego escalado 2x.



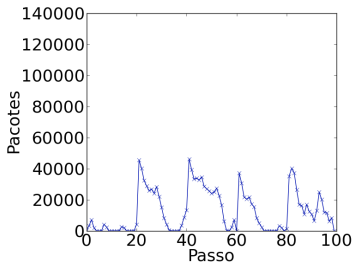
(c) Pacotes encaminhados - Tráfego escalado 4x.



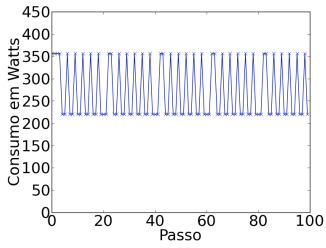
(d) Uso de buffer - Tráfego normal.



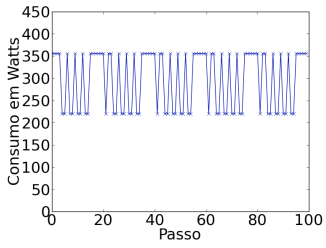
(e) Uso de buffer - Tráfego escalado 2x.



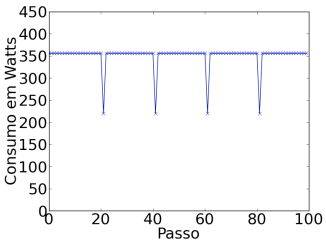
(f) Uso de buffer - Tráfego escalado 4x.



(g) Consumo energético - Tráfego normal.



(h) Consumo energético - Tráfego escalado 2x.



(i) Consumo energético - Tráfego escalado 4x.

Figura 7.11: Simulações com o uso do MDR-ECO.

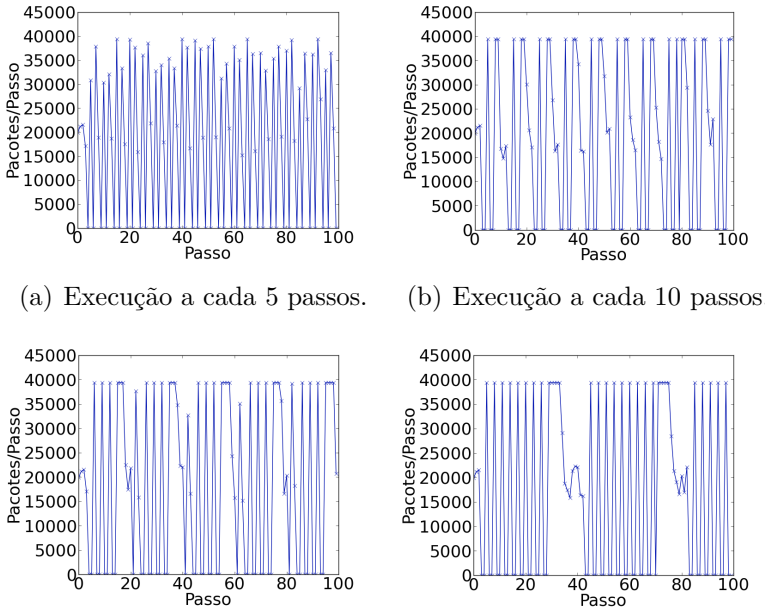
tal a ser gasto em cada estado, para diminuir o impacto no uso do buffer causado pelo estado *Otiose* de economia de energia. Outro aspecto que pode ser observado é que quando a intensidade da carga é aumentada, o comportamento do ECO tende ao comportamento no cenário com a CPU sempre no estado de maior consumo energético e capacidade de encaminhamento. Ao observar o comportamento do buf-

fer, pode-se notar a mesma questão da seqüência de picos em todos os cenários. No cenário de baixa carga, a seqüência de picos é bem intensa e assume valores baixos. No cenário de carga intermediária nota-se que além da seqüência de picos baixos presentes no cenário de carga baixa, encontra-se uma outra seqüência de picos maiores e com maior duração. A duração desses picos remete ao período de planejamento do ECO nesse cenário, que é de 20 passos. Em cada intervalo o ECO planeja o uso na maior parte do tempo do estado *Efficient* de encaminhamento de pacotes e no tempo restante o uso do estado *Otiose* de economia de energia. Dada a política de chaveamento entre os estados, enquanto o tempo total planejado para o estado *Otiose* não é atingido, existe uma tendência de acúmulo de pacotes devido à carga apresentada não ser baixa e, quando o tempo total planejado para o estado *Otiose* é atingido, a CPU gasta o restante de seu tempo no estado *Efficient*, fazendo com que o buffer seja esvaziado e todos os pacotes demandados sejam encaminhados. No cenário de carga intensa, pode-se notar que o padrão do uso do buffer aproxima-se ao padrão do cenário usando-se a CPU no estado de maior encaminhamento e consumo energético. Isso deve-se ao fato do ECO, em situações de alta demanda, adotar uma postura de priorizar o encaminhamento de pacotes em detrimento da economia de energia. Como pode-se notar também, o uso do buffer é maior do que no cenário usando o estado fixo na CPU e isso deve-se aos poucos momentos em que o ECO entra no estado *Otiose* para economizar energia. Observando-se o comportamento do consumo de energia, pode-se notar claramente a sua relação com a política de distribuição do tempo gasto em cada estado da CPU adotada no ECO, podendo-se observar a mesma seqüência de picos e baixas de consumo. Pode-se notar também que os períodos de baixa de consumo tornam-se menos freqüentes conforme a carga é aumentada. É importante notar que nos dados para caracterizar a CPU utilizados e provenientes de [45], os estados *Efficient* e *Cautious* acabam sendo definidos como o mesmo estado da CPU, dado que o estado de maior encaminhamento de pacotes e o estado com melhor eficiência energética de encaminhamento para esta CPU coincidem. Uma última observação importante é que, conforme mencionado na Seção 4.1, a utilização do estado *Otiose* necessariamente sacrifica o uso do buffer em troca da economia de energia fornecida, como fica claro ao observar a menor utilização do buffer no caso de carga intensa do que no estado de carga intermediária.

7.5.2 Simulações Variando o Número de Rodadas entre Execuções do Mecanismo de Dinâmica Rápida

Nesta subseção são apresentados os testes para verificar o efeito da variação de número de rodadas entre execuções do MDR-ECO em seu desempenho de encaminhamento e de economia de energia. Os testes são executados com quatro intervalos

distintos: 5, 10, 20 e 40 rodadas. A definição do número de rodadas foi feita de forma a apresentar cenários onde o MDR-ECO seria executado com altíssima freqüência (a cada 5 rodadas), alta freqüência (a cada 10 rodadas), freqüência padrão utilizada nos outros testes (a cada 20 rodadas) e baixa freqüência (a cada 40 rodadas). Opta-se por utilizar o perfil de tráfego HTTP da CAIDA escalado por um fator de duas vezes para a execução dos testes, colocando o servidor em uma situação de carga intermediária. Os resultados de encaminhamento de pacotes variando a freqüência de execução do MDR-ECO são exibidos na Figura 7.12.



(a) Execução a cada 5 passos.

(b) Execução a cada 10 passos.

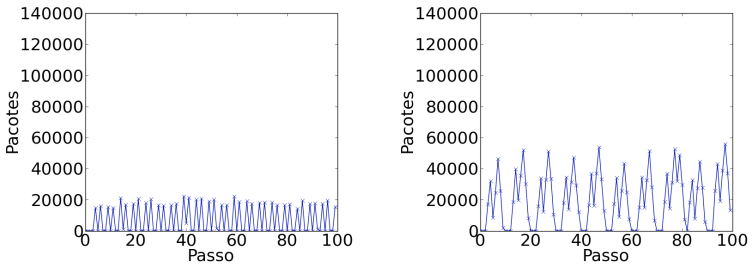
(c) Execução a cada 20 passos.

(d) Execução a cada 40 passos.

Figura 7.12: Encaminhamento de pacotes variando a freqüência de execução do MDR-ECO.

Como pode-se observar na Figura 7.12, conforme aumenta-se o intervalo entre as execuções do MDR-ECO, nota-se que a CPU passa a operar cada vez mais em sua capacidade máxima quando no estado de encaminhamento. Outra observação relevante é a questão da proporcionalidade do tempo gasto pela CPU entre os estados *Efficient* e *Otiose*. Conforme se aumenta o intervalo entre as execuções do MDR-ECO, pode-se notar um aumento no tempo consecutivo que a CPU gasta no estado *Efficient*, embora a freqüência desses momentos de uso consecutivo do estado *Efficient* diminua de forma mais agressiva, proporcionalmente ao aumento do intervalo entre execuções. Os resultados de uso de buffer variando a freqüência de execução do MDR-ECO são exibidos na Figura 7.13.

Conforme pode-se observar na Figura 7.13, conforme o intervalo entre execuções aumenta, o uso do buffer aumenta, refletindo o efeito observado no encaminhamento de pacotes gerado pela menor freqüência dos períodos prolongados de execução do

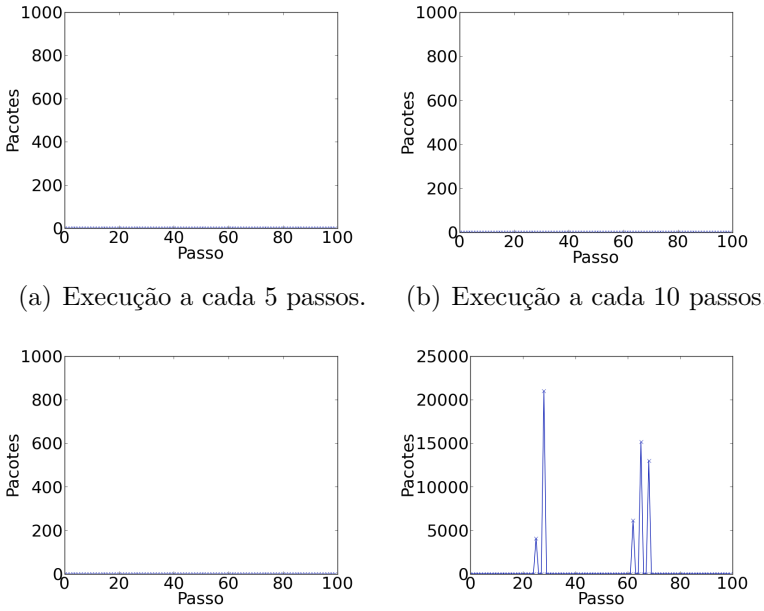


(a) Execução a cada 5 passos. (b) Execução a cada 10 passos.

(c) Execução a cada 20 passos. (d) Execução a cada 40 passos.

Figura 7.13: Uso do buffer variando a freqüência de execução do MDR-ECO.

estado *Efficient*. Para o caso em que a execução é efetuada a cada 40 passos, o acúmulo de pacotes no buffer dentro de uma execução é tão crítico que se pode verificar inclusive a perda de pacotes, conforme se pode observar na Figura 7.14.



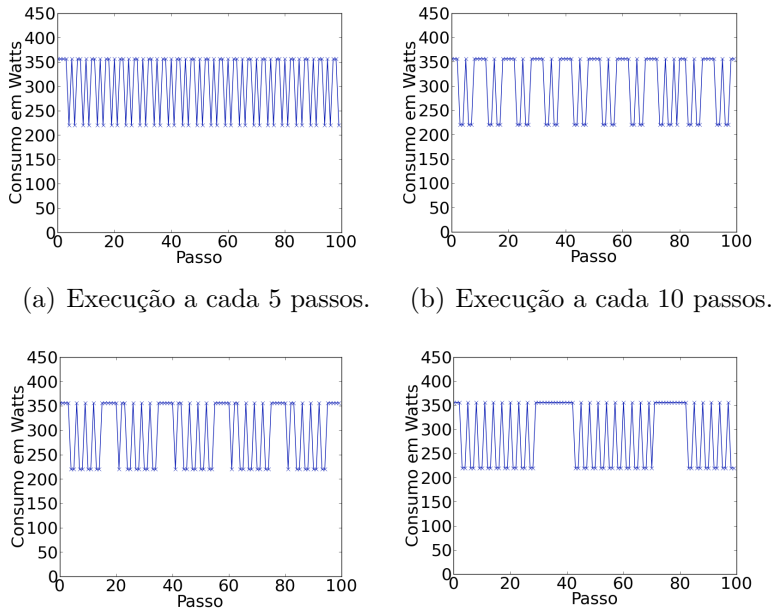
(a) Execução a cada 5 passos. (b) Execução a cada 10 passos.

(c) Execução a cada 20 passos. (d) Execução a cada 40 passos.

Figura 7.14: Perdas de pacotes variando a freqüência de execução do MDR-ECO.

Finalmente, a Figura 7.15 exibe o comportamento do consumo energético ao

variar o intervalo entre execuções do MDR-ECO.



(a) Execução a cada 5 passos.

(b) Execução a cada 10 passos.

(c) Execução a cada 20 passos.

(d) Execução a cada 40 passos.

Figura 7.15: Consumo energético variando a freqüência de execução do MDR-ECO.

Como pode-se verificar na Figura 7.15, o consumo de energia também reflete o padrão observado no encaminhamento de pacotes no período final das execuções do MDR-ECO. Como a freqüência destes períodos prolongados de uso do estado *Efficient* diminui mas agressivamente do que o aumento na duração dos períodos, conforme aumenta-se o intervalo entre as execuções, diminui-se o consumo médio.

Verificando todos os resultados em conjunto, pode-se notar que quanto mais freqüente a execução do ECO, maior é o consumo energético e menor o uso do buffer, enquanto o comportamento contrário pode ser observado ao aumentar o intervalo entre as execuções, até um ponto onde se passa a observar perdas de pacotes. Como mencionado na Seção 4.1, dentro de um período de execução, o ECO atua como se estivesse atendendo à demanda média do período. Combinando-se isso ao fato do estimador de demanda perder a sensibilidade à variação do tráfego conforme o intervalo entre execuções aumenta, o ECO pára de responder apropriadamente à carga demandada e passa a sofrer danos no encaminhamento de pacotes devido à maior sensibilidade aos picos de demanda.

Outra questão que pode ser levantada após a análise do comportamento do ECO nos testes desta subseção e da anterior é a da criação de um mecanismo para variar o intervalo de execução do ECO de acordo com as condições da rede e de objetivos de SLA das redes virtuais e objetivos de consumo de energia. Tal mecanismo poderia ser utilizado para configurar o ECO de forma a gerar economia de energia mais agressiva em servidores encaminhando tráfegos de baixa prioridade, especialmente quando a

carga da rede estiver baixa, e para configurar o ECO de forma mais agressiva no encaminhamento de pacotes quando o tráfego encaminhado for de maior prioridade ou a carga da rede estiver mais intensa.

Verificando-se os testes com diferentes cargas de redes, pode-se verificar que o ECO aumentou muito o uso de buffer para o caso de carga intermediária, utilizando-o mais, inclusive, do que no caso com carga intensa. Dependendo do tipo de aplicação, este aumento no uso do buffer pode causar problemas em seu funcionamento. Dessa forma, é interessante modificar o comportamento do ECO para utilizar menos o buffer em casos de carga de rede intermediária, mantendo ainda a característica de economia de energia. O próximo capítulo apresenta uma versão alternativa do ECO que busca resolver esta questão.

Capítulo 8

O ECOv2

Conforme verificado na Seção 7.5, o ECO pode passar a utilizar muito o buffer dependendo da carga na rede e da freqüência com que é executado devido ao uso do estado *Otiose*. Neste capítulo apresenta-se o ECOv2, uma versão alternativa do ECO desenvolvida com o intuito de diminuir a utilização do buffer. Para isso, o ECOv2 apóia-se nas mesmas premissas de funcionamento do ECO, entretanto substituindo o estado *Otiose* pelo estado *Oversaving*.

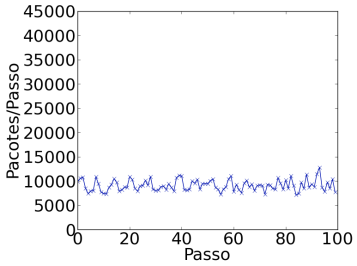
O estado *Oversaving*, assim como o estado *Otiose*, possui o objetivo de economizar energia colocando a CPU em um estado de baixo consumo energético. Entretanto, ao contrário do estado *Otiose*, o estado *Oversaving* encaminha pacotes. O estado *Oversaving* é definido como o estado da CPU que possui o mais baixo consumo energético mas com capacidade de processamento de pacotes, potencialmente, a mais baixa oferecida pela CPU.

Devido a esta alteração, em situações de baixa demanda, o ECOv2 passa a gastar menos tempo no estado *Efficient* do que o ECO e mais tempo no estado *Oversaving* do que o que seria passado no estado *Otiose* usando-se o ECO. Isso deve-se ao fato de parte dos pacotes enviados no estado *Efficient* utilizando-se o ECO, passarem a ser enviados no estado *Oversaving* com o ECOv2.

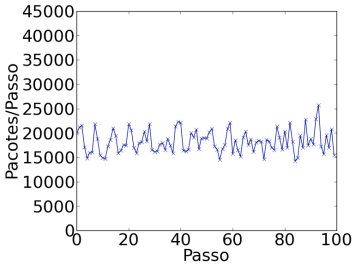
Na seção a seguir será efetuada uma análise do ECOv2 análoga a feita na Seção 7.5 para que possam ser verificadas as mudanças de comportamento causadas pela substituição do estado *Otiose* pelo estado *Oversaving*.

8.1 Análise em Curta Escala de Tempo

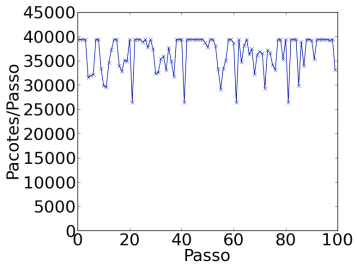
Nesta seção objetiva analisar o comportamento do ECOv2 utilizando simulações em curta escala de tempo, avaliando as características de encaminhamento de pacotes e de consumo energético. Dessa forma, são conduzidos testes com os mesmos cenários apresentados na Seção 7.5.



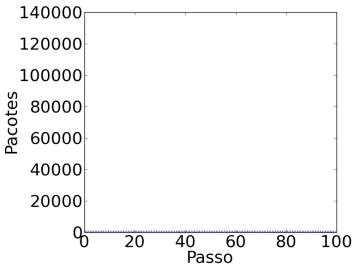
(a) Pacotes encaminhados - Tráfego normal.



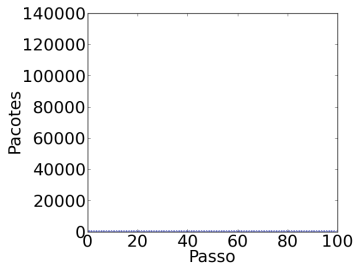
(b) Pacotes encaminhados - Tráfego escalado 2x.



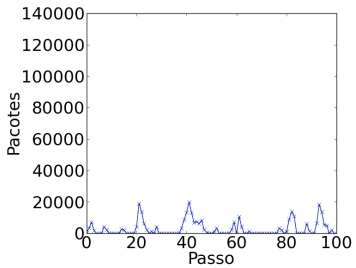
(c) Pacotes encaminhados - Tráfego escalado 4x.



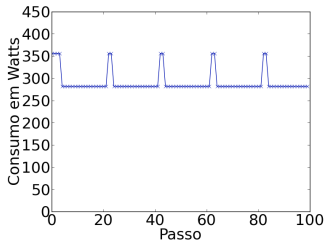
(d) Uso de buffer - Tráfego normal.



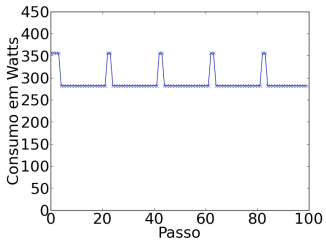
(e) Uso de buffer - Tráfego escalado 2x.



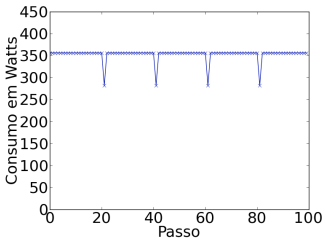
(f) Uso de buffer - Tráfego escalado 4x.



(g) Consumo energético - Tráfego normal.



(h) Consumo energético - Tráfego escalado 2x.



(i) Consumo energético - Tráfego escalado 4x.

Figura 8.1: Simulações com o uso do MDR-ECOv2.

8.1.1 Simulações Variando a Intensidade do Tráfego

Esta Subseção apresenta os testes com o ECOv2 variando a intensidade do tráfego ao qual o mecanismo é exposto. Para isso, efetuam-se simulações utilizando um servidor com apenas um roteador virtual carregando tráfego baseado no perfil HTTP dos dados fornecidos pela CAIDA. As simulações duram cem rodadas e são efetuadas

MDR	Consumo (W)	Uso Buffer (Pacotes)
Sem MDR	356,0	0,0
ECO	270,3	9176,8
ECOv2	290,9	0,0

Tabela 8.1: Desempenho médio dos MDRs - tráfego normal.

MDR	Consumo (W)	Uso Buffer (Pacotes)
Sem MDR	356,0	0,0
ECO	296,2	30778,0
ECOv2	290,9	0,0

Tabela 8.2: Desempenho médio dos MDRs - tráfego 2x.

Sem MDR	356,0	1632,4
ECO	350,6	13027,6
ECOv2	353,0	2949,6

Tabela 8.3: Desempenho médio dos MDRs - tráfego 4x.

em três cenários. O primeiro cenário utiliza o tráfego HTTP inalterado, constituindo um cenário de baixa carga. O segundo cenário utiliza o mesmo perfil de tráfego, entretanto escalado por um fator de duas vezes, constituindo assim um cenário de carga intermediária. O terceiro cenário é de carga intensa, utilizando o perfil de tráfego escalado por um fator de quatro vezes. O MDR-ECOv2 é executado a cada 20 passos de simulação. Os resultados dos testes estão exibidos na Figura 8.1.

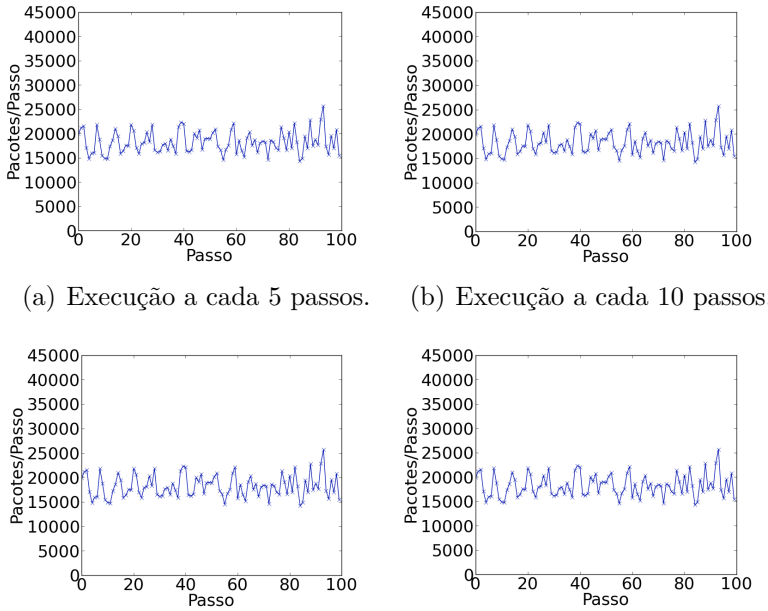
Conforme pode-se observar na Figura 8.1, a substituição do estado *Otiose* pelo estado *Oversaving* melhorou consideravelmente as características de encaminhamento do ECOv2 em relação ao ECO, fazendo com que o uso do buffer seja eliminado nas situações de tráfego baixo e intermediário e drasticamente reduzido na situação de tráfego intenso. Pode-se observar fazendo uma comparação dos resultados do ECOv2 com os resultados sem uso de MDR na Figura 7.9 que o comportamento de encaminhamento do ECOv2 aproxima-se muito do comportamento obtido sem uso de MDR, entretanto obtendo-se resultados muito melhores quanto ao gasto energético.

Em relação ao gasto energético, o ECOv2 gasta mais energia que o ECO nos cenários de baixa carga e de carga intensa, mas gasta menos energia no cenário de carga intermediária, como pode-se observar nas Tabelas 8.1, 8.2 e 8.3, que sintetizam a média dos resultados em curta escala de tempo variando o tráfego para os cenários sem MDR, com ECO e com ECOv2 para os três cenários.

É importante enfatizar que a maior economia de energia obtida pelo ECOv2 em relação ao ECO no cenário com carga intermediária advém do fato que o tempo que era gasto no estado *Otiose* sem encaminhar pacotes foi substituído por uma maior quantidade de tempo gasto no estado *Oversaving* que, por encaminhar pacotes, permitiu a drástica redução do tempo gasto no estado *Efficient*, que, embora mais eficiente, possui maior gasto energético que o estado *Oversaving*.

8.1.2 Simulações Variando o Número de Rodadas Entre Execuções do Mecanismo de Dinâmica Rápida

Nesta subseção são apresentados testes para avaliar o comportamento do MDR-ECOv2 em relação ao encaminhamento de pacotes e consumo energético ao variar o intervalo entre as suas execuções. Os testes são efetuados utilizando os intervalos de 5, 10, 20 e 40 rodadas, os mesmos utilizados nos testes com o ECO na Subseção 7.5.2. Assim como nos testes com o ECO, utiliza-se um servidor com um roteador virtual e perfil de tráfego HTTP escalado por um fator de duas vezes. Os resultados de encaminhamento de pacotes variando o intervalo entre as execuções do MDR-ECOv2 estão na Figura 8.2.



(a) Execução a cada 5 passos. (b) Execução a a cada 10 passos.

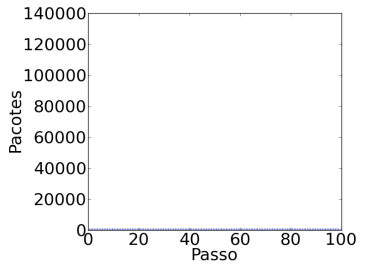
(c) Execução a cada 20 passos. (d) Execução a cada 40 passos.

Figura 8.2: Encaminhamento de pacotes variando a freqüência de execução do MDR-ECOv2.

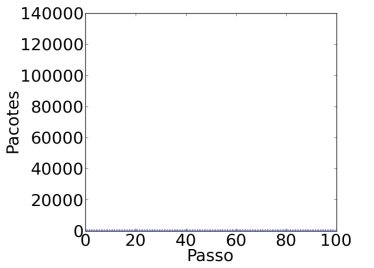
Como pode-se observar na Figura 8.2, o encaminhamento do ECOv2 não é influenciado pela variação do intervalo entre execuções do MDR nesse cenário. Isso pode ser explicado pelo fato da capacidade de encaminhamento do estado *Oversaving* ser o suficiente para encaminhar a carga de tráfego demandada. A observação do uso do buffer na Figura 8.3 confirma esse fato, visto que independente do número de rodadas de intervalo, o ECOv2 não precisou armazenar pacotes em buffer.

A observação da Figura 8.4 entretanto, indica que o consumo energético do ECOv2 tende a diminuir quando o intervalo entre execuções é aumentado, visto que o estado *Oversaving* passa a ser mais utilizado para o encaminhamento de pacotes.

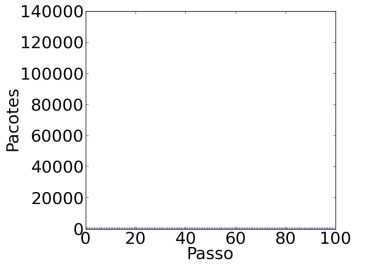
Comparando os resultados do ECOv2 nesses testes com os resultados obtidos



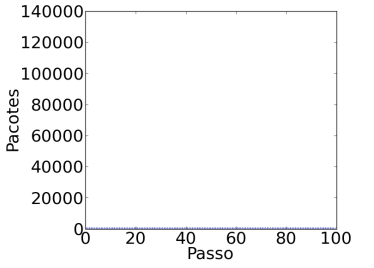
(a) Execução a cada 5 passos.



(b) Execução a cada 10 passos.

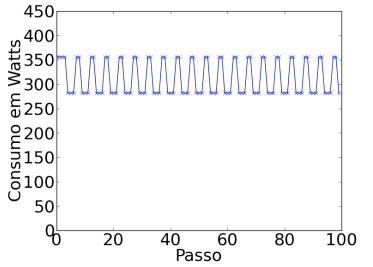


(c) Execução a cada 20 passos.

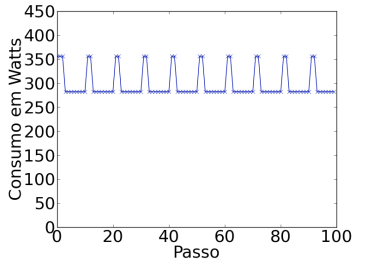


(d) Execução a cada 40 passos.

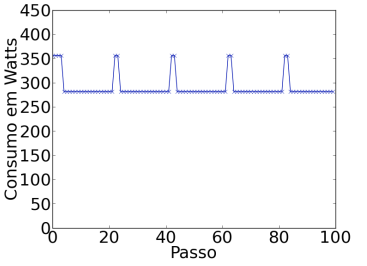
Figura 8.3: Uso do buffer variando a freqüência de execução do MDR-ECOv2.



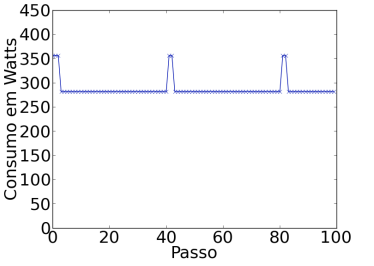
(a) Execução a cada 5 passos.



(b) Execução a cada 10 passos.



(c) Execução a cada 20 passos.



(d) Execução a cada 40 passos.

Figura 8.4: Consumo energético variando a freqüência de execução do MDR-ECOv2.

pelo ECO, pode-se observar que o ECOv2 apresentou a característica desejável de diminuição do consumo energético ao aumentar o intervalo entre as execuções, sem apresentar as características danosas de maior utilização do buffer, chegando até a acarretar em perdas de pacotes. A adição da capacidade de encaminhamento de pacotes no estado de economia de energia tornou um cenário ruim para o ECO em um

cenário favorável para o ECOv2. É provável que as mesmas tendências observadas nos testes com o ECO possam ser observadas com o ECOv2 caso seja utilizado um cenário com carga mais intensa do que a capacidade de encaminhamento do estado *Oversaving*. Dessa forma, picos de demanda tenderiam a ser mascarados pela menor freqüência de execução do MDR e não seriam atendidos pela capacidade de encaminhamento do estado *Oversaving*, acarretando na maior utilização do buffer.

A observação dos resultados desta seção sugere a investigação de um novo mecanismo que adote um comportamento híbrido do ECO e ECOv2 dependendo da situação da carga no servidor. O novo mecanismo poderia potencialmente utilizar o estado *Otiose* combinado com os estados *Oversaving* e *Efficient* em situações de baixa carga e os estados *Oversaving*, *Efficient* e *Cautious* em cenários de carga intermediária e intensa.

Finalmente, pode-se pensar também em alterar o comportamento de uma futura versão do ECO de acordo com o tipo de tráfego encaminhado no servidor. Caso o servidor esteja carregando tráfego sensível a latência, o ECO poderia reconfigurar-se para ser menos agressivo na economia de energia, priorizando o rápido encaminhamento de pacotes. Caso o servidor esteja carregando tráfego insensível a latência, o ECO podeira reconfigurar-se para economizar energia de forma mais agressiva em detrimento de um maior uso do buffer.

Capítulo 9

Conclusão

9.1 Considerações Finais

A Internet, embora seja um tremendo sucesso, padece de um processo de ossificação e não mais consegue atender aos requisitos das aplicações existentes. Para permitir a inovação no núcleo da rede e atender aos novos requisitos das aplicações atuais, a comunidade científica chegou ao consenso sobre a necessidade de uma nova Internet. Dentro das propostas de arquitetura para a Internet do Futuro nota-se uma corrente, denominada pluralista, que faz forte uso da virtualização como meio de viabilizar o funcionamento de múltiplas pilhas de protocolos de rede concorrentes. Nesse cenário, diversos trabalhos apontam os roteadores de software como plataforma para prover a flexibilidade demandada pelas redes virtuais. Embora apresentem a flexibilidade necessária para o suporte às redes virtuais, os roteadores de software padecem do problema de baixas taxas de encaminhamento em relação aos tradicionais roteadores baseados em hardware dedicado e firmware proprietário. Para superar esse problema, são propostos os roteadores de software em cluster, que agregam a flexibilidade dos roteadores de software ao desempenho obtido por roteadores de hardware dedicado.

O uso de roteadores de software em cluster, entretanto, levanta diferentes questões, entre as quais a alocação dos recursos do cluster entre as redes virtuais e o gerenciamento de energia. No contexto de um mundo onde a computação verde toma cada vez mais importância por aspectos que variam desde as questões ambientais de sustentabilidade até as questões econômicas do gasto energético da infraestrutura de TI, este trabalho propôs o ECO-ALOC. O ECO-ALOC é um mecanismo de alocação energeticamente eficiente de recursos para ambientes de redes virtuais em roteadores de software em cluster. O ECO-ALOC é composto por dois mecanismos que atuam em escopos e escalas de tempo diferentes. O ECO (*Efficient/Cautious/Otiose* é um mecanismo que atua no escopo de um servidor comandando em alta granularidade

o consumo energético da CPU ao mesmo tempo em que provê a capacidade de processamento necessária para atender a demanda de tráfego das redes virtuais. Já o ALOC (*Automatic Load Organizer for Cluster*) é um mecanismo que atua no escopo do cluster como um todo em uma escala de tempo maior, migrando redes virtuais para balanceamento e consolidação das redes virtuais, desligando e religando servidores do cluster de forma a economizar energia ao mesmo tempo em que atende à demanda de recursos das redes virtuais.

Nesta dissertação de mestrado, inicialmente buscou-se na literatura as ações que poderiam ser utilizadas para gerenciar a alocação de recursos e os gastos energéticos dos servidores que formam o roteador de software em cluster, posteriormente definindo quais ações utilizar e qual modelo de roteador de software em cluster utilizar dentre os dois encontrados na literatura: o modelo do Routebricks e do Flowstream. Devido a possibilidade de desligar servidores para economizar energia, foi adotado o modelo baseado no Flowstream. Dado o problema abordado e a não identificação de simuladores apropriados, foi proposto um modelo de funcionamento de roteador de software em cluster e foi realizada a sua implementação para avaliar os mecanismos propostos.

Para melhor avaliar o ECO, foi desenvolvido um Mecanismo de Dinâmica Rápida alternativo, baseado em uma técnica de otimização. A escolha de desenvolver um mecanismo alternativo baseado em otimização veio da observação na literatura do uso frequente desse tipo de técnica para problemas envolvendo alocação de recursos.

A avaliação dos mecanismos propostos foi feita utilizando-se dados de tráfegos reais provenientes da RNP e da CAIDA. Além de mostrar a eficácia dos mecanismos propostos, com destaque para a economia de energia de 93% obtida pelo ECO-ALOC em um cenário de baixa carga utilizando dados provenientes do enlace da RNP entre Rio de Janeiro e Belo Horizonte, forneceu artifícios para melhor compreender o funcionamento do ECO-ALOC em diversos cenários e inspirar modificação para sua melhoria. Verificando-se o alto do uso de buffer do ECO em alguns cenários, propôs-se o ECOv2: uma versão alternativa do ECO que substitui o uso do estado *Otiose*, que possui baixo consumo energético e nenhuma capacidade de encaminhamento de pacotes, pelo estado *Oversaving*, definido como o estado da CPU de mais baixo consumo energético com capacidade de encaminhamento de pacotes. Não somente o ECOv2 atingiu o objetivo de diminuição de uso de buffer, como também conseguiu desempenho de encaminhamento de pacotes próximo ao limite dado pela alocação estática do estado de CPU com maior capacidade de processamento e consumo mantendo um consumo energético próximo ao do ECO, perdendo significativamente somente no cenário de carga baixa.

9.2 Trabalhos Futuros

A análise do comportamento do ECO, ECOv2 e ALOC sugerem alguns trabalhos futuros. Um primeiro trabalho futuro seria a investigação de um mecanismo para a configuração dinâmica dos parâmetros do ECO-ALOC de acordo com as condições da rede e com condições impostas por acordos de QoS com as redes virtuais. Pode-se fazer estudos para identificar os principais possíveis cenários de rede e, em cada um deles, verificar quais configurações de parâmetros são as mais apropriadas para o objetivo de economizar energia ou priorizar o encaminhamento de pacotes. Tendo estes dados, pode-se criar um mecanismo que analise a situação da rede e faça uma correlação com os cenários da base de dados para aplicar os parâmetros corretos dada a situação da rede e a relação desejada de prioridade entre economizar energia e encaminhar pacotes. Os testes em curta escala de tempo com o ECO e ECOv2 sugerem a possibilidade de formar um mecanismo que se aproxime do ECO em situações de carga muito baixa do servidor e que se aproxime do ECOv2 em situações de carga intermediária ou intensa para formar um novo MDR com melhores resultados.

Uma outra questão a ser considerada é a investigação de outros componentes de hardware para aumentar a economia de energia. A investigação de estimadores melhores de demanda tanto para o ECO quanto para o ALOC é sempre possível dada a vastidão da área de pesquisa de estimadores. O desenvolvimento de um mecanismo de alocação de banda dos enlaces externos entre as redes virtuais que leve em consideração o consumo energético da rede é um outro possível trabalho futuro.

Outro aspecto é explorar outros cenários de simulação para descobrir novos aspectos relevantes para a resolução do problema de alocação de recursos de processamento de pacotes em roteadores de software em cluster. Outro possível trabalho futuro é a implementação de um protótipo para validar na prática a eficiência do mecanismo proposto e verificar a adequação do simulador para modelar o problema estudado. O simulador pode ser estendido para modelar também o problema da alocação de banda entre as redes virtuais.

Uma outra questão interessante para ser explorada é a aplicação do MDR alternativo em ambientes de dinâmica mais lenta do que os ambiente de redes, como um datacenter ou uma nuvem. Pode-se explorar as diversas configurações possíveis de pesos e desenvolver um mecanismo que os reconfigure dinamicamente de acordo com a carga e sensibilidade a atrasos no processamento para melhorar a eficiência energética do ambiente onde for aplicado.

Outro ponto que pode ser explorado é a aplicação do ECO-ALOC em nuvens para melhorar a eficiência energética desses ambientes. O problema de alocação de recursos na nuvem é consideravelmente semelhante ao problema de alocação de

recursos de processamento de pacotes em roteadores de software em cluster. Dessa forma, o ECO-ALOC pode ser aplicado em nuvens se considerar que os serviços da nuvem dentro de máquinas virtuais devem ser alocados entre os servidores da nuvem, tomando o cuidado de efetuar a alocação dos serviços levando em consideração não somente o uso de processador e memória RAM, mas também o uso de banda de cada serviço.

Finalmente, pode-se pensar em expandir o ECO-ALOC para lidar com ambientes de redes móveis, integrando-se a mecanismos que resolvem problemas específicos destes ambientes como a seleção da infraestrutura apropriada de acesso a rede em dispositivos móveis que dispõe de diversas interfaces [70], e a integração do ECO-ALOC com outros mecanismos de gerenciamento de ambientes de redes virtuais, como mecanismos que tratem da segurança [71].

Referências Bibliográficas

[1] *Rede Ipê da RNP.* http://www.rnp.br/en/backbone/index.php, Acessado em fevereiro de 2011.

[2] FELDMANN, A. "Internet clean-slate design: what and why?" *ACM SIGCOMM Comput. Commun. Rev.*, pp. 59–64, 2007.

[3] SPYROPOULOS, T., FDIDA, S., KIRKPATRICK, S. "Future Internet: Fundamentals and Measurement", *ACM SIGCOMM Comput. Commun. Rev.*, pp. 101–106, 2007.

[4] MOREIRA, M. D. D., FERNANDES, N. C., COSTA, L. H. M. K., et al. "Internet do Futuro: Um Novo Horizonte", *Minicursos do Simpósio Brasileiro de Redes de Computadores - SBRC'2009*, pp. 1–59, 2009.

[5] STOICA, I., ADKINS, D., ZHUANG, S., et al. "Internet Indirection Infrastructure", *IEEE/ACM Trans. Netw.*, pp. 205–218, 2004.

[6] HE, J., ZHANG-SHEN, R., LI, Y., et al. "DaVinci: Dynamically Adaptive Virtual Networks for a Customized Internet". In: *ACM CoNext*, 2008.

[7] EGI, N., GREENHALGH, A., HANDLEY, M., et al. "Towards high performance virtual routers on commodity hardware", *ACM CoNEXT*, dez. 2008.

[8] FEAMSTER, N., GAO, L., REXFORD, J. "How to lease the Internet in your spare time", *ACM SIGCOMM Comput. Commun. Rev.*, pp. 61–64, 2007.

[9] TENNENHOUSE, D. L., WETHERALL, D. J. "Towards an active network architecture", *SIGCOMM Comput. Commun. Rev.*, pp. 5–17, 1996.

[10] BRADEN, R., FABER, T., HANDLEY, M. "From protocol stack to protocol heap: role-based architecture", *SIGCOMM Comput. Commun. Rev.*, pp. 17–22, 2003.

[11] ANDERSON, T., PETERSON, L., SHENKER, S., et al. "Overcoming the Internet Impasse through Virtualization", *IEEE Comput*, pp. 34–41, 2005.

[12] CROWCROFT, J., HAND, S., MORTIER, R., et al. "Plutarch: an Argument for Network Pluralism". In: *ACM SIGCOMM Workshop on Future Directions in Network Architecture*, pp. 258–266, set. 2003.

[13] ANDERSON, T., PETERSON, L., SHENKER, S., et al. "Overcoming the Internet Impasse through Virtualization", *IEEE Comput.*, pp. 34–41, 2005.

[14] CHOWDHURY, M. K. N., BOUTABA, R. "A survey of network virtualization", *Elsevier Comput. Net.*, pp. 862–876, 2010.

[15] GREENHALGH, A., HUICI, F., HOERDT, M., et al. "Flow processing and the rise of commodity network hardware", *ACM SIGCOMM Comput. Commun. Rev.*, pp. 20–26, 2009.

[16] SCHAFFRATH, G., WERLE, C., PAPADIMITRIOU, P., et al. "Network virtualization architecture: proposal and initial prototype". ACM VISA, pp. 63–72, 2009.

[17] EGI, N., GREENHALGH, A., HANDLEY, M., et al. "Evaluating Xen for Router Virtualization". In: *IEEE ICCCN*, pp. 1256 –1261, 2007.

[18] POPEK, G. J., GOLDBERG, R. P. "Formal requirements for virtualizable third generation architectures", *Commun. ACM*, pp. 412–421, 1974.

[19] DOBRESCU, M., EGI, N., ARGYRAKI, K., et al. "RouteBricks: Exploiting Parallelism To Scale Software Routers". In: *ACM SOSP*, 2009.

[20] ARGYRAKI, K., BASET, S., CHUN, B.-G., et al. "Can software routers scale?" In: *ACM SIGCOMM PRESTO*, pp. 21–26, 2008.

[21] HUANG, S., SESHADRI, D., DUTTA, R. "Traffic Grooming: A Changing Role in Green Optical Networks". In: *IEEE GLOBECOM*, pp. 1–6, 2009.

[22] NEDEVSCHI, S., POPA, L., IANNACCONE, G., et al. "Reducing network energy consumption via sleeping and rate-adaptation". In: *USENIX NSDI*, pp. 323–336, 2008.

[23] BOLLA, R., BRUSCHI, R., RANIERI, A. "Green Support for PC-Based Software Router: Performance Evaluation and Modeling". In: *IEEE ICC*, pp. 1–6, 2009.

[24] *Can the World's Telecoms Slash Their Energy Consumption 1,000-Fold?* http://www.scientificamerican.com/article.cfm?id=green-touch-launch, Acessado em janeiro de 2011.

[25] *Shannon's Theory Explained.* http://www.greentouch.org/index.php?page=shannons-law-explained, Acessado em janeiro de 2011.

[26] PREIST, C., SHABAJEE, P. "Energy Use in the Media Cloud: Behaviour Change, or Technofix?" In: *IEEE CLOUDCOM*, pp. 581–586, 2010.

[27] MOREIRA, M. D. D., LAUFER, R. P., FERNANDES, N. C., et al. "A Stateless Traceback Technique for Identifying the Origin of Attacks from a Single Packet". In: *IEEE ICC'11 CISS'*, A ser publicado no ICC CISS 2011.

[28] FERNANDES, N. C., MOREIRA, M. D. D., MORAES, I. M., et al. "Virtual Networks: Isolation, Performance, and Trends". In: *Annals of Telecommunications*, 2010.

[29] BRADEN, R., FABER, T., HANDLEY, M. "From protocol stack to protocol heap: role-based architecture", *ACM SIGCOMM Comput. Commun. Rev.*, pp. 17–22, 2003.

[30] *JunOS Manual: Configuring Virtual Routers.* http://www.juniper.net/techpubs/software/erx/junose72/swconfig-system-basics/html/virtual-router-config5.html, Acessado em janeiro de 2011.

[31] PISA, P. S., FERNANDES, N. C., CARVALHO, H. E. T., et al. "OpenFlow and Xen-Based Virtual Network Migration". In: *The World Computer Congress 2010 - Network of the Future Conference*, pp. 170–181, 2010.

[32] BARHAM, P., DRAGOVIC, B., FRASER, K., et al. "Xen and the art of virtualization". In: *Proceedings of the nineteenth ACM Symposium on Operating Systems Principles - SOSP03*, out. 2003.

[33] KOHLER, E., MORRIS, R., CHEN, B., et al. "The Click Modular Router", *ACM Transaction on Computer Systems*, v. 18, n. 3, pp. 263–297, 2000.

[34] MCKEOWN, N., ANDERSON, T., BALAKRISHNAN, H., et al. "OpenFlow: enabling innovation in campus networks", *ACM SIGCOMM Comput. Commun. Rev.*, pp. 69–74, 2008.

[35] *Openflow Switch by Pronto.* http://www.openflow.org/foswiki/bin/view/OpenFlow/Deployment/Vendor/Pronto, Acessado em janeiro de 2011.

[36] ANWER, M. B., FEAMSTER, N. "Building a fast, virtualized data plane with programmable hardware". ACM VISA, pp. 1–8, 2009.

[37] NAOUS, J., GIBB, G., BOLOUKI, S., et al. "NetFPGA: reusable router architecture for experimental research". In: *ACM SIGCOMM PRESTO*, pp. 1–7, 2008.

[38] HAN, S., JANG, K., PARK, K., et al. "PacketShader: a GPU-accelerated software router", *ACM SIGCOMM Comput. Commun. Rev.*, pp. 195–206, 2010.

[39] WOOD, T., SHENOY, P., VENKATARAMANI, A., et al. "Sandpiper: Black-box and Gray-box Resource Management for Virtual Machines". In: *USENIX NSDI*, 2007.

[40] BARROSO, L., HOLZLE, U. "The Case for Energy-Proportional Computing", *IEEE Computer*, v. 40, n. 12, pp. 33–37, 2007.

[41] NIYATO, D., CHAISIRI, S., LEE BU SUNG, S., et al. "Optimal Power Management for Server Farm to Support Green Computing". In: *IEEE/ACM CCGRID*, pp. 84–91, 2009.

[42] *Power Management Virtualization Technology for a Green IT.* http://www.vmware.com/technical-resources/advantages/power-saving.html, Acessado em janeiro de 2011.

[43] PERVILÄ, M., KANGASHARJU, J. "Running Servers around Zero Degrees". In: *ACM SIGCOMM Workshop on Green Networking*, pp. 9–14, 2010.

[44] FISHER, W., SUCHARA, M., REXFORD, J. "Greening Backbone Networks: Reducing Energy Consumption by Shutting Off Cables in Bundled Links". In: *ACM SIGCOMM Workshop on Green Networking*, pp. 29–34, 2010.

[45] BOLLA, R., BRUSCHI, R., DAVOLI, F., et al. "Energy-aware performance optimization for next-generation green network equipment". In: *ACM SIGCOMM PRESTO*, pp. 49–54. ACM, 2009.

[46] AMD DAN LI AMD MINGWEI XU, Y. S. "Energy-aware Routing in Data Center Network". In: *ACM SIGCOMM Workshop on Green Networking*, pp. 1–7, 2010.

[47] KRIOUKOV, A., MOHAN, P., ALSPAUGH, S., et al. "NapSAC: Design and Implementation of a Power-Proportional Web Cluster". In: *ACM SIGCOMM Workshop on Green Networking*, pp. 15–21, 2010.

[48] *Dell PowerEdge R710 Energy Star Datasheet.* http://www.dell.com/downloads/global/products/pedge/en/PowerEdge_-R710_16GB_DIMMs870W_Energy_Star_DataSheet.pdf, Acessado em janeiro de 2011.

[49] *Intel Xeon Processor 5500 Series Datasheet.* http://www.intel.com/Assets/PDF/datasheet/321321.pdf, Acessado em janeiro de 2011.

[50] *Intel PRO/1000 PT Quad Port Server Adapter Product Brief.* http://www.intel.com/assets/pdf/prodbrief/314100.pdf, Acessado em janeiro de 2011.

[51] LEFEVRE, L., ORGERIE, A.-C. "Designing and evaluating an energy efficient Cloud", *J. Supercomput.*, v. 51, n. 3, pp. 352–373, 2010.

[52] VASIC, N., BARISITS, M., SALZGEBER, V., et al. "Making cluster applications energy-aware". In: *ACM ACDC*, pp. 37–42, 2009.

[53] WANG, Y., KELLER, E., BISKEBORN, B., et al. "Virtual routers on the move: live router migration as a network-management primitive", *ACM SIGCOMM Comput. Commun. Rev.*, pp. 231–242, 2008.

[54] CHENG, Y., ZHUANG, W. "Dynamic inter-SLA resource sharing in path-oriented differentiated services networks", *IEEE/ACM Trans. Netw.*, pp. 657–670, 2006.

[55] *RFC 2474 - Definition of the Differentiated Services Field (DS Field) in the IPv4 and IPv6 Headers.* http://tools.ietf.org/html/rfc2474, Acessado em janeiro de 2011.

[56] *RFC 2475 - An Architecture for Differentiated Services.* http://tools.ietf.org/html/rfc2475, Acessado em janeiro de 2011.

[57] YIN, Y., POO, G.-S. "User-oriented hierarchical bandwidth scheduling for ethernet passive optical networks", *Comput. Commun.*, pp. 965–975, 2010.

[58] CHEN, X., JUKAN, A., DRUMMOND, A. C., et al. "A multipath routing mechanism in optical networks with extremely high bandwidth requests". In: *IEEE GLOBECOM*, pp. 2065–2070, 2009.

[59] KIM, J., DALLY, W. J., ABTS, D. "Flattened butterfly: a cost-efficient topology for high-radix networks". ISCA, pp. 126–137. ACM, 2007.

[60] *Python Programming Language - Official Website.* http://www.python.org/, Acessado em janeiro de 2011.

[61] *The Cooperative Association for Internet Data Analysis.* http://www.caida.org/home/, Acessado em janeiro de 2011.

[62] *Equinix Inc. Official Website.* http://www.equinix.com/, Acessado em janeiro de 2011.

[63] *Manpage of TCPDUMP.* http://www.tcpdump.org/tcpdump_man.html, Acessado em janeiro de 2011.

[64] KELLER, E., GREEN, E. "Virtualizing the data plane through source code merging". In: *ACM SIGCOMM PRESTO*, pp. 9–14, 2008.

[65] KIM, H., CLAFFY, K., FOMENKOV, M., et al. "Internet traffic classification demystified: myths, caveats, and the best practices". In: *ACM CoNEXT*, dez. 2008.

[66] *RFC 3954 - Cisco Systems NetFlow Services Export Version 9.* http://tools.ietf.org/html/rfc3954, Acessado em janeiro de 2011.

[67] *Estatísticas de tráfego da rede Ipê.* http://www.rnp.br/ceo/trafego/, Acessado em fevereiro de 2011.

[68] BEHESHTI, N., GANJALI, Y., GHOBADI, M., et al. "Experimental study of router buffer sizing". In: *ACM SIGCOMM IMC*, pp. 197–210, 2008.

[69] WALSWORTH, C., ABEN, E., CLAFFY, K., et al. *The CAIDA Anonymized 2009 Internet Traces - 09/17/2009.* http://www.caida.org/data/passive/passive_2009_dataset.xml, Acessado em janeiro de 2011.

[70] PIRMEZ, L., JR., J. C. C., DELICATO, F. C., et al. "SUTIL - Network selection based on utility function and integer linear programming", *Comput. Netw.*, pp. 2117–2136, 2010.

[71] FERNANDES, N. C., DUARTE, O. C. M. B. "XNetMon: Uma Arquitetura com Segurança para Redes Virtuais", *SBSeg'2010*, pp. 339–352, 2010.