



Programa Interdisciplinar de Pós-Graduação em

Computação Aplicada

Mestrado Acadêmico

Clebio Dossa

CoreLB: uma proposta de balanceamento de carga na rede
com OpenFlow e SNMP

São Leopoldo, 2016

Clebio Dossa

CORELB: UMA PROPOSTA DE BALANCEAMENTO DE CARGA NA REDE COM
OPENFLOW E SNMP

Dissertação apresentada como requisito parcial
para a obtenção do título de Mestre pelo
Programa de Pós-Graduação em Computação
Aplicada da Universidade do Vale do Rio dos
Sinos — UNISINOS

Orientador:
Prof. Dr. Rodrigo Righi

São Leopoldo
2016

D724c

Dossa, Clebio

CoreLB : uma proposta de balanceamento de carga na rede com Openflow e SNMP / por Clebio Dossa. – 2016.

78 f.: il. ; 30 cm.

Dissertação (mestrado) — Universidade do Vale do Rio dos Sinos, Programa de Pós-graduação em Computação Aplicada, São Leopoldo, RS, 2016.

“Orientação: Prof. Dr. Rodrigo Righi.”

1. OpenFlow (Protocolo de redes de computadores). 2. SNMP.
3. Balanceamento de carga. 4. Redes de computadores – Gerência;
5. SDN (Tecnologia de redes de computadores). Título.

CDU: 004.7

Clébio Dossa

CoreLB: uma proposta de balanceamento de carga na rede com OpenFlow e SNMP

Dissertação apresentada à Universidade do Vale do Rio dos Sinos – Unisinos, como requisito parcial para obtenção do título de Mestre em Computação Aplicada.

Aprovado em 18 Agosto 2016

BANCA EXAMINADORA

Prof. Dr. Lucas Mello Schnorr – UFRGS

Prof. Ms. Diego Luís Kreutz – University of Luxembourg

Prof. Dr. Cristiano Andre da Costa – UNISINOS

Prof. Dr. Rodrigo da Rosa Righi

Visto e permitida a impressão
São Leopoldo,

Prof. Dr. Sandro José Rigo

Coordenador PPG em Computação Aplicada

(Esta folha serve somente para guardar o lugar da verdadeira folha de aprovação, que é obtida após a defesa do trabalho. Este item é obrigatório, exceto no caso de TCCs.)

RESUMO

Atualmente, muitos serviços distribuem a carga entre diversos nós computacionais direcionando as conexões com alguma estratégia de balanceamento para divisão da carga. O advento do uso de redes definidas por software (SDN) está mudando paradigmas da administração de redes, absorvendo serviços especializados, automatizando processos e gerando inteligência para regras estáticas com uma grande variedade de opções de implementação. O balanceamento de carga é um dos serviços especializados que pode usufruir dos conceitos de SDN, sem definições e processos estáticos como ocorre muitas vezes nos atuais modelos usados de balanceamento de carga. A definição dos protocolos que suportam SDN usualmente permitem soluções alternativas e eficientes para este problema, desta forma, neste trabalho, é apresentada uma proposta de metodologia para balanceamento de carga entre distintos servidores de um *pool* com a troca do destino de tráfego realizada pela rede. Esta solução é chamada *Core-based load balance* (CoreLB), pois o serviço especializado de balanceamento de carga é realizado pela rede onde a administração de pacotes é nativamente realizada. A metodologia faz uso do protocolo SNMP para análise de recursos dos servidores com o objetivo de avaliar a situação de carga de cada nó computacional e de estatísticas de consumo de rede através do protocolo OpenFlow. Este trabalho avaliou o balanceamento de carga em serviços Web e a união de estatísticas de rede e da carga dos servidores, para a tomada de decisão de balanceamento, mostra-se uma metodologia eficiente e com melhores tempos de resposta ao usuário comparado com outras metodologias de avaliadas. Também melhorou a distribuição de consumo de recursos entre os servidores.

Palavras-chave: OpenFlow. SNMP. Balanceamento de carga. Redes. Administração de redes de computadores. Carga de rede. Distribuição computacional. Arquitetura SDN. Roteamento de pacotes. Plano de controle.

ABSTRACT

Currently, most services balance the load between distinct hosts forwarding connections with a load balance strategy in front. Usually, a dedicated appliance is responsible to perform the balance and may be a fault point and become expensive. The new concepts of computer network architecture with Software-Defined Networking (SDN) are changing the network management, absorbing specialist services, automating process and building intelligence to static rules with loads of delivery options. The load balance is a specialized service that can enjoy in a positive way of SDN concepts, with low costs, in a flexible way as per the process needs instead of a plastered process definitions that occurs in many actual models. The OpenFlow protocol definition allow us to use a new solution to address this issue. This work shows a load balance purpose between distinct hosts with the destination change of connections made by the network core. It calls *Core-based load balance* (CoreLB) because the specialized load balance service move to the network core where the package forwarding is naturally made. This solution intend to use the SNMP protocol to analyse the hosts resources to evaluate server's load. Using the network forwarding statistics and OS load informations, an efficient solution of load balance, the methodology proved to be efficient with better users' response times average of 19% than no balanced scenario as well as around 9% better than others load balance strategies and a properly balance consumption of resources from hosts side. This process can be inhered in distinct models, however, this research intend to evaluate Web Services.

Keywords: OpenFlow. SNMP. Load Balance. Network. computer network management. network loads. Distributing computin. resource distribution. SDN architecture. packet forwarding. control plane. packet switching.

LISTA DE FIGURAS

Figura 1:	Modelo atual de balanceamento de carga e modelo proposto	18
Figura 2:	Modelo SDN	24
Figura 3:	Topologia OpenvSwitch	27
Figura 4:	Execução do Openload	28
Figura 5:	Topologia da solução proposta representando os agentes e atores com suas comunicações	40
Figura 6:	Diagrama de sequência dos agentes SNMP e OF	42
Figura 7:	Diagrama de sequencia dos agente LB	43
Figura 8:	Aplicações Web disponíveis em cada servidor	45
Figura 9:	Topologia virtualziada com Xen Server	46
Figura 10:	Algoritmo TimeKeeper para definir a qualidade inicial	49
Figura 11:	TimeKeeper diagrama	50
Figura 12:	Agente OF diagrama	51
Figura 13:	Agente SNMP diagrama	52
Figura 14:	Algoritmo agente LB	54
Figura 15:	Exemplo do cálculo de pontos	55
Figura 16:	Consumidores de CPU	58
Figura 17:	Consumidores de Memória	59
Figura 18:	Consumidores de Rede	60
Figura 19:	Consumidores do serviço Web	61
Figura 20:	Algoritmo Concentrador	62
Figura 21:	Algoritmo Time Slice Based Choice	62
Figura 22:	Algoritmo Random Time Slice and Choice	63
Figura 23:	Comportamento de um nó computacional recebendo toda a carga gerada pelo OpenLoad	64
Figura 24:	Monitoramento LOAD durante o uso do método Random Time Slice and Choice	71
Figura 25:	Monitoramento LOAD durante o uso do método Time Slice Based Choice	72
Figura 26:	Monitoramento LOAD durante o uso do método CoreLB	73

LISTA DE TABELAS

Tabela 1:	Legenda de trabalhos estudados	34
Tabela 2:	Análise comparativa de itens considerados	35
Tabela 3:	Análise comparativa de estratégias implementadas	35
Tabela 4:	Dados analisados para determinar a qualidade de cada nó computacional . .	48
Tabela 5:	Tipo de dados em cada recurso	53
Tabela 6:	Número de testes realizados com o OpenLoad e configuração de clientes . .	64
Tabela 7:	Resultados de qualidade de nós computacionais	66
Tabela 8:	Resultados da avaliação dos métodos	68
Tabela 9:	Comparativo do valor de consumo médio	69
Tabela 10:	Comparativo do valor de consumo máximo	70

LISTA DE SIGLAS

ATM	Asynchronous Transfer Mode
BGP	Border Gateway Protocol
DNS	Domain Name System
ECMP	Equal Cost Multipath
GNU	GNU is Not Unix
IRQ	Interrupt Request
ITU	International Telecommunication Union
MIB	Management Information Base
MRTG	Multi Router Traffic Grapher
NAT	Network Address Translation
OID	Object Identifier
ONF	Open Network Foundation
SDN	Software-Defined Networking
SNMP	Simple Network Management Protocol

SUMÁRIO

1 INTRODUÇÃO	17
1.1 Questão de Pesquisa	18
1.2 Objetivo	19
1.3 Organização do Trabalho	20
2 FUNDAMENTAÇÃO TEÓRICA	21
2.1 Balanceamento de Carga	21
2.2 Os Protocolos e Serviços Utilizados	23
2.2.1 Software-Defined Networking	23
2.2.2 A ONF e o Modelo OpenFlow	24
2.2.3 Protocolo SNMP	25
2.3 Ferramentas para Implementação e Execução do Protótipo	25
2.3.1 RRDTool	26
2.3.2 OpenvSwitch	26
2.3.3 Openload	27
3 TRABALHOS RELACIONADOS	29
3.1 Balanceamento de Carga entre Servidores	29
3.1.1 Modelo de balanceamento de carga dinâmico com o uso do OpenFlow e SNMP	29
3.1.2 Avaliação de técnicas de balanceamento de carga com o uso do OpenFlow	30
3.1.3 Comparativo de técnicas de balanceamento de carga em OpenFlow com o uso do Openload	30
3.1.4 O uso de reserva de recursos para balanceamento de carga de serviços Web	31
3.1.5 Implementação de um balanceamento de carga dinâmico com OpenFlow	32
3.2 Balanceamento de Carga em Redes de Computadores	32
3.2.1 Balanceamento de carga baseado em fatias de fluxos usando OpenFlow	32
3.2.2 Balanceamento de carga em redes de data center com arquiteturas Folded-Clos	33
3.3 Análise Comparativa	33
4 MODELO PROPOSTO	37
4.1 Decisões do Projeto	37
4.2 Arquitetura	38
4.2.1 Atores e Agentes	38
4.3 Objetivo da Arquitetura Proposta	41
4.4 Modelo de Comunicação Proposto	42
5 IMPLEMENTAÇÃO	45
5.1 Funcionamento do <i>CoreLB</i>	46
5.1.1 TimeKeeper	49
5.1.2 Agente OF	51
5.1.3 Agente SNMP	52
5.1.4 Agente LB	53
5.2 Sistema de Pontuação	55

6 RESULTADOS	57
6.1 Metodologia de Avaliação	57
6.1.1 Consumidores	57
6.1.2 Técnicas de balanceamento comparadas	61
6.1.3 Gerador de carga	63
6.2 Avaliação dos Resultados	65
6.2.1 Pontuação definida pelo CoreLB algoritmo de qualidade	65
6.2.2 Resultados estatísticos do OpenLoad	67
6.2.3 Resultados de <i>Load Average</i>	68
7 CONCLUSÃO	75
7.1 Contribuições Realizadas	76
7.2 Trabalhos futuros	76
REFERÊNCIAS	77

1 INTRODUÇÃO

Desempenho e alta disponibilidade são itens críticos para *data centers* atuais. A infraestrutura de rede necessita maximizar a vazão, diminuir a latência e permitir uma elasticidade natural para atender as atuais demandas de aplicações, contudo, processos e equipamentos não podem ser rígidos e engessados. Neste cenário, mecanismos de balanceamento de carga realizam um importante trabalho distribuindo acessos concorrentes ou adequando aos limites da capacidade. Muitos métodos e fabricantes propõem soluções complexas, difíceis de serem implementadas e testadas; rígidas e caras, que necessitam de hardware potente para armazenar milhares de fluxos de comunicação (WANG; LAN; CHEN, 2014). Além disso, o uso de equipamento dedicado para balanceamento de carga adiciona latência ao processamento, pode ser um gargalo ou mesmo um ponto adicional de falha. Outro fator relevante é a rede que é um ponto essencial na comunicação, e interliga nós primários, oferece serviços de troca de rotas entre redes e a movimentação de pacotes (TECHOPEDIA, 2015).

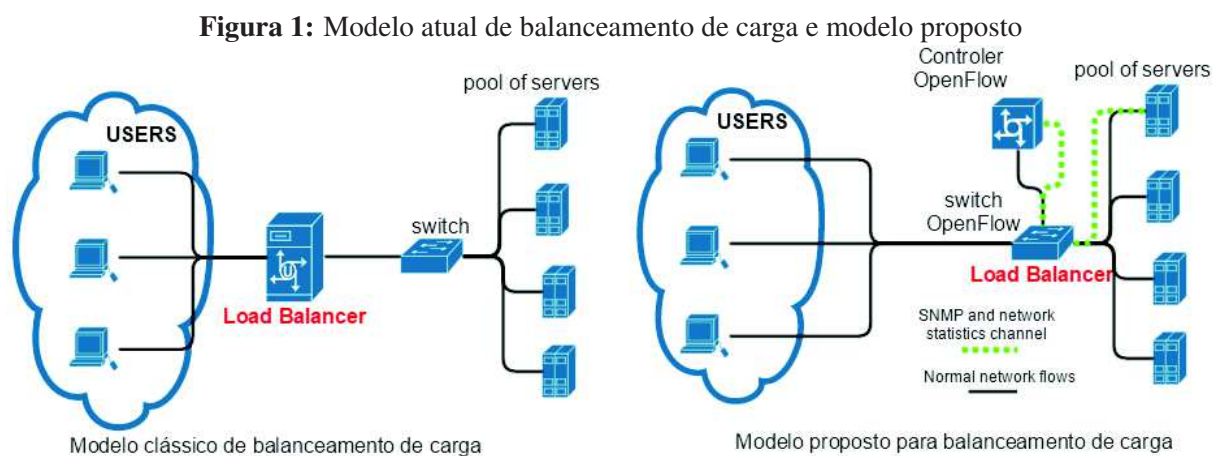
Com a possibilidade de delegar a tarefa de balanceamento de carga à rede, onde equipamentos específicos já realizam a administração dos fluxos de conexão, é possível reduzir a necessidade de adição de equipamentos dedicados para esta tarefa. Neste contexto, redes programáveis permitem a manutenção de regras de fluxos e fornecem estatísticas para tomada de decisão. O protocolo OpenFlow proporciona o acesso a um modelo que pode ser explorado para realização da tarefa de balanceamento de carga. Neste sentido, o OpenFlow permite a criação de técnicas distintas para o balanceamento de carga com flexibilidade, administrando cada fluxo da comunicação e proporcionando a habilidade de controlar cada *host* a qualquer momento de maneira centralizada (MARCON D., 2011).

Existe uma série de técnicas para implementações de balanceamento de carga, o protocolo OpenFlow tem sido usado para realizar este serviço, entretanto há uma lacuna na decisão de balanceamento realizada pelo OpenFlow e análise do uso de recursos dos servidores do *pool* de conexão. O OpenFlow tem uma visão dos recursos de rede, dos fluxos e dados transferidos, porém, não avalia o estado computacional dos servidores de destino. Algumas pesquisas realizadas demonstram o uso do OpenFlow usando trocas de regras de maneiras distintas para realizar o balanceamento de carga, mas não tem uma visão do nó computacional e de sua atual situação. Este trabalho realiza uma pesquisa do impacto do uso do protocolo OpenFlow atuando como balanceador de carga e avalia das vantagens do uso da rede para esta tarefa. Esta análise propõe a criação de uma metodologia com o uso do OpenFlow em conjunto com o protocolo SNMP, proporcionando a execução de balanceamento de carga adaptável e equilibrando à carga de maneira eficaz. Com a visão de distintos pontos: fluxos de comunicação de todos os servidores do *pool* de balanceamento através do OpenFlow, e da capacidade de recursos de cada servidor através do SNMP, é possível ter a visão da situação do ambiente.

Apesar da grande diversidade de algoritmos de balanceamento de carga existentes, a análise de decisão com informações baseadas na rede não é comum de ser utilizada para este propósito

(BELYAEV; GAIVORONSKI, 2014). Com isso, este trabalho realiza uma pesquisa e avaliação de uma metodologia que une informações de tráfego de rede com consumo de recursos dos servidores, para tomada de decisão no balanceamento de carga. Em adição à união do protocolo SNMP com o uso do OpenFlow tem-se também a simplificação da estrutura de balanceamento de carga com o uso dos conceitos e inovação de redes definidas por software que permitem uma melhor automatização dos processos.

A Figura 1 ilustra o modelo atual utilizado para balanceamento de carga, onde o equipamento o centralizado em frente aos servidores concentra as conexões recebidas e redireciona para o destino desejado. Também existe a representação do modelo proposto, no qual o balanceamento é realizado pela rede.



A seguir, está apresentada a questão de pesquisa deste trabalho e o objetivo, onde pontos específicos que são estudados estão apresentados juntamente com o modelo projetado para ser estudado. Por fim, e a organização de todo o restante desta proposta está apresentada na última seção deste capítulo.

1.1 Questão de Pesquisa

A chave para o bom desempenho em serviços de software, como serviços Web, é a eficiência de um bom mecanismo de distribuição de carga usado para distribuir as requisições de clientes entre os servidores. Segundo (SEHERY; CLANCY, 2015), atingir um balanceamento de carga ideal para demandas de tráfego desconhecidas, é difícil por não se saber qual será a demanda futura. Um balanceamento de carga eficiente permite a criação de soluções escaláveis fazendo um bom uso dos recursos disponíveis. Desta forma, o serviço de balanceamento de carga deve ser capaz de controlar situações de sobrecarga (*overload*), limitando o acesso aos servidores quando o processamento está além da sua capacidade. Esta característica garante a disponibilidade e desempenho das requisições que já estão na fila de processamento.

A questão de pesquisa proposta neste trabalho está descrita abaixo:

"Como desenvolver um modelo de balanceamento de carga eficiente, considerando a situação de carga dos servidores e o estado da rede."

Para contextualizar, entende-se o termo *eficiente* como sendo o melhor equilíbrio do ponto de vista do uso de recursos computacionais e o melhor desempenho do ponto de vista do usuário que acessa o serviço computacional. A *situação de carga dos servidores* são os recursos de hardware e software de cada servidor.

1.2 Objetivo

Existem soluções de balanceamento de carga conhecidas como *client-based load balance* e também *server-based load balance*. Na primeira, a decisão do destino é realizada pelo cliente, e a segunda com a intervenção de algum serviço no meio do caminho de comunicação, para tomar a decisão de balanceamento. Ambas possuem pontos fracos - a solução baseada em cliente necessita de intervenção nos clientes como instalação de softwares ou adição de comunicação entre clientes; a solução baseada em servidor necessita da adição de processamento no meio da comunicação criando latência ao processo. A simplificação deste tipo de ambiente é necessária para que o processo de balanceamento de carga seja eficiente, já que ambientes complexos criam regras e algoritmos complexos que causam o incremento do custo computacional nas ações.

A solução *server-based load balance* tem como qualidade a facilidade para usar informações sobre o estado dos servidores. Embora muitas destas soluções precisem da comunicação entre os servidores para troca de informação de estado, esta comunicação é muito mais simples do que a intervenção em clientes que podem ser distintos e de maior variedade. As informações de capacidade e uso de recursos computacionais são dados valiosos para garantir o bom desempenho no processamento de uma requisição. Desta forma, este item é considerado para elaboração do mecanismo de balanceamento de carga apresentado neste trabalho.

CoreLB é uma solução que executa a distribuição da carga baseado nas informações de fluxos de rede, bytes transferidos e também uso dos recursos dos servidores com o uso do protocolo SNMP. O Objetivo do protocolo SNMP é usufruir dos mesmos dados que *server-based load balance* tem como vantagem para tomada de decisão, que são as informações de uso dos recursos de cada servidor. **Este trabalho elabora uma metodologia que usufrui de distintas informações de ambos os aspectos de forma equilibrada para garantir a eficiência do balanceamento de carga.** Os objetivos desta pesquisa, de forma específica, podem ser descritos nos itens abaixo:

- Simular um ambiente para balanceamento de carga com suporte ao OpenFlow usando serviços Web como destino;
- Analisar o impacto do uso do OpenFlow para realização de balanceamento de carga;
- Criar um protótipo com um modelo onde o balanceamento de carga é definido com base

na análise de estatísticas de rede e de recursos dos servidores com o auxílio do protocolo SNMP;

- Testar o protótipo em diferentes situações de carga com diferentes tipos de requisições;
- Definir a metodologia de decisão de balanceamento baseado nas estatísticas do consumo de recursos computacionais, atribuindo importâncias diferentes para cada variável analisada com o intuito de atingir o balanceamento de carga eficiente;
- Definir técnicas de monitoramento adaptáveis e inteligentes com capacidade de se modelar em diferentes situações de carga e que não onere os serviços sendo capaz de representar a realidade do que está acontecendo no ambiente;

1.3 Organização do Trabalho

O restante desse trabalho está dividido em fundamentação teórica - capítulo 2 - na qual as técnicas de balanceamento de carga, protocolos, padrões e ferramentas são abordadas. A explanação sobre redes de computadores consiste no uso do conceito de SDN e o protocolo OpenFlow, seguido da definição do protocolo SNMP. Após, são explicadas algumas ferramentas necessárias para elaboração do modelo proposto.

O capítulo 3 analisa trabalhos relacionados ao uso do protocolo OpenFlow para balanceamento de carga e técnicas que utilizam análise de dados de redes para realizar o balanceamento. Também foram considerados trabalhos que avaliam e testam algoritmos clássicos de balanceamento de carga com o uso do OpenFlow, e trabalhos que propõem o uso de algoritmos associados com monitoramento de recursos.

O capítulo 4 contém o modelo proposto pelo *CoreLB* e toda sua metodologia funcional juntamente com a arquitetura desenvolvida no protótipo para análise e estudo da proposta. O capítulo 4 limita-se a implementação do protótipo e o capítulo 6 é a sequencia dos resultados obtidos com os testes do protótipo apresentado no capítulo 5.

O último capítulo é a conclusão com o cronograma e itens de pesquisa que estão no escopo deste trabalho. Este capítulo é dedicado ao modelo desenvolvido que demonstra como está a divisão do projeto com sua arquitetura e metodologia que será usada para avaliação e análise de resultados.

2 FUNDAMENTAÇÃO TEÓRICA

O objetivo desse capítulo é apresentar os conceitos e ferramentas utilizadas para a fundamentação da pesquisa elaborada e o modelo proposto que será exposto nos capítulos posteriores. A seção dedicada à explicação sobre balanceamento de carga apresenta os modelos mais tradicionais utilizados para esta tarefa, e é discutida a razão pela qual este tipo de técnica pode ser falha no contexto desejado. As seções que seguem ressaltam técnicas, ferramentas e itens envolvidos com a pesquisa que é realizada. Não se pretende exaurir o tema de maneira abrangente, apenas nos detalhes que se relacionam com esta pesquisa.

2.1 Balanceamento de Carga

Balanceamento de carga é uma técnica bastante comum e executada de maneira distinta, como por exemplo, através do uso do protocolo DNS que consiste na configuração do mesmo nome para distintos IPs. O cliente que solicita o nome ao serviço DNS, irá receber um dos endereços configurados usando a técnica *Round Robin*. Balanceamento de carga com DNS é simples, porém, não considera a carga dos servidores. Além do mais, a grande desvantagem do balanceamento usando DNS é a falta de controle do cache de DNS de cada cliente. Desta forma, o cliente pode não realizar a solicitação ao serviço de DNS para resolução de nomes, mas realizar uma resolução local baseada em uma consulta prévia (NAKAI; MADEIRA; BUZATO, 2015).

Existem também soluções em que o cliente é responsável por realizar o balanceamento de carga entre os servidores, porém, há a complexa necessidade de cada um dos clientes consultar/receber o estado de cada servidor para saber quem tem a melhor capacidade de processamento. O maior problema desta solução é a necessidade de instalação de uma forma de aquisição da informação de carga de cada servidor nos clientes. (SHANG et al., 2013).

O uso de *Reverse Proxy* é outra técnica que também pode ser empregada para o propósito de balanceamento de carga onde este *proxy* é responsável pela distribuição das conexões entre os servidores do *pool*. A técnica usufrui de algumas vantagens como *cache* de informações, que efetivamente reduz o tempo de processamento, porém, necessita de desenvolvimento de módulos específicos para cada serviço atendido, e não deixa de ser um equipamento adicional no processamento da requisição.

Os recursos de *Network Address Translation* (NAT) também são comumente usados para balanceamento de carga. A tradução de endereços realizada por *gateway* pode mapear endereços públicos para diversos endereços privados, fornecendo um balanceamento de conexões entre os endereços privados, muito semelhante ao uso de *Reverse Proxy*, de forma que, tem os mesmos pontos negativos em sua elaboração. Outra desvantagem do uso de NAT é a inexistência do recurso de *cache* que possibilita um recuperação mais rápido de algumas informações.

Todos estes primeiros exemplos podem usar algoritmos distintos como: *Round Robin*, em

que as requisições são divididas entre todos os servidores e de maneira circular; *Randomized* em que as requisições são repassadas aos servidores de forma randômica; *Load based* e uma nova requisição é enviada ao servidor com menor carga. Os algoritmos de *Round Robin* e *Randomized* funcionam muito bem quando o número de requisições é maior que o número de servidores e as requisições são muito semelhantes em termos de consumo computacional. Os algoritmos de *Load based* requerem constante atualização da informação da carga dos servidores, o que pode ser bastante custoso e gerar excesso de comunicação dependendo de como a implementação é realizada.

Existem técnicas e algoritmos adicionais aos apresentados para o propósito de balanceamento de carga que funcionam, geralmente, de maneira mais especializada com a aplicação consumida, porém, seguindo a ideia de algum destes algoritmos apresentados considerados como "base primária" de técnicas de balanceamento de carga. De uma forma mais simplista, (RAGALATHA P., 2013) define estas técnicas como sendo dois principais mecanismos de balanceamento: *Static Load Balancing* e *Dynamic Load Balancing*.

Embora muitos sejam os métodos de balanceamento de carga que tem sido propostos na literatura atual, os métodos tradicionais não satisfazem as necessidades de redes de *data centers* (WANG; LAN; CHEN, 2014). O uso dos recursos e a utilização da rede são itens que podem ser considerados para a decisão de balanceamento de carga. Por outro lado, a falta de maneiras eficientes de obter estatísticas dos comutadores de rede envolvido no processo de comunicação dificulta este processo. Para suprir esta necessidade, o modelo de redes definidas por software (SDN) proporciona o monitoramento de tráfego e informações através de um controlador, que age de forma centralizada. Com o advento desta nova metodologia de administração de redes, informações antes não consideradas no balanceamento de carga para servidores podem agora ser adicionadas às decisões dos métodos tradicionais de balanceamento, de forma mais simples e sem necessidade de distintos algoritmos e soluções proprietárias para a coleta das informações.

O uso de informações de tráfego de rede são comuns em balanceamento de tráfego entre *links* de rede e existe uma variedade de eficientes técnicas para o processo de balanceamento de tráfego em *links*. Estas técnicas também podem ser usadas para balanceamento de requisições destinadas a um serviço específico. Atualmente, muitas das redes de *data centers* entregam balanceamento das requisições em canais baseados em *Equal Cost Multipath* (ECMP), para distribuir o tráfego uniformemente entre distintos caminhos. O ECMP utiliza funções de *hash* aplicadas aos cabeçalhos dos fluxos para direcionar um fluxo à uma rota com a menor utilização (SEHERY; CLANCY, 2015). Assim sendo, estas técnicas clássicas, largamente utilizadas em balanceamento de tráfego entre links, podem ser associadas às necessidades para balanceamento de requisições para um serviço, conjuntamente com os métodos já utilizados em *Load based*, para atingir o modelo *Dynamic Load Balancing*. A união destas duas informações pode atingir um balanceamento mais eficaz garantindo o melhor desempenho do acesso aos serviços, e melhora da distribuição do uso de recursos computacionais.

2.2 Os Protocolos e Serviços Utilizados

Esta seção é destinada à fundamentação dos protocolos e serviços em administração de redes e monitoramento de recursos necessários para a elaboração do protótipo proposto. Para definição do projeto, existe a necessidade do monitoramento de recursos de rede e de recursos dos servidores. Para contemplar a visão de redes, está definido o uso do protocolo OpenFlow e para contemplar a visão de servidores, o protocolo SNMP. Inicialmente, existe a definição de SDN que é necessária para introduzir o protocolo OpenFlow.

2.2.1 Software-Defined Networking

Software-Defined Networking é um modelo em redes de computadores que permite a administração e manutenção de serviços de rede através de uma abstração com funções de alto nível. A principal ideia de SDN está em torno da automação de processos e procedimentos em redes com o intuito de otimizar e padronizar tarefas associadas à administração de equipamentos de rede. As primeiras iniciativas do conceito de SDN datam de 1980s e 1990s (KREUTZ et al., 2015). Conceitos inicialmente implementados para manutenção da rede de telefonia. Com uma proposta de melhorar a manutenção e eficiências destas redes. Outra iniciativas, neste mesmo período, queriam melhorar a manutenção de ATM, Ethernet, BGP ou mesmo MPLS. Atualmente, empresas renomadas como Cisco ¹ e Juniper Networks ² exploram o uso deste modelo de forma mais consolidada (VAUGHAN-NICHOLS, 2011).

De acordo com a ONF, SEND é uma arquitetura emergente que separa as funções de *control* e de *forwarding* para habilitar a programação das funções de controle (JARRAYA; MADI; DEBBABI, 2014). Nesta arquitetura, os comutadores se tornam equipamentos de repasse, que tomam ações baseadas nas regras geradas pelo controlador. Outro aspecto deste modelo é a definição de bibliotecas para o plano de controle e plano de dados com uma interface de software programável, que permite o controlador administrar o plano de dados do comutador.

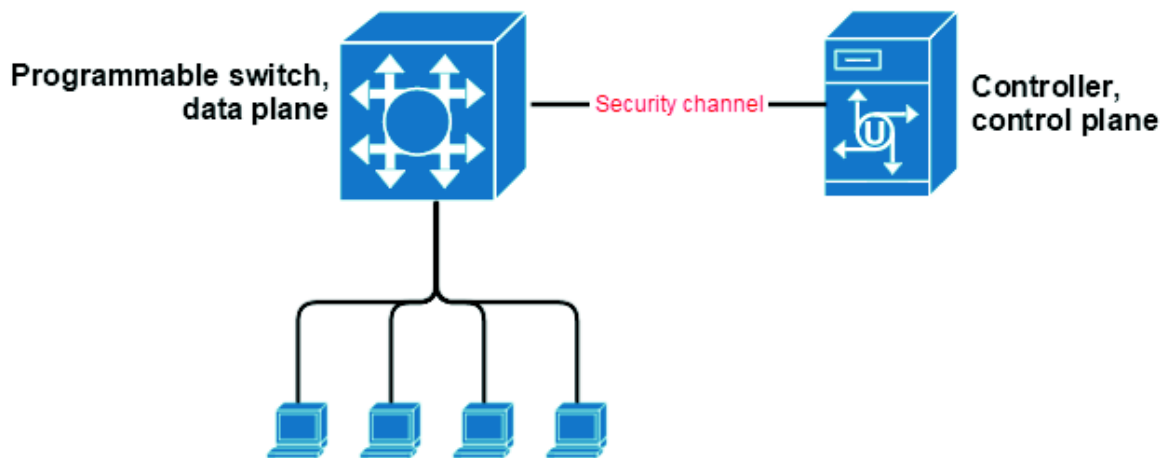
Clássicos roteadores ou *switches* executam a tarefa de *data plane* (plano de dados) e *control plane* (plano de controle) no mesmo equipamento com definições e características específicas de cada fornecedor. O plano de dados, onde a transmissão dos pacotes é realizada, e o plano de controle, onde as decisões de como este pacote será administrado são realizados. O plano de controle define as decisões adicionando as regras de transmissão de pacotes no plano de dados, conforme figura 2. Para a implementação desta arquitetura, sugere-se um canal dedicado de comunicação entre o comutador e o controlador para que esta comunicação não seja interferida e nem interfira o tráfego externo. Além deste ponto, sugere-se um canal dedicado por questões de segurança, mesmo existindo criptografia na troca de mensagens.

(THEINSTITUTE.IEEE.ORG, 2014)

¹<http://www.cisco.com>

²<http://www.juniper.net/>

Figura 2: Modelo SDN



A tecnologia SDN será essencial para a alteração da natureza das redes do futuro, com vantagens na economia de equipamentos e custos. Ela irá definir operações e a forma para criar programas flexíveis e redes dinâmicas capazes de interagir e monitorar terminais, máquinas inteligentes e outros equipamentos além de controlar robôs para suportar serviços inovadores. O advento de SDNs irá afetar cada porção de nossas vidas.

Diferentes fabricantes de comutadores criam protocolos para definição da comunicação entre o comutador e o controlador usando técnicas e modelos proprietários.

2.2.2 A ONF e o Modelo OpenFlow

A abordagem defendida pela *Open Network Foundation* (ONF) requer a implementação de roteadores ou switches que suportam o protocolo OpenFlow e da mesma forma controladores SDN para realizar a manutenção de fluxos de rede destes equipamentos. (MCKEOWN et al., 2008) foi um trabalho publicado por pesquisadores da Universidade de Stanford em 2 Abril de 2008 na SIGCOMM ³. Esta publicação deu origem ao OpenFlow como sendo uma forma para pesquisadores executarem protocolos experimentares nas redes de computadores utilizadas diariamente. A ONF se tornou uma organização essencial para a fundamentação e padronização de SDN. De acordo com (ONF, 2015), SDN é uma arquitetura dinâmica, maleável, eficiente e adaptável, tornando ideal para ambientes de alta performance e dinâmicos, como a natureza das aplicações atuais. O OpenFlow é um protocolo definido pela ONF que tornou-se um elemento fundamental para soluções SDN.

Nos dias atuais esta tendência não está presente apenas em universidades, mas também grandes nomes da indústria como Cisco, Oracle, Microsoft, Google, Juniper entre outros que são membros da ONF. O uso do OpenFlow tem sido explorado em diversos aspectos. Muitos

³<http://www.sigcomm.org/>

modelos e técnicas clássicas de administração e implementação de redes estão agora sendo implementadas com o uso do OpenFlow. O balanceamento de carga realizado pela rede, onde o OpenFlow atua, diminui a necessidade de adição de equipamentos para realização desta tarefa. Desta forma, é um protocolo perfeito para atender **um dos objetivos desta pesquisa que é a simplificação do ambiente de balanceamento de carga.**

2.2.3 Protocolo SNMP

O *Simple Network Management Protocol* (SNMP) é padrão para monitorar e administrar equipamentos usando a camada de aplicação, em conjunto com o protocolo UDP, enviando requisições para o equipamento alvo. Cada alvo administrado precisa manter suas informações em uma estrutura de árvore chamada *Management Information Base* (MIB) e conservar esta estrutura atualizada para que o cliente SNMP consiga consultar este determinado objeto no equipamento remoto ou mesmo realizar alguma alteração no objeto. Com uma estrutura hierárquica de MIBs bem definida, o cliente sabendo a MIB desejada é possível administrar o equipamento alvo remotamente (ZELTSERMAN, 1999).

A MIB é identificada por um *Object Identifier* (OID). Cada OID é único e formado por uma série de números particulares com uma organização em árvore. Estes números pertencem a vendedores de equipamentos, companhias de informática ou organizações. O protocolo SNMP é constituído de duas entidades, uma no cliente e outra no servidor. Uma entidade SNMP consiste em uma engrenagem que é responsável por disparar requisições, processar requisições, segurança, controle de acesso e o controle de aplicações que leem e escrevem em MIBs (HEO; KIM; CHOI, 2010)

Com o padrão estabelecido pelo SNMP, e muitos fabricantes de equipamentos e desenvolvedores de Sistemas Operacionais adotarem este padrão, o uso do protocolo SNMP é muito vantajoso para fins de monitoramento de recurso pois é amplamente suportado nos equipamentos de *data centers*. Não apenas o monitoramento mas também a administração do equipamento remoto pode ser realizada através da alteração do valor de determinados objetos que irão disparar a execução de algo local.

2.3 Ferramentas para Implementação e Execução do Protótipo

Esta seção apresenta as ferramentas utilizadas para o desenvolvimento, validação e testes no protótipo. Para elaborar o protótipo, é necessário um ambiente virtual que é construído com KVM e o software para virtualização da rede OpenvSwitch. Os testes de carga são efetuados com a ferramenta OpenLoad. Para monitorar a rede, o próprio serviço OpenvSwitch fornece um cliente nos padrões estabelecidos pelo OpenFlow, capaz de controlar e coletar estatísticas. Para monitorar os servidores, o serviço SNMP proporciona um cliente da mesma forma. As informações são armazenadas em um repositório criado pela ferramenta RRDTool.

2.3.1 RRDTool

RRDTool é um software de licença GNU ⁴, desenvolvido inicialmente por Tobias Oetiker e atualmente por uma relação de patrocinadores que é mantida pela *Swiss Federal Institute of Technology*. Sua primeira versão estável foi lançada em 1999 com o intuito de aprimorar a ferramenta *Multi Router Traffic Grapher* (MRTG) ⁵. Atualmente, está na versão 1.5 lançada em 2015. O nome RRDTool é uma abreviatura de *Round-Robin Database TOOL*.

É uma ferramenta que tem como objetivo armazenar, em linha de tempo dados como uso de rede, temperatura, CPU, memória, processos, etc. É uma espécie de banco de dados, porém, com algumas particularidades quando comparado aos banco de dados comuns, que torna esta ferramenta ideal para cenários específicos de monitoramento. As informações armazenadas têm como chave primária o *timestamp* (data e hora) em que o dado é inserido. Os dados são armazenados de maneira circular, desta forma, o espaço utilizado pela base de dados permanece constante ao longo do tempo. Isso é possível pois, no arquivo de dados, as informações são rotacionadas com base no limite de informações especificado na criação do arquivo. Existe um marco inicial que é a data da criação do banco, quando o arquivo é criado, e se determina uma quantidade de dados que é o controle para a rotação da base. Os dados que ultrapassarem a quantidade prevista de informações serão inseridos nas posições iniciais do arquivo, substituindo as informações mais antigas.

As únicas operações permitidas nos arquivos de banco de dados são de “inserir” ou “consultar” dados. A ferramenta disponibiliza uma vasta gama de configurações e clientes para navegar, pesquisar, plotar gráficos, redefinir tamanho, entre outras, que podem ser facilmente integradas com *scripts* ou qualquer algoritmo com o objetivo de criar soluções que dependam de dados históricos ou estatísticos para tomada de decisão.

2.3.2 OpenvSwitch

OpenvSwitch ⁶ é um produto de licença Apache 2.0 ⁷ designado para habilitar automação de redes através de extensões programáveis suportando protocolos e interfaces padrões de redes de computadores. Segundo (PFAFF et al., 2015), com o aumento da complexidade de redes virtuais, necessidades de virtualização de redes e as limitações dos *switches* virtuais existentes, o OpenvSwitch ganhou popularidade.

A ferramenta pode operar em ambas plataformas: em software no *hypervisor*, que é o mecanismo responsável pela alocação de recursos virtuais, ou mesmo em hardware. Esta solução já está incorporada em muitas plataformas virtuais como XenServer ⁸, Xen Cloud Platform ⁹,

⁴<http://www.gnu.org/gnu/gnu-history.html>

⁵<http://oss.oetiker.ch/mrtg/>

⁶<http://openvswitch.org/>

⁷<http://www.apache.org/licenses/LICENSE-2.0.html>

⁸<http://support.citrix.com/article/CTX130418>

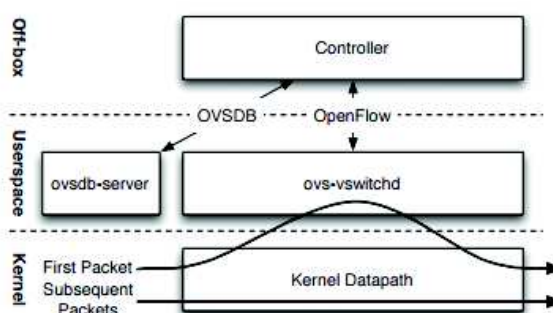
⁹<http://xen.org/products/cloudxen.html/>

KVM ¹⁰, VirtualBox ¹¹, entre outras. A ferramenta é nativamente distribuída no kernel do Linux com pacotes para distintas distribuições e suportada pelo FreeBSD e NetBSD.

O OpenvSwitch consiste em dois módulos principais que trabalham de forma integrada para fazer a manipulação dos pacotes de rede. Um módulo é integrado no kernel do sistema operacional para receber os pacotes na principal interface de rede e criar interfaces virtuais. Este módulo é uma espécie de plano de dados que executa ações de manipulação dos fluxos de acordo com regras predefinidas. O segundo módulo é desenvolvido para melhorar a performance no sistema operacional.

É um *switch* virtual utilizado para ambientes SDN e, desta forma, a principal maneira de interação com o plano de dados é através das definições do protocolo OpenFlow. A ferramenta ajuda na elaboração do modelo proposto nesta pesquisa, pois, além de suportar o protocolo OpenFlow, é vastamente utilizada em ambientes produtivos de equipamentos virtuais. A figura 3 demonstra a interação entre o kernel do sistema operacional, os módulos do OpenvSwitch e o OpenFlow.

Figura 3: Topologia OpenvSwitch



Fonte: (PFAFF et al., 2015)

2.3.3 Openload

OpenLoad ¹² é uma ferramenta para teste de carga de aplicações Web. É de simples execução, e é possível simular cargas semelhantes à realidade de muitos ambientes para medir a performance das aplicações testadas. Esta ferramenta é particularmente útil para testes de otimização, pois, é possível observar o impacto das solicitações imediatamente durante os testes.

É uma ferramenta em linha de comando que realiza cargas de requisições em determinadas URLs. Quando iniciada, é informada a URL que será acessada e o número de conexões simultâneas. Com estas informações, a ferramenta dispara as rajadas circulares de conexões e calcula diversas estatísticas referentes às solicitações. Este processo é interrompido quando a execução é finalizada manualmente.

¹⁰<http://www.linux-kvm.org/>

¹¹<http://www.virtualbox.org/>

¹²<http://openweblod.sourceforge.net/>

Figura 4: Execução do Openload

```

$ openload localhost 10
URL: http://localhost:80/
Clients: 10
MaTps 355.11, Tps 355.11, Resp Time 0.015, Err 0%, Count 511
MaTps 339.50, Tps 199.00, Resp Time 0.051, Err 0%, Count 711
MaTps 343.72, Tps 381.68, Resp Time 0.032, Err 0%, Count 1111
MaTps 382.04, Tps 727.00, Resp Time 0.020, Err 0%, Count 1838
MaTps 398.54, Tps 547.00, Resp Time 0.018, Err 0%, Count 2385
MaTps 425.78, Tps 670.90, Resp Time 0.014, Err 0%, Count 3072

Total TPS: 452.90
Avg. Response time: 0.021 sec.
Max Response time: 0.769 sec

```

Fonte: <http://openwebload.sourceforge.net/>

O OpenLoad, também conhecido como OpenWebLoad, é usado nesta pesquisa para validar a eficiência de balanceamento de carga com diferentes situações, validando o tempo de atendimento final da requisição em conjunto com o consumo dos recursos dos servidores do *pool*.

A figura 4 apresenta as informações retornadas após o Openload executar as operações de acesso ao serviço remoto, onde:

- **MaTps**: estimativa de transações por segundo durante 20 segundos.
- **Tps**: Transações por segundo, é o número de requisições processadas por segundo;
- **Resp Time**: O tempo de resposta das requisições;
- **Err**: percentagem de erros;
- **Count**: Somatório de requisições processadas;
- **Total TPS**: média de toda execução realizada;
- **Avg. Response time**: média de tempo de resposta;
- **Max Response time**: tempo máximo obtido em solicitação.

Devido às opções de estatísticas e a facilidade de uso do Openload, esta é uma ferramenta útil para validar o cenário do modelo proposto neste trabalho. Além de gerar diferentes cargas de requisições, as estatísticas como transações processadas por segundo e tempo de resposta auxiliam no processo de análise de desempenho.

3 TRABALHOS RELACIONADOS

O objetivo deste capítulo é explorar os mais recentes trabalhos relacionados com a proposta apresentada e que tenham relevância com o tema discutido. Aqui são apresentadas propostas e pesquisas publicadas que possuem características semelhantes ao modelo sugerido ou ao contexto em que se aplicam. Os trabalhos aqui apresentados foram selecionados através dos seguintes sítios digitais:

- ACM Digital Library ¹
- Springer ²
- ScienceDirect ³
- IEEEXplore Digital Library ⁴

Este capítulo está dividido em duas seções que classificam os trabalhos avaliados em trabalhos relevantes para balanceamento de carga de servidores e para balanceamento de carga de redes. Para completar este capítulo, é elaborada uma análise comparativa de itens relevantes entre os trabalhos avaliados. Esta análise é precisa e considera os pontos mais relevantes para o balanceamento de carga.

3.1 Balanceamento de Carga entre Servidores

Nesta seção, existem trabalhos selecionados devido ao tópico ser o balanceamento de carga entre servidores, onde, técnicas e avaliações foram realizadas em soluções para que uma requisição atinja um determinado servidor. Neste contexto, foi dada relevância a trabalhos com o uso de definições de SDN e o protocolo OpenFlow para realização da tarefa de balanceamento de carga. Também foram selecionados trabalhos que usam técnicas de obtenção de dados estatísticos para tomada de decisão de balanceamento.

3.1.1 Modelo de balanceamento de carga dinâmico com o uso do OpenFlow e SNMP

O trabalho desenvolvido em (SHANG et al., 2013) propõe uma estratégia de balanceamento de carga que considera a situação de recursos do *pool* de servidores. Um programa é responsável pelo levantamento de recursos dos servidores em intervalos de tempo através de consultas SNMP. Este algoritmo está incorporado a um controlador OpenFlow, de forma que, a troca de destino dos fluxos é baseada nas informações coletadas por este programa.

¹<http://dl.acm.org/>

²<http://link.springer.com/>

³<http://www.sciencedirect.com/>

⁴<http://ieeexplore.ieee.org/>

Usando *Server-based load balancing* como estratégia, o controlador determina o servidor com melhor capacidade para receber a requisição e o fluxo é alterado, dinamicamente, com troca de regras no plano de dados do *switch* OpenFlow. Teoricamente, com este modelo existe uma distribuição equilibrada, mesmo usando servidores com diferentes capacidades. Ambientes compostos por servidores com diferentes capacidades são, de certa forma, um desafio para soluções de balanceamento de carga, pois, necessitam de configurações e controles adicionais para balancear de forma igualitária de acordo com a capacidade de cada servidor.

As avaliações neste trabalho demonstram uma flexibilidade e baixo custo de implementação desta solução para balanceamento de carga. Além disso, o autor conclui que o uso desta técnica reduz o tempo de resposta de *Web Servers* e permite uma distribuição mais racional de recursos. Entretanto, o intervalo de coleta do uso de recursos pode ser um problema para uma distribuição correta e controle de congestionamento, pois, em poucos segundos, um único servidor pode ser sobrecarregado com requisições.

3.1.2 Avaliação de técnicas de balanceamento de carga com o uso do OpenFlow

Em (MARCON D., 2011) é feita uma avaliação usando diferentes técnicas de balanceamento de carga implementadas com o protocolo OpenFlow: *Random choice*, *Time slice based choice* e *Weighted balancing*. Através de conexões com o protocolo ICMP, para avaliar o tempo de resposta, as *requests* eram enviadas para cada servidor do *pool* usando uma das estratégias definidas durante um intervalo de tempo determinado. A avaliação considerou a melhor distribuição da carga baseada no tempo final de resposta da requisição ICMP.

Segundo os autores, a técnica mais eficiente foi *Weighted balancing*, que para cada nova requisição, é avaliado o servidor com melhor capacidade de processamento, ou seja, que menos recebeu requisições e tem a melhor capacidade de responder à próxima requisição. No entanto, a complexidade da implementação desta técnica é linear ao número de servidores que atendem o serviço, pois, para cada fluxo recebido é realizada uma análise de performance em cada um dos servidores do *pool*.

Esta técnica de consulta constante da performance dos servidores é bastante custosa e faz com que as requisições destinadas aos servidores fiquem enfileiradas no *switch* durante o tempo em que o algoritmo determina o servidor com melhor capacidade. Porém, a distribuição fica realmente precisa e equilibrada.

3.1.3 Comparativo de técnicas de balanceamento de carga em OpenFlow com o uso do Openload

Este trabalho utilizou o Openload para avaliar o desempenho das soluções de balanceamento de carga com o uso de regras de tráfego definidas com o protocolo OpenFlow. Os autores utilizaram ambientes simulados com o Mininet para realizar o experimento e a ferramenta Openload

para gerar cargas de processos. As cargas Web são geradas para realizar medições do tempo de resposta e transações por segundo.

O principal objetivo da pesquisa de (KAUR et al., 2015) foi avaliar algumas soluções de balanceamento de carga em *switches* OpenFlow. O switch virtual utilizado foi o OpenvSwitch e as regras de destino de tráfego alteradas no plano de dados através do controlador POX ⁵. Essa avaliação utilizou ambientes simulados para avaliação de desempenho onde a estratégia *Round Robin* foi comparada com *Random choice*.

Os autores avaliaram que a estratégia *Round Robin* tem melhor desempenho nos testes realizados, no entanto, salientaram a necessidade de testes em hardware real e não apenas testes em ambientes simulados. Este pesquisa tem uma relevância importante, pois, comprova que a implementação de balanceamento de conexões funciona em *switches* OpenFlow, no entanto as pesquisas foram limitadas e não avaliaram diferentes tipos de requisições, ou mesmo a situação dos servidores do *pool*.

3.1.4 O uso de reserva de recursos para balanceamento de carga de serviços Web

A solução posposta em (NAKAI; MADEIRA; BUZATO, 2015) faz uso de informações de estado (*status*) e carga de servidores monitorados para que um agente de decisão, conhecido como *RB-Master*, realize o balanceamento de carga. É uma solução que necessita de um cliente, *RB-Nodes*, para realizar operações de coleta de informações e latência. Ambos os módulos são responsáveis por coletar e divulgar informações sobre os servidores.

Nesta pesquisa, o número de requisições processadas, carga e capacidade foram itens utilizados para realizar a avaliação da solução. Os autores concluem que o monitoramento apropriado depende do modelo da tarefa de cada aplicação. Em alguns casos, é necessário calibrar o sistema para identificar a capacidade do servidor, com informações como o número de requisições em processamento e outros itens mais específicos. A ideia básica está também na necessidade de classificar as requisições de acordo com seu custo computacional e determinar a capacidade de processamento de cada servidor baseado no custo computacional de cada requisição.

Pode ser feita uma analogia entre as possibilidades criadas pelo OpenFlow e o módulo *RB-Master* em termos de ações e objetivos. Um dos problemas desta solução com o uso de um módulo centralizador é a capacidade de escalonamento, conforme explicado por (NAKAI; MADEIRA; BUZATO, 2015). Um único centralizador pode não ser suficiente, sendo necessário a divisão em zonas, em que distintos centralizadores são responsáveis por um grupo distinto de servidores. Outro problema pode ser a necessidade de intervenção nos servidores do *pool*, em que um agente precisa ser instalado e configurado para realizar as operações de monitoramento e tomada de decisão.

⁵<https://openflow.stanford.edu/display/ONL/POX+Wiki>

3.1.5 Implementação de um balanceamento de carga dinâmico com OpenFlow

(RAGALATHA P., 2013) propõe um balanceamento de carga dinâmico usando OpenFlow. Com o uso de um peso para cada servidor e o fluxo de rede baseado no número de transações processadas, é utilizada uma divisão para definir o servidor com melhor capacidade para processar a requisição. A carga do servidor é definida pelo *número de conexões ativas / peso do servidor*. Desta forma, com dados estatísticos de número de requisições em processamento, é estimada a carga atual de cada servidor.

O peso do servidor precisa ser previamente definido de acordo com sua capacidade de processamento. Esta necessidade dificulta a elasticidade do processo para inclusão e exclusão de servidores. Em adição à esta dificuldade, a solução não considera requisições de diferentes naturezas que podem ter diferentes custos de processamento.

A solução apresenta uma regra estática quando todos os servidores estão com a mesma carga de processamento ou sem processamento: *Round Robin* é usado para a definição do servidor, o primeiro servidor na definição irá receber a próxima requisição nesta situação.

Apesar desta solução ser bastante engessada, a ideia de definir limiares para a capacidade de cada servidor permite criar o limite que garante a qualidade de processamento da requisição. Desta forma, é possível controlar o número de requisições enviadas para cada servidor.

3.2 Balanceamento de Carga em Redes de Computadores

Nesta seção, existem trabalhos que foram selecionados devido ao tópico ser o balanceamento de carga de redes onde técnicas e avaliações foram realizadas em soluções que o propósito é o equilíbrio e uso mais eficiente de *links* de conexão. Os itens de tráfego de rede geralmente são considerados em situações em que se deseja obter um melhor desempenho e utilização de *links* de conexão para melhorar a vasão. Normalmente, estas técnicas causam efeitos colaterais como a necessidade de reorganização de pacotes, que é um processo bastante custoso, porém, tais técnicas são amplamente exploradas em *links* de acesso ou mesmo saída de *Data Centers* para garantir a disponibilidade e qualidade do serviço.

3.2.1 Balanceamento de carga baseado em fatias de fluxos usando OpenFlow

A ideia principal do trabalho de (WANG; LAN; CHEN, 2014) é melhorar o balanceamento de fluxos que trafegam em diferentes *links* classificando os fluxos em duas categorias: agressivos e normais. Esta pesquisa tende a melhorar o balanceamento de carga e o problema de reorganização de pacotes, que é uma tarefa custosa no processo de manipulação de fluxos.

Os autores propõem um algoritmo chamado OFS que analisa fatias de fluxos e classifica-os de acordo com as definições mencionadas. Para isso, os fluxos que trafegam em cada um dos links são avaliados em intervalos determinados de tempo. Com estas duas granularidades,

os fluxos podem ser direcionados ou redirecionados para os *links* que melhor comportam cada categoria. Com a classificação dos fluxos, cria-se uma espécie de previsão do que vai acontecer com os próximos tráfegos da mesma origem, gerando a possibilidade de manter estes fluxos em um *link* com a capacidade de transporte necessária.

O algoritmo proposto pelo autor mostrou-se eficiente com a classificação de fluxos, melhorando o balanceamento entre os canais de comunicação e diminuindo o problema de reorganização de pacotes, que eram os principais objetivos do trabalho.

3.2.2 Balanceamento de carga em redes de data center com arquiteturas Folded-Clos

A arquitetura de Clos Network tem o intuito principal de não ser bloqueante e foi desenvolvida por Charles Clos, em 1953, para soluções telefônicas da época. Atualmente esta topologia é amplamente usada para o processamento das interconexões de redes de computadores. O conceito de redes Folded-Clos é uma variação da arquitetura criada por Charles Clos e tem uma grande vantagens do uso de redundância de rotas de conexão onde é realizado o balanceamento dos fluxos. A proposta desenvolvida por (SEHERY; CLANCY, 2015) propõe dois algoritmos para controle de fluxos em uma rede Folded-Clos: *Selective Randomized Load Balance* e *Flow fit*.

Selective Randomized Load Balance é um algoritmo proposto com a analogia de bolas e caixas, onde, bolas são fluxos e caixas os *links*. No modelo tradicional, é colocada uma bola em uma caixa que é determinada de forma aleatória. Os autores propõem determinar duas caixas de forma aleatória e não apenas uma, com isso, colocar a bola na caixa com menos bolas. A conclusão é de que, desta forma, reduz-se o número de bolas em caixas, exponencialmente.

O *Flow fit* usa uma técnica para determinar o destino do fluxo, analisando informações sobre o tamanho individual de um fluxo, o tamanho do *link* e a capacidade de fluxos em cada link. Com estas informações, define-se que um *link* sobrecarregado é um *link* com mais da metade de sua capacidade em uso e, caso contrário, o *link* está abaixo da capacidade. O tempo de monitoramento das informações de uso do *link* é de 5 segundos. Desta forma, os fluxos maiores podem ser transferidos para links com melhor capacidade.

3.3 Análise Comparativa

Esta seção apresenta os trabalhos estudados de forma pontual comparando as características que cada um explorou. Para avaliar estas características e a abrangência de cada trabalho, foram elaboradas duas tabelas: a tabela 2 que apresenta os itens considerados para execução do balanceamento de carga; e a tabela 3 que disponibiliza as estratégias de balanceamento que cada trabalho pesquisou.

Os itens da tabela 2 foram definidos com base em informações da infraestrutura que envolve

Tabela 1: Legenda de trabalhos estudados

3.1.1	(SHANG et al., 2013)	Modelo de balanceamento de carga dinâmico com o uso do OpenFlow e SNMP
3.1.2	(MARCON D., 2011)	Avaliação de técnicas de balanceamento de carga com o uso do OpenFlow
3.1.3	(KAUR et al., 2015)	Comparativo de técnicas de balanceamento de carga em OpenFlow com o uso do Openload
3.1.4	(NAKAI; MADEIRA; BUZATO, 2015)	O uso de reserva de recursos para balanceamento de carga de serviços Web
3.1.5	(RAGALATHA P., 2013)	Implementação de um balanceamento de carga dinâmico com OpenFlow
3.2.1	(WANG; LAN; CHEN, 2014)	Balanceamento de carga baseado em fatias de fluxos usando OpenFlow
3.2.2	(SEHERY; CLANCY, 2015)	Balanceamento de carga em redes de data center com arquiteturas Floded-Clos

os dados de rede e de recursos dos servidores. O item **Carga dos Servidores** classifica os trabalhos que avaliam a utilização de recursos dos servidores do *pool* antes de definir se a conexão será destinada para um, ou outro servidor. Dados de servidores podem ser amplos, como informações de CPU, memória, IO, dados do sistema operacional, número de processos ou mesmo dados específicos da aplicação em execução. Desta forma, a classificação não foi detalhada, apenas levando em conta se o modelo estudado considera algum dado dos servidores ou não. **Fluxos da rede** aponta os trabalhos que consideram o fluxos repassados para os servidores do *pool* e tentam balancear as conexões de acordo com os fluxos transmitidos. **Latência da rede** é um item que verifica os tempo de resposta que os servidores do *pool* consomem para uma determinada solicitação, a maior latência pode significar que o servidor está sobrecarregado ou mesmo a rota de comunicação congestionada. O item de **Bytes Processados** são informações semelhantes aos fluxos de rede, mas significam a quantidade de bytes enviados e retornados pelo servidor.

Os itens da tabela 3 são referentes às estratégias de balanceamento de carga que os modelos utilizaram para elaborar as respectivas propostas. As estratégias definidas abrangem os algoritmos encontradas na literatura estudada, ou mesmo as soluções apresentadas pelos trabalhos relacionados para implementar suas propostas. **Random choice** é um algoritmo que, em intervalos de tempo, seleciona randomicamente um dos servidores do *pool* para encaminhar a requisição. **Round Robin** é um algoritmo que destina as requisições de forma circular, ou seja, uma para cada servidor do *pool*, usando uma lista de cima para baixo e retorna ao início quando encontra o fim. **Load Based** são as estratégias que, de alguma forma, consideram a carga dos servidores ou da rede antes de definir quem receberá a conexão. O item que considera **Thresholds** são os algoritmos que estimam a capacidade de cada servidor, ou mesmo limitam o número de requisições que podem ser destinada para cada servidor do *pool*. Entretanto, este

item não avalia realmente a situação dos servidores, apenas cria um limite "seguro" de requisições que cada servidor pode receber.

Com o intuito de manter a organização, cada linha das tabelas representa um dos trabalhos estudados, respeitando a legenda disponível na tabela 1.

Tabela 2: Análise comparativa de itens considerados

	Considera carga dos servidores	Considera fluxos da rede	Considera latência da rede
3.1	X		
3.2	X		
3.3			
3.4	X		X
3.5		X	
3.6		X	
3.7		X	

Tabela 3: Análise comparativa de estratégias implementadas

	Random Choice	Round Robin	Load Based	Implementa Thresholds
3.1	X	X	X	X
3.2	X	X	X	
3.3	X	X		
3.4			X	X
3.5				X
3.6				X
3.7	X	X		X

Na tabela 2 observa-se que o número de **Bytes Transferidos** para cada um dos servidores do *pool* de balanceamento é um item ignorado em todos os trabalhos estudados, porém, é um item importante pois, pode representar a demanda de informação processada por cada um dos servidores ou mesmo ser um item estabelecido como um *threshold* na definição da carga histórica processada por cada servidor.

De maneira geral, o uso das estratégias apresentadas na tabela 3 podem ser combinados, direta ou indiretamente, com os itens da tabela 2 criando algoritmos eficientes, de acordo com a proposta desta pesquisa. Assim sendo, pode haver um arranjo entre os itens das duas tabelas para elaborar uma proposta mais precisa e abrangente.

Os trabalhos estudados tem a deficiência de explorar recursos de rede e dos servidores simultaneamente para definir o balanceamento de carga. O modelo proposto por (NAKAI; MADEIRA; BUZATO, 2015) no item 3.4, é o que mais se assemelha com a proposta aqui apresentada, porém, ignora dados de **Fluxos de Rede** e **Bytes Transferidos** para a definição do servidor destino.

Considerando esta avaliação realizada, observa-se as seguintes oportunidades de pesquisa:

- União de indicadores de rede e de carga dos servidores para balanceamento de carga;
- Definição de itens menos relevantes, como **Thresholds**, para definir indicadores limites antes de definir o servidor que receberá a requisição;
- Explorar itens de rede como **fluxos** e **bytes** como variação de qualidade que, de maneira unificada e considerando as devidas proporções de importância, define o servidor de destino;

Este capítulo apresentou os trabalhos encontrados que mais se relacionam com a proposta que será apresentada neste trabalho. Esta seção apresentou uma avaliação dos principais aspectos encontrados que são relacionados com balanceamento de carga, algoritmos e informações estatísticas, que visam criar soluções de balanceamento de carga. O próximo capítulo apresenta o modelo proposto, que pretende abranger todos os itens apresentados nesta seção, usufruindo destes dados de forma coesa e equilibrada, para criar uma solução eficaz para os problemas discutidos nos modelos tradicionais de balanceamento de carga.

4 MODELO PROPOSTO

Neste capítulo será introduzida as decisões que levaram a este projeto, a arquitetura desenvolvida juntamente com a implementação que foi realizada para atingir o objetivo de balanceamento de carga e caracterização do modelo, denominado *CoreLB*.

Este modelo cobre algumas lacunas dos trabalhos estudados, avaliando o uso das técnicas propostas nestes trabalhos. O intuito foi modelar uma solução com ganhos na qualidade e na precisão do balanceamento de carga para serviços Web. O modelo proposto também tem como objetivo o fomento do uso de SDN com a integração do protocolo SNMP para monitoramento de recursos, pois, estes dois universos são válidos para análise da situação dos nós computacionais e processamento de requisições em execução.

A metodologia apresentada neste trabalho é projetada para ser executada em *switches*, onde os fluxos de conexões já são manipulados nativamente, sem a necessidade de equipamentos adicionais. A coleta de informações sobre recursos dos servidores pertencentes ao *pool* é realizada através do serviço SNMP, que é padrão e bem definido para este propósito. Desta forma, cada fluxo é direcionado para um dos servidores, de acordo com a política de balanceamento em uso.

4.1 Decisões do Projeto

A metodologia apresentada pode ser utilizada para soluções que necessitem balanceamento de conexões entre um *pool* de servidores. Porém, a avaliação da solução foi realizada com serviços Web, nos quais não existe a dependência de sessão, conhecidas como comunicação *stateless*. Uma sessão é uma semi-permanente informação interativa, também conhecida como um diálogo, entre dois ou mais dispositivos, ou entre um computador e um usuário. Uma sessão é estabelecida em um determinado momento e terminada em algum momento posterior.

As soluções *Server-Based Load Balance* são implementadas através de hardware ou software, com a obtenção de informações pertencentes ao *pool* de servidores. É um modelo geralmente recomendado, pois, não necessita alteração nos clientes, de forma a deixá-los decidir em que servidor conectar. Pode ser implementado em hardware, onde existe um *appliance* no meio da comunicação que garante a função de balanceamento, ou software, em que existe algum aplicativo, como um *cluster*, instalado entre os servidores. Ambas podem fazer o uso de algoritmos como *Round Robin*, *Random Choice*, *Time Slice Based Choice*, *Weighted Balancing*, entre outros, que determinam a política de balanceamento.

As técnicas clássicas para balanceamento de carga podem ser reproduzidas da mesma forma com o modelo proposto pelo protocolo OpenFlow, inclusive, com melhor performance, de acordo com a pesquisa demonstrada em (C.C. OKEZIE OKAFOR K.C, 2013) indica um melhor desempenho no modelo OpenFlow de *switch* comparado com modelos convencionais, mostrando-se ser uma melhor alternativa para software *Ethernet Switching* ou mesmo *IP Routing*. Entretanto, a solução para o problema de balanceamento de carga, como implementar e o

que controlar para atender o balanceamento, são itens distintos.

A primeira etapa necessária, foi o desenvolvimento do ambiente executando a tarefa de balanceamento das requisições pela rede. Para realizar este processo, foi necessário usar o protocolo OpenFlow com a ferramenta OpenvSwitch que suporta o protocolo. O OpenFlow viabiliza a adição de regras no plano de dados do *switch* virtual, que é alimentado com informações do destino que os fluxos entrantes devem tomar. São regras de roteamento de fluxos que podem ser alteradas de acordo com a necessidade de processamento, latência ou mesmo o histórico dos fluxos. É possível fazer balanceamento no OpenFlow com um algoritmo que avalia a situação dos servidores do *pool*, dados de rede e adiciona regras de fluxos de acordo com o modelo de balanceamento de carga. O processo de análise dos servidores e das estatísticas de rede pode ser cíclico, pode também ser dinâmico ou automatizado, de acordo com a melhor estratégia definida.

O ambiente deve ser configurado de tal forma que os clientes irão ter o conhecimento de um único ponto de acesso (endereço de IP ou *hostname*) que é uma conexão destinada ao *switch* virtual criado pelo OpenvSwitch, em que os servidores, também virtuais, estarão conectados. Todos os testes idealizados para este trabalho foram executados em equipamentos virtuais. O ambiente virtual permite a inclusão e exclusão de servidores de maneira econômica para avaliação que está sendo realizada neste trabalho.

A última etapa foi a definição de como é realizada a escolha do servidor ativo em cada intervalo de tempo. Para isso, são utilizadas informações estatísticas da rede, proporcionadas pela própria camada de monitoramento do OpenFlow, e da obtenção do uso de recursos dos servidores com o uso do protocolo SNMP, de acordo com as definições que serão explicadas no capítulo 5 onde a implementação é explicada. As proporções e relevâncias de cada informação coletadas e analisadas para definição do servidor ativo podem ter importâncias distintas, ou seja, alguns dados podem ser mais relevantes que outros.

Este capítulo ainda apresenta a arquitetura, modelagem, os objetivos e modelo de comunicação para a criação do protótipo desenvolvido. Após será apresentado o capítulo de Resultados.

4.2 Arquitetura

Nesta seção, é apresentada a arquitetura do modelo proposto. A exposição que se segue está dividida em itens envolvidos no cenário na seção de Atores e Agentes, a apresentação do objetivo e o modelo de comunicação.

4.2.1 Atores e Agentes

Atores são os componentes que interpretam ou representam/executam a ação de balanceamento de carga. São peças fundamentais para o funcionamento da atuação, neste contexto, na execução do balanceamento de carga. Agentes são componentes que irão interagir com os

atores para coletar dados estatísticos e comandar a execução de ações. Existem duas ações que são executadas por todos os atores e agentes: *Consumir* e *Prover*. Isso significa que, em alguns momentos, os atores consomem ou provêm informações e os agentes, da mesma forma, podem consumir ou prover informações.

Existem quatro Atores fundamentais na topologia de balanceamento de carga tradicional : o **Usuário**, a **Rede** (*comutadores*), o **Balanceador de Carga** e os **Nós Computacionais**. Na topologia proposta neste trabalho, o *Usuário* não recebe intervenção e o *Balanceador de Carga* não existe, pois, a ação de balanceamento é executada pelo *Comutador*. Desta forma, restam dois Atores que precisam de intervenção no modelo: o *Comutador* e os e os *Nós Computacionais*, assim sendo, na relação abaixo, estão estes dois Atores e um terceiro nomeado **Agregador**, que é responsável por armazenar dados estatísticos dos dois primeiros Atores, e manter estes dados acessíveis para todos os agentes que serão descritos a seguir. Desta forma, a relação de atores é:

- Nós Computacionais;
- Comutadores;
- Agregador.

Os Agentes são responsáveis por interagir com estes Atores. Os Atores não interagem entre si, eles apenas interagem com os Agentes e os Agentes, por sua vez, também não interagem entre si, eles apenas interagem com os Atores. Com o intuito executar a comunicação necessária para *Consumir* ou *Prover* informações, estão projetados os seguintes agentes:

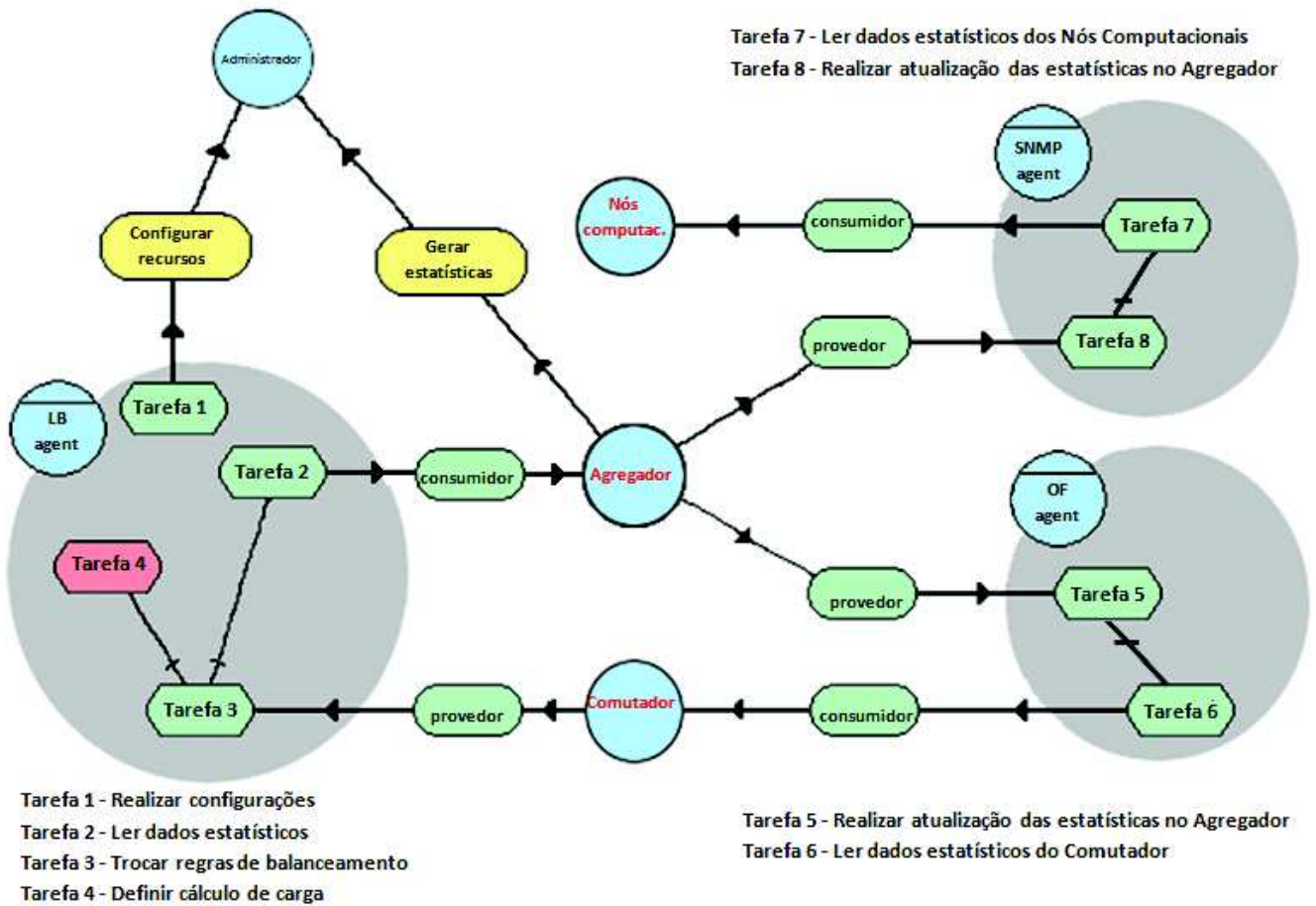
- Agente SNMP;
- Agente OF;
- Agente LB.

A figura 5 define os Atores e Agentes com suas respectivas tarefas e objetivos. A topologia usa a abordagem i^* ¹, que é uma modelagem para orientação de agentes que contempla os requisitos de cada agente de acordo com suas características e necessidades. Este modelo de diagrama para representar objetivos, requisitos e para modelar dependência entre processos é usado como padrão pela *International Telecommunication Union* (ITU) para modelagem de redes de maneira sócio comunicativa ². Este modelo contempla as dependências entre Atores e Agentes, com as respectivas ações. De acordo com a figura 5: os Atores estão ao centro com legenda em vermelho; os Agentes são os círculos com um corte, para se diferenciarem dos Atores; cada um dos Agentes tem seu universo de atividades representado pelos círculos

¹<http://www.cs.toronto.edu/km/istar/>

²<http://www.cs.toronto.edu/km/istar/>

Figura 5: Topologia da solução proposta representando os agentes e atores com suas comunicações



maiores em cor cinza; os itens em verde representam as tarefas que cada Agente executa e a característica de cada tarefa fora do círculo;

O **agente SNMP** é um *consumidor* para os **Nós Computacionais** e um *provedor* para o **Agregador**. Este agente realiza duas tarefas principais: ler as estatísticas de recursos de cada servidor e armazenar as informações estatísticas de recursos no Agregador. A operação de leitura de recursos de hardware é baseada em MIBs específicas para monitoramento de recursos como CPU, memória, *load*, entre outras, usando as definições do protocolo SNMP. Este agente realiza uma operação cíclica consultando todos os servidores do *pool* em intervalo de tempo predeterminados. Para cada Nó Computacional existe uma base específica no Agregador de maneira estruturada com o uso da ferramenta RRDTool, em que os dados coletados são armazenados. Este agente utiliza os clientes disponíveis pela ferramenta RRDTool para armazenar os dados coletados no Agregador.

O **agente OF** é *consumidor* do **Comutador** realizando a tarefa de coleta de estatística de fluxos e bytes transferidos para cada servidor do *pool*, e *provedor* do **Agregador** armazenando estas estatísticas coletadas. A coleta de informações do *Comutador* é realizada com o cliente distribuindo pela ferramenta OpenvSwitch. Estas operações são realizadas de forma cíclica da mesma forma que o agente anterior, porém, não necessariamente nos mesmos intervalos de

tempo.

O **agente LB**, por sua vez, é *consumidor* do **Agregador** e *provedor* do **Comutador**. Este agente tem uma complexidade um pouco maior, pois ele realiza a tarefa de definição do servidor com melhor capacidade de processamento. Ele depende dos agentes OF e SNMP para executar sua tarefa de forma eficiente. Usa o cliente RRDTool para coleta dos dados armazenados e o cliente do OpenvSwitch para realizar a troca de regras no *Comutador* que irá fazer o balanceamento de carga definido.

4.3 Objetivo da Arquitetura Proposta

Os usuários que irão usufruir do serviço têm conhecimento de um único endereço IP, chamado de *service IP* que é o ponto único de acesso. O comutador deve tratar este endereço IP e direcionar a conexão para o servidor com melhor capacidade de processamento. Para isso, torna-se necessário ter o controle da manipulação dos fluxos de rede e saber a situação de recursos dos servidores do *pool*. O **agente SNMP** realiza a tarefa de análise de recursos dos servidores, o **agente OF** adiciona maior precisão à análise de estatísticas, pois, coleta informações da rede do ponto de vista do *switch* e o **agente LB** define a ação de balanceamento. Com estes três Agentes, o ambiente para execução de balanceamento de carga está completo, pois, conecta todos os Atores possibilitando a comunicação entre todos os envolvidos.

O balanceamento no contexto do OpenFlow é realizado alterando o destino do pacote para direcionar a conexão entrante para um servidor com o maior poder de processamento no momento da solicitação. O melhor poder de processamento é definido por um algoritmo, que considera os dados coletados através do protocolo SNMP em adição com os dados estatísticos fornecidos pelo protocolo OpenFlow. Após a definição do servidor, serão utilizadas as técnicas de *layer 3* do OpenFlow para direcionamento da conexão. Neste contexto, o pacote destinado a um endereço IP único, conhecido pelo cliente, tem seu endereço de destino reescrito para o endereço de IP do servidor desejado, quando a requisição alcançar o *switch* em que os servidores estão conectados. No momento em que o servidor retorna a requisição, o IP de origem é alterado para o IP que foi inicialmente solicitado para não corromper a comunicação. Desta forma, são duas alterações realizadas no pacote recebido: no momento em que ele entra, e no momento em que sai. Esta tarefa é nativamente realizada pelo *switch* com as definições estabelecidas pelo **agente LB**.

Uma das preocupações com este modelo é o desempenho que o *switch* necessita ter para não criar gargalos em ambientes com altas demandas de solicitação. Entretanto, a análise de desempenho realizada por (BIANCO et al., 2010) comprova a eficiência e vazão das operações em *layer 3* do OpenFlow superando entre 30% e 40% a performance de *switching layer 2* em implementações baseadas em Linux. Nestes testes, também se observa uma ótima performance do OpenFlow para lidar com múltiplos fluxos e capacidade de gerenciá-los, combinado com a excelente vazão e baixa latência, itens que têm encorajado o uso desta tecnologia.

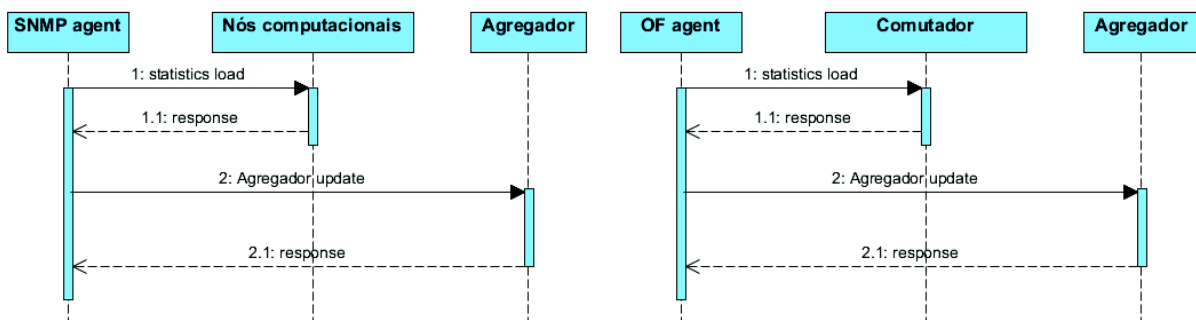
4.4 Modelo de Comunicação Proposto

As comunicações são bilaterais, cada agente conecta com seu parceiro e com o *Agregador*. O *agente SNMP* e o *agente OF* não são dependentes um do outro e trabalham em paralelo, seus parceiros são os Nós computacionais e o Comutador, respectivamente. O processo de comunicação destes dois agentes segue o modelo apresentado a seguir:

1. O agente conecta no parceiro e solicita os respectivos dados;
2. O agente recebe o retorno dos dados solicitados;
3. O agente conecta no *Agregador* e armazena os dados recebidos;
4. O *Agregador* confirma a execução solicitada.

O *agente SNMP* está inicialmente projetado para realizar consultas em intervalos de tempo menores que o *agente OF*. A demanda de requisições SNMP não pode interferir o processamento do servidor. Os dados de rede funcionam como informações "*delta*" e cada servidor tem um determinado valor chamado de "*quality node*", o qual é reduzido de acordo com o *delta* de rede. Maior número de fluxos, bytes e latência significam um *delta* maior, o que induz a redução do valor atribuído ao *quality node* do servidor em questão. A figura 6 apresenta o diagrama de sequência modelado para o *agente SNMP* e para o *agente OF*.

Figura 6: Diagrama de sequência dos agentes SNMP e OF



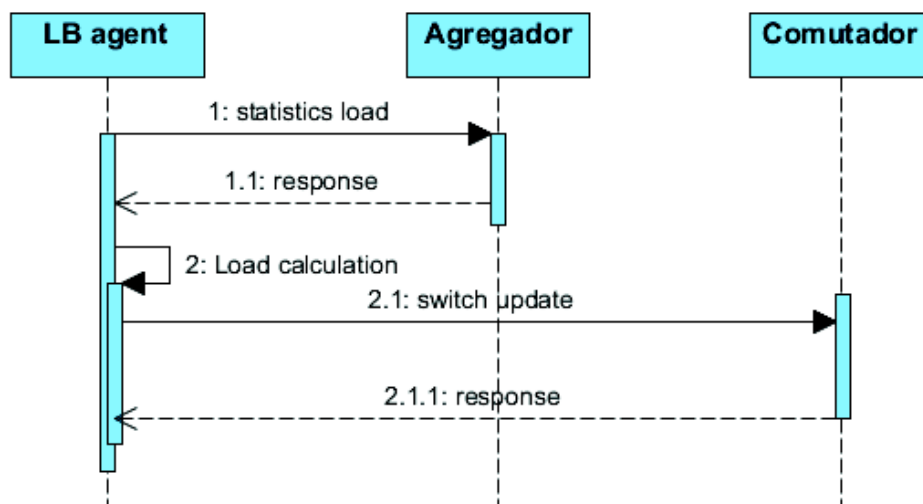
O *agente LB* é um pouco diferenciado dos outros agentes. Este agente não tem dependência direta dos outros agentes, porém, as informações coletadas por eles são fundamentais para que o *agente LB* execute sua tarefa de forma assertiva. A tarefa inicial deste agente é consultar o *Agregador* para realizar a carga dos dados coletados com o intuito de definir o *quality node* de cada servidor. Este agente é composto por um algoritmo de balanceamento de carga que, com os dados coletados, determina as devidas proporções e pesos de cada informação para definir *quality node* em intervalos de tempo. O *agente LB* define o servidor que tiver o maior *quality node* como o próximo destino. A intervalo de execução deste agente é dinâmico, ou seja, ele se adapta à demanda de requisições. Quando maior a discrepância do *quality node* entre os

servidores, mais frequente ele realiza o cálculo. Caso o *quality node* esteja semelhante entre os servidores, este intervalo de cálculo é executado com menor frequência. A sequência de comunicação é apresentada abaixo:

1. O agente conecta no *Agregador* para coletar os dados;
2. O agente executa o cálculo de balanceamento;
3. O agente conecta no *Comutador* e executa a operação de balanceamento;
4. O *Comutador* confirma a execução da operação.

A figura 7 apresenta o diagrama de sequência do *agente LB*. Este agente possui três atividades, duas externas e uma atividade interna para calcular a carga de cada servidor.

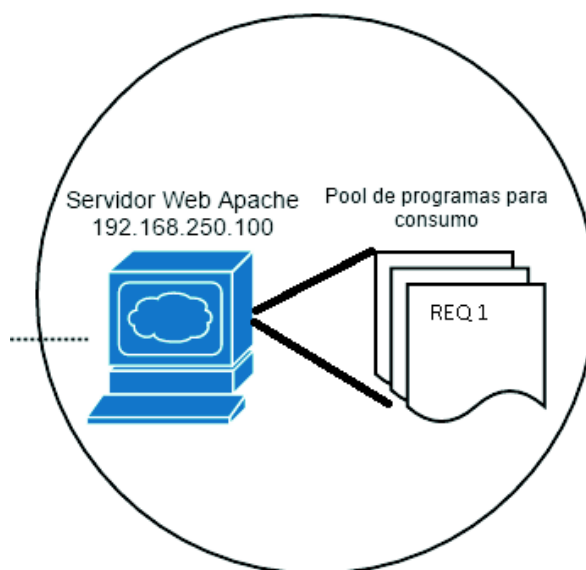
Figura 7: Diagrama de sequência dos agente LB



5 IMPLEMENTAÇÃO

A proposta desenvolvida neste trabalho é a de um modelo para execução de balanceamento de carga baseado nos indicadores de consumo rede e de recursos computacionais dos servidores que atendem à um serviço Web. Os testes para a validação do protótipo foram executados usando servidor de páginas Apache¹, onde, cada nó computacional tem o serviço instalado e rodando, atendendo requisições na porta 80 com o protocolo HTTP. Os servidores disponibilizam uma série de programas com diferentes características que são executados quando uma específica URL é requisitada, a figura 8 apresenta esta implementação. Os programas disponibilizados pelo serviço Web tem diferentes características pois realizam operação com diferente consumo de recursos computacionais no servidor em execução. Alguns consomem CPU, outros memória, banda de rede, swap e outros recursos. Os programas desenvolvidos para validação do protótipo estão detalhadamente explicados na seção 6.1.1 (Consumidores).

Figura 8: Aplicações Web disponíveis em cada servidor



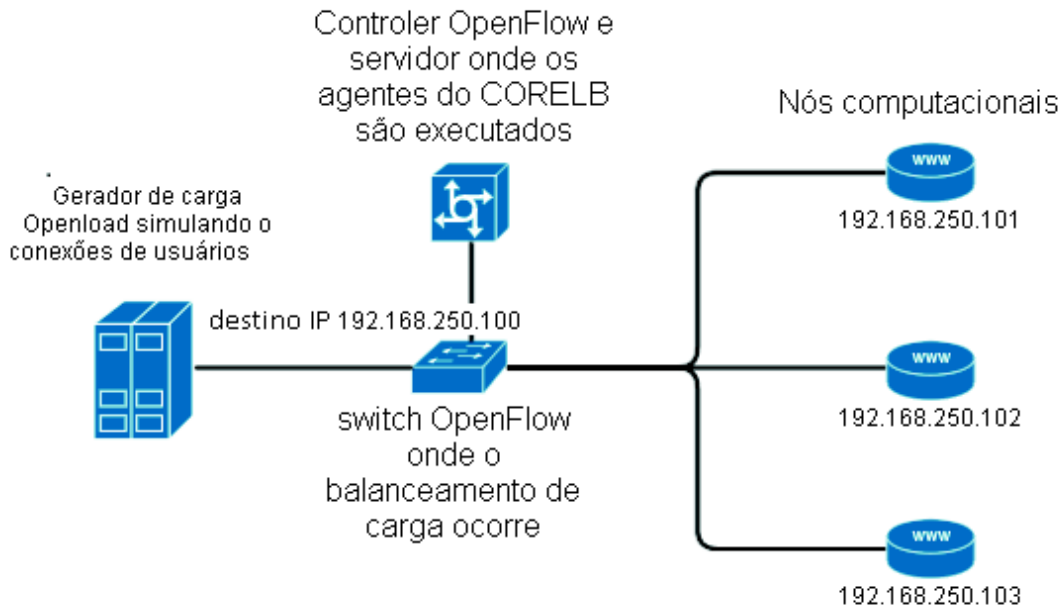
O protótipo para balanceamento de carga foi testado neste ambiente Web que realiza operações semelhantes a comércio eletrônico, rede social ou mesmo portal corporativo. A figura 9 representa o ambiente que foi virtualizado com Xen Server² para realizar os testes e validação do *CoreLB*. A topologia sugerida pelo OpenFlow designa um controlador que é um Sistema Operacional capaz de comunicar-se com o *switch* e realizar as tarefas de administração de fluxos. Neste mesmo servidor, onde o controlador é executado, existe os agentes do *CoreLB*. Neste ambiente, foram inicializadas três servidores virtuais conectados no *switch* virtual OpenvSwitch, equipamentos chamados de nós computacionais. Para simular o tráfego de usuários, uma servidor virtual adicional foi conectada ao *switch* para executar o Openload, que dispara conexões

¹<https://httpd.apache.org/>

²<http://xenserver.org/>

para o IP virtual 192.168.250.100, conhecido como *service IP*, que tem seu fluxo direcionado para um nós computacionais de acordo com as definições do *CoreLB*.

Figura 9: Topologia virtualizada com Xen Server



Na topologia virtualizada, há o node 1, node 2 e node 3, respectivamente 192.168.250.101, 192.168.250.102 e 192.168.250.103, que são servidores virtuais rodando Ubuntu 15.04 com 2Gb de memória RAM, 1Gb de swap e um processador dedicado Intel Core i7 com 2.00GHz. O servidor que virtualiza este ambiente é também um Ubuntu 15.04 com 8Gb de memória RAM, 2Gb de swap e 4 processadores dedicados Intel Core i7 com 2.00GHz. O OpenvSwitch é executado pela máquina virtualizadora onde o *CoreLB* também é executado para realizar as tarefas propostas. O cliente é uma máquina virtual externa de mesma configuração que os nós computacionais onde o OpenLoad é executado.

5.1 Funcionamento do *CoreLB*

O nó computacional ativo (**NoAtv**), que recebe as conexões no momento, é determinado de acordo com a qualidade computacional, ou seja, o nó com melhor qualidade é o nó ativo. Cada um dos nós é classificado com uma determinada qualidade a partir de um calculo do uso dos recursos computacionais. Esta qualidade é reduzida a medida em que o nó processa maior número de tarefas que os demais. A discrepância de qualidade entre o nó superior (**NodeBest**), com melhor qualidade, e nó inferior (**NodeWorst**), com pior qualidade, é o fator que determina a realização do balanceamento de carga para outro nó computacional. A qualidade computacional é determinada por uma pontuação. O nó computacional que realiza mais tarefas e tem maior consumo de recursos de hardware têm a pontuação de qualidade aumentada. A maior pontuação

significa menor qualidade e a menor pontuação significa que o nó computacional executou e processou um conjunto de tarefas que consumiu menor quantidade de recursos.

Os recursos analisados para determinar qualidade são provenientes de dois pontos distintos: **Switch OpenFlow** e serviço **SNMP** de cada nó computacional. Na tabela 4 há os dados utilizados para definição de qualidade dos nós computacionais e de onde os dados provem. As análises de definição de qualidade ocorrem em tempos determinados e arbitrários e controlados por parâmetros disparados pelo agente **TimeKeeper**, que será posteriormente explicado. O sistema de pontuação para definir a qualidade compara o consumo de recursos dos nós computacionais, ou seja, o nó que tiver maior consumo de um dos recursos monitorados, recebe a pontuação definida em **PontAtrib**. Como existem uma série determinada de recursos monitorados, vários pontos podem ser atribuídos aos nós computacionais em cada execução dos agentes. Por exemplo, o nó computacional que tiver consumido maior quantidade de *CPU User*, no ultimo intervalo de tempo monitorado, irá somar **PontAtrib** à sua qualidade, desta forma, maior pontuação representa menor qualidade. Nos testes realizados e resultados obtidos, todos os tipos de recursos monitorados tem atribuído o mesmo **PontAtrib**, porém esta pontuação pode ser diferenciada ou mesmo negativa de acordo com o uso do recurso e sua importância. No caso do recurso *CPU idle*, que significa mais tempo com recursos livres e sem processar tarefas, o **PontAtrib** é negativo, com o intuito de reduzir a pontuação definida, melhorando assim o indicador de qualidade.

O monitoramento de dados de rede, *pacotes* e *bytes* transferidos, ocorre com maior frequência por se tratar de uma consulta centralizada ao *switch* OpenFlow, onde concentra-se as estatísticas completas destas informações para todos os nós computacionais. Por se tratar de uma consulta menos custosa, pois não requer processamento dos nós computacionais ou mesmo comunicação com eles, este procedimento não influencia pode ser mais frequente. Desta forma, os dados de rede são considerados pontos *delta* de qualidade e a formula definida para qualidade é o consumo de recursos monitorados usando o protocolo SNMP mais o *delta* de tráfego de rede conforme fórmula abaixo.

$$qualitynode_n = load_n + networkdeltaquality_n \quad (5.1)$$

A medida em que o tempo passa e os equipamentos processam tarefas ou recebem dados através da rede, esta pontuação aumenta para nós computacionais. Esta qualidade deve ser truncada em intervalos tempo conhecidos como **T3** (explicado na seção TimeKeeper) para que a contagem recomesse, marcando um novo inicio no processo de pontuação. Assim, como o agente TimeKeeper, outros agentes são necessários para a execução da definição de qualidade e execução do balanceamento de carga. Os agentes OF, SNMP e LB são detalhados nas seções posteriores ao TimeKeeper que é abordado nas próximas subseções.

Tabela 4: Dados analisados para determinar a qualidade de cada nó computacional

SNMP	CPU User	O tempo de CPU gasto para requisições processadas em <i>User Space</i> . São todas as requisições de programas que não pertencem ao Kernel do Sistema Operacional
SNMP	CPU Idle	É o tempo em que o processador não está sendo usado por nenhum programa
SNMP	CPU System	É o tempo gasto no processamento de requisições de programas gerenciados pelo Kernel do Sistema Operacional
SNMP	CPU Wait	É o tempo onde as requisições precisam esperar para acessar os recursos de CPU. Isso acontece quando existe um gargalo por falta de CPU ou em ambientes virtualizados onde máquinas virtuais competem para acessar os recursos
SNMP	CPU Nice	Tempo gasto reagendando prioridades de processamento
SNMP	CPU Kernel	Representa o tempo de CPU gasto para processar códigos do Kernel do Sistema Operacional
SNMP	CPU Interrupt	Tempo de processamento de requisições the IRQ de hardware.
SNMP	CPU SoftIRQ	É o tempo gasto em requisições de CPU que tem prioridade para processar a fila de IRQ de software. As requisições para interrupções tem máxima prioridade pois precisam ser processadas rapidamente, caso contrário o sistema pode apresentar lentidão ou os <i>buffers</i> de hardware podem sofrer <i>overflow</i> e dados serão perdidos
SNMP	Memória RAM total	Memória total disponível no Sistema Operacional
SNMP	Memória RAM usada	O total de memória usado ao longo do tempo
SNMP	Memória RAM shared	Total de memória que é compartilhada entre processos
SNMP	Memória RAM buffer	Total de memória alocada para utilização com buffers
SNMP	Memória RAM cached	Memória usada para realizar o cache de informações
SNMP	SWAP total	Total de memória alocada para <i>paging requests</i>
SNMP	SWAP usado	Total de memória em uso para <i>paging requests</i>
OpenFlow	Pacotes de rede processados	Número de pacotes transferidos e recebidos do nó computacional que foram manipulados pelo switch
OpenFlow	Bytes de rede processados	Número de bytes transferidos e recebidos do nó computacional

5.1.1 TimeKeeper

O TimeKeeper é responsável pelas execuções dos Agentes de monitoramento e balanceamento. O conceito de TimeKeeper existe neste processo para o controle da execução das tarefas de forma dinâmica, que baseia suas decisões na discrepância de qualidade e tempos diferenciados de execução. Este agente é executado em um laço infinito e é responsável pela inicialização do restante dos Agentes para que cumpram as tarefas e realizem o balanceamento de carga. O TimeKeeper também é, indiretamente, responsável por controlar a discrepância de qualidade dos nós computacionais, pois, as execuções dos Agentes manipula os indicadores de qualidade dos nós computacionais e a discrepância de qualidade altera o destino das requisições.

Inicialmente, um intervalo de tempo é necessário para calibrar e definir a qualidade inicial de cada nó computacional, intervalo definido como **TempIni**. Quando o processo é inicializado, o TimeKeeper dispara o Agente SNMP. Após o termino desta execução, aguarda o intervalo definido em **TempIni** e dispara novamente o Agente SNMP. O consumo de recurso de cada nó computacional no intervalo de **TempIni** definirá a qualidade inicial (**QualIni**) de cada nó. O nó com melhor qualidade será o nó ativo (**NoAtv**). A figura 10 mostra um algoritmo abstraído do código do TimeKeeper para realizar a definição de (**QualIni**) e, conseqüentemente, determinar o (**NoAtv**), que é uma tarefa realizada pelo Agente LB que será posteriormente explicado.

Figura 10: Algoritmo TimeKeeper para definir a qualidade inicial

```

1  # define initial quality
2  nodes(n)=0
3  # first snapshot of resources
4  start snmp_agent
5
6  sleep TempIni
7
8  # second snapshot of resrouces
9  start snmp_agent
10 # start lb agent to perform initial LB
11 start lb_agent

```

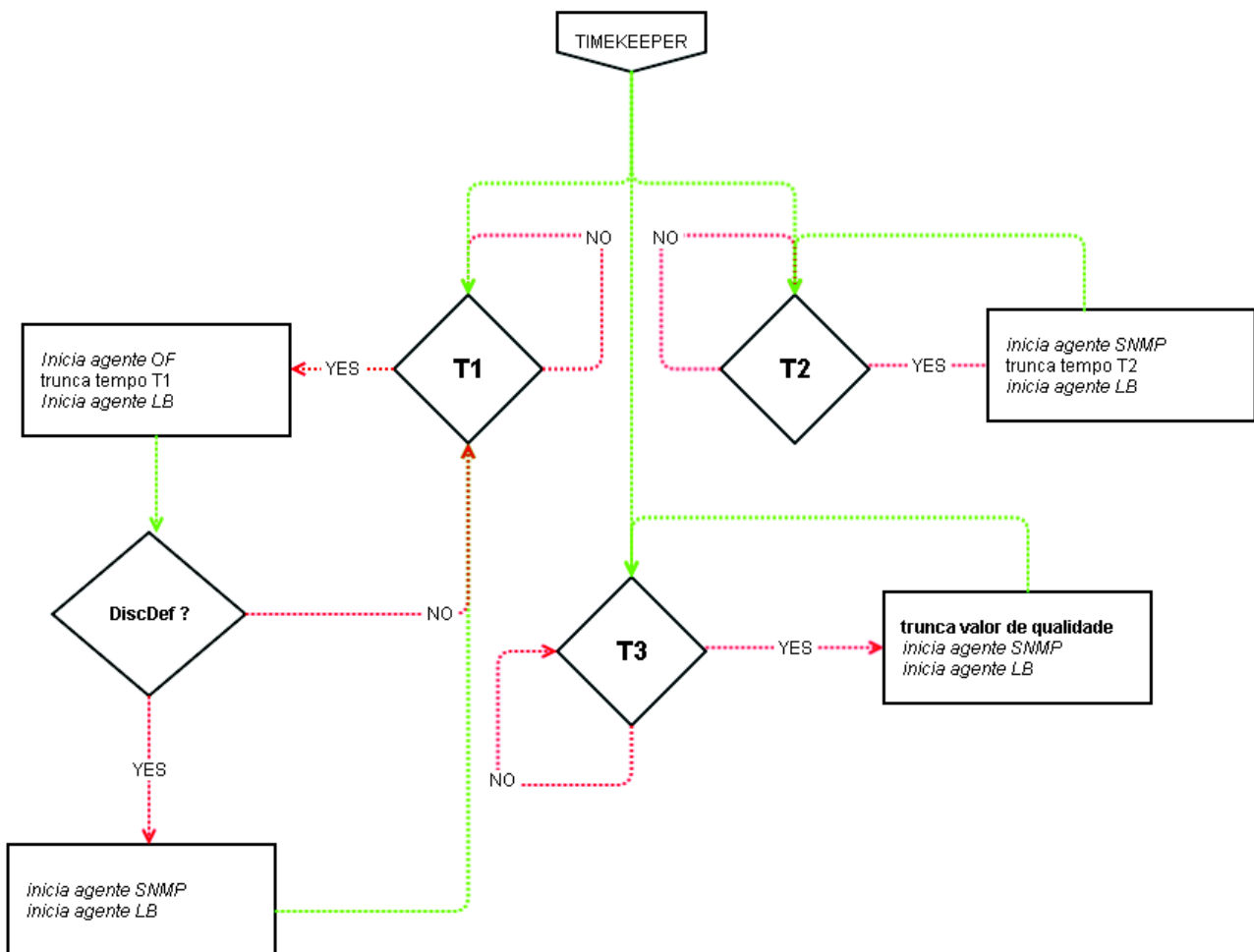
Após a definição da qualidade inicial e quem será o **NoAtv** para receber as requisições, o TimeKeeper inicializa e mantém a execução dos Agentes SNMP, OF e LB em intervalos de tempos predefinidos ou variáveis, durante sua execução. Os tempos variáveis, ocorrem devido a tomada decisões arbitrárias para igualar a qualidade de cada nó computacional em caso de discrepância. Os tempos de execução predefinidos são três: T1, T2 e T3; cada um deles tem uma tarefa específica conforme descrito abaixo:

1. **T1** é o tempo periódico de execução do agente OF;
2. **T2** é o tempo periódico de execução do agente SNMP;

3. **T3** é o tempo periódico de execução do truncamento da qualidade de cada nó computacional.

Os intervalos de tempo arbitrários são controlados pelas execuções do Agente OF em T1. O Timekeeper percebendo uma discrepância definida (**DiscDef**) entre os nós, executa o Agente SNMP para validar a correta qualidade computacional de cada nó. Esta execução no intervalo T1, que pertence ao Agente OF, induz a uma verificação pontual de qualidade. Após esta execução, através Agente SNMP, existindo uma discrepância ainda maior (**DiscMax**) entre a qualidade do nó com melhor e pior qualidade, é executado o Agente LB para efetuar a troca de tráfego. A figura 11 apresenta o diagrama de funcionamento do TimeKeeper com as ações executadas em cada tempo de monitoramento e a tomada de decisão para executar o monitoramento arbitrário no item **DiscDef** que faz a comparação da discrepância de qualidade entre o pior e melhor nó computacional.

Figura 11: TimeKeeper diagrama



A seguir estão detalhados o funcionamento dos três principais Agentes do *CoreLB*: OF, SNMP e LB. Estes agentes são responsáveis pela interação com os Atores para coleta de estatísticas e execução das tarefas de balanceamento.

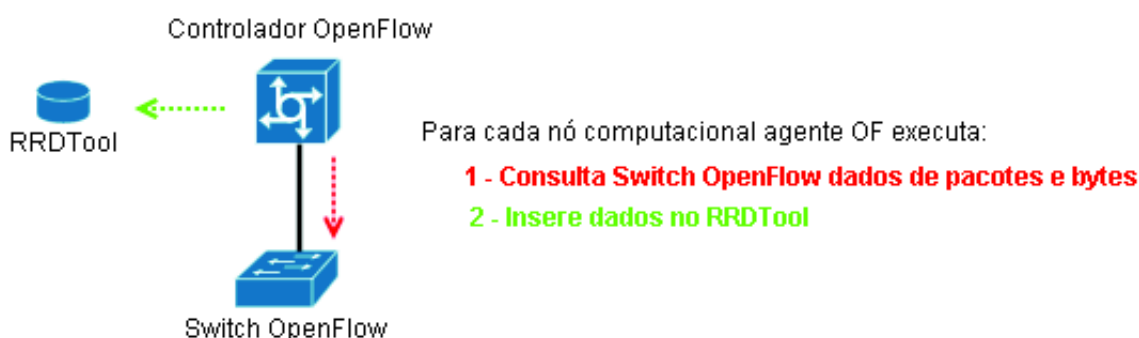
5.1.2 Agente OF

O agente OF é responsável pela integração com o *switch* OpenFlow. Ele conhece o protocolo de comunicação, proposto pelo OpenFlow, com o *switch* e realiza operações de leitura das estatísticas de rede das interfaces virtuais onde estão conectados os nós computacionais. Toda vez em que este agente é executado, ele solicita o número de pacotes e bytes transferidos e recebidos na interface onde cada servidor está conectado. Com esta informação é possível saber qual nó computacional teve o maior número de dados processados pois teve maior quantidade de tráfego na interface de rede. Após a coleta destes dados, a informação é armazenada no Agregador, executado pelo RRDTool, onde tem todo o histórico da estatística de tráfego armazenado.

Estas consultas realizadas no *switch* OpenFlow pelo Agente OF são menos custosas que as consultas realizadas pelo Agente SNMP. São apenas 2 consultas para cada nó computacional com um caminho de comunicação direto, pois, o controlador está conectado diretamente ao *switch* em uma interface dedicada, enquanto as consultas SNMP são em maior número e percorrem um caminho maior para comunicação pois precisam passar e ser processadas pelo *switch*. A figura 12, demonstra o fluxo das atividades realizadas pelo Agente OF, no mesmo canal de comunicação em que o controlador OpenFlow executa as atividades de administração do *switch*.

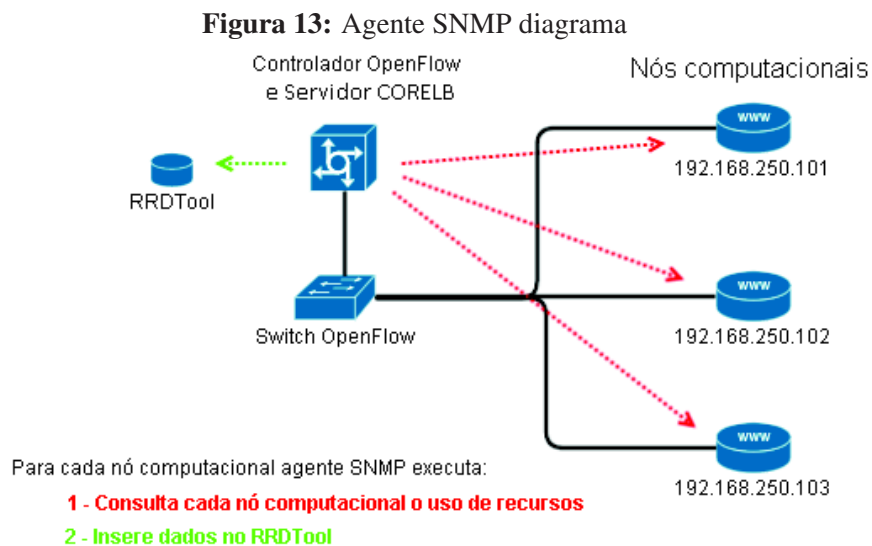
A primeira tarefa é receber a lista dos nós computacionais existentes através da previa configuração do arquivo de *profile* existentes no *CoreLB*. Além do endereço IP do nó computacional, é necessário saber a interface virtual que o servidor está conectado no *switch*, os dados estatísticos são armazenados nas interfaces e não no endereço IP. Com esta lista, o agente solicita para o *switch* as informações estatísticas sobre transmissão de pacotes e bytes de cada interface relacionada com os nós computacionais.

Figura 12: Agente OF diagrama



5.1.3 Agente SNMP

O agente SNMP é bastante semelhante ao agente OF, porém, o monitoramento é realizado nos nós computacionais e não no *switch* OpenFlow. Para realizar esta tarefa, o agente efetua a leitura do arquivo de *profile*, recebe a lista dos nós computacionais que necessitam ser monitorados e, toda vez que o agente é acionado, realiza a coleta das estatísticas dos equipamentos através do protocolo SNMP. As MIBs utilizadas no protótipo são de tipo padrão disponíveis juntamente com a instalação do serviço SNMP, porém não são estritas. As MIBs usadas nos testes do algoritmo do protótipo estão descritas na tabela 5. Os recursos escolhidos visam, principalmente, avaliar o uso de recursos de hardware com o intuito de estimar a carga de processamento de tarefas em cada nó. A figura 13 mostra o processo de comunicação realizado pelo agente SNMP com cada nó computacional e com o RRDTool.



Basicamente, as estatísticas das MIBs são consultadas e inseridas nas bases do RRDTool. A execução do agente SNMP não é apenas estática, o intervalo de consulta pode variar de acordo com a discrepância de qualidade dos nós computacionais: o agente SNMP é executado também intervalos **T1** toda a vez que o *threshold DiscDef* existir entre a qualidade de **NodeWorst** e **NodeBest**. Além desta execução, este agente tem uma execução estática em **T2** que é intencionalmente programada para acontecer indiferente discrepância de qualidade entre os nós computacionais.

As informações providas pelo SNMP são de dois tipos: **GAUGE**, contadores específicos que aumentam e diminuem ao longo do tempo e servem para expressar temperaturas, número de pessoas em uma sala ou velocidade de um carro. E **COUNTER**, um contador contínuo onde o número fica crescendo até ser truncado (geralmente com o reinício do serviço SNMP ou um overflow). O RRDTool é uma base capaz de lidar com estes tipos de dados e reconhecer quando, por exemplo, o contador foi truncado e organizar internamente os dados, da mesma forma que é possível controlar valores limiares. A tabela 5 classifica os tipos usados em cada

recurso monitorado juntamente com a MIB usada.

Tabela 5: Tipo de dados em cada recurso

RECURSO	TIPO	MIB
CPU User	COUNTER	.1.3.6.1.4.1.2021.11.50.0
CPU Nice	COUNTER	.1.3.6.1.4.1.2021.11.51.0
CPU System	COUNTER	.1.3.6.1.4.1.2021.11.52.0
CPU Idle	COUNTER	.1.3.6.1.4.1.2021.11.53.0
CPU Wait	COUNTER	.1.3.6.1.4.1.2021.11.54.0
CPU Kernel	COUNTER	.1.3.6.1.4.1.2021.11.55.0
CPU Interrupt	COUNTER	.1.3.6.1.4.1.2021.11.56.0
CPU SoftIRQ	COUNTER	.1.3.6.1.4.1.2021.11.61.0
Memória RAM total	GAUGE	.1.3.6.1.4.1.2021.4.5.0
Memória RAM usada	GAUGE	.1.3.6.1.4.1.2021.4.6.0
Memória RAM shared	GAUGE	.1.3.6.1.4.1.2021.4.13.0
Memória RAM buffer	GAUGE	.1.3.6.1.4.1.2021.4.14.0
Memória RAM cached	GAUGE	.1.3.6.1.4.1.2021.4.15.0
SWAP total	GAUGE	.1.3.6.1.4.1.2021.4.3.0
SWAP usado	GAUGE	.1.3.6.1.4.1.2021.4.4.0

5.1.4 Agente LB

O agente LB utiliza os dados previamente consultados por outros agentes e define **NodeWorst** e **NodeBest**, nó com pior qualidade e nó com melhor qualidade. O agente LB é inicializado após a execução de qualquer outro Agente, efetuando assim tarefas distintas para cada origem de execução, porém, semelhantes: definir o nó computacional que mais consumiu o recurso que esta sendo analisado e atribuir a **PontAtrib** para este nó. Por fim, o nó com maior pontuação é considerado o **NodeWorst** e com menor pontuação **NodeBest**.

O agente LB identifica os recursos monitorados por cada agente e conhece os nós computacionais através da leitura dos *profiles*. Outro ponto necessário em sua execução é determinar o Agente que executou anteriormente, assim sendo:

- Execução anterior do agente OF: é realizado a comparação dos valores coletados de dados de rede nas duas últimas consultas efetuadas. O agente LB vai comparar os dados de rede para atribuir **PontAtrib** para o nó computacional que tiver o maior número de pacotes e bytes processados com o intuito de reduzir a qualidade do nó computacional nos intervalos estáticos de **T1**.
- Execução anterior do agente SNMP: é realizado a comparação dos valores coletados nas duas últimas consultas de cada recurso monitorado em cada nó computacional encontrado no *profile*. O consumo de CPU, por exemplo, que foi coletado na execução anterior é comparado com a leitura atual para determinar se teve aumento ou não no consumo deste

recurso no intervalo que pode ser variado, menor ou maior de acordo com a discrepância de qualidade dos nós. A mesma comparação é realizada para todos os itens da tabela 4 que correspondem ao agente SNMP. O nó computacional que mais teve consumo no recurso recebe **PontAtrib**, alterando sua qualidade total.

Conforme observado, para todos os recursos monitorados a comparação é realizada da mesma forma, com o intuito de avaliar o total consumido de cada recurso no ultimo intervalo das coletas. Assim sendo, o calculo definido por $node(n) \rightarrow resource(agent).total = resource(agent).actual - resource(agent).old$ representa o consumo atual to recurso no intervalo. Este algoritmo está representado na figura 14.

Figura 14: Algoritmo agente LB

```

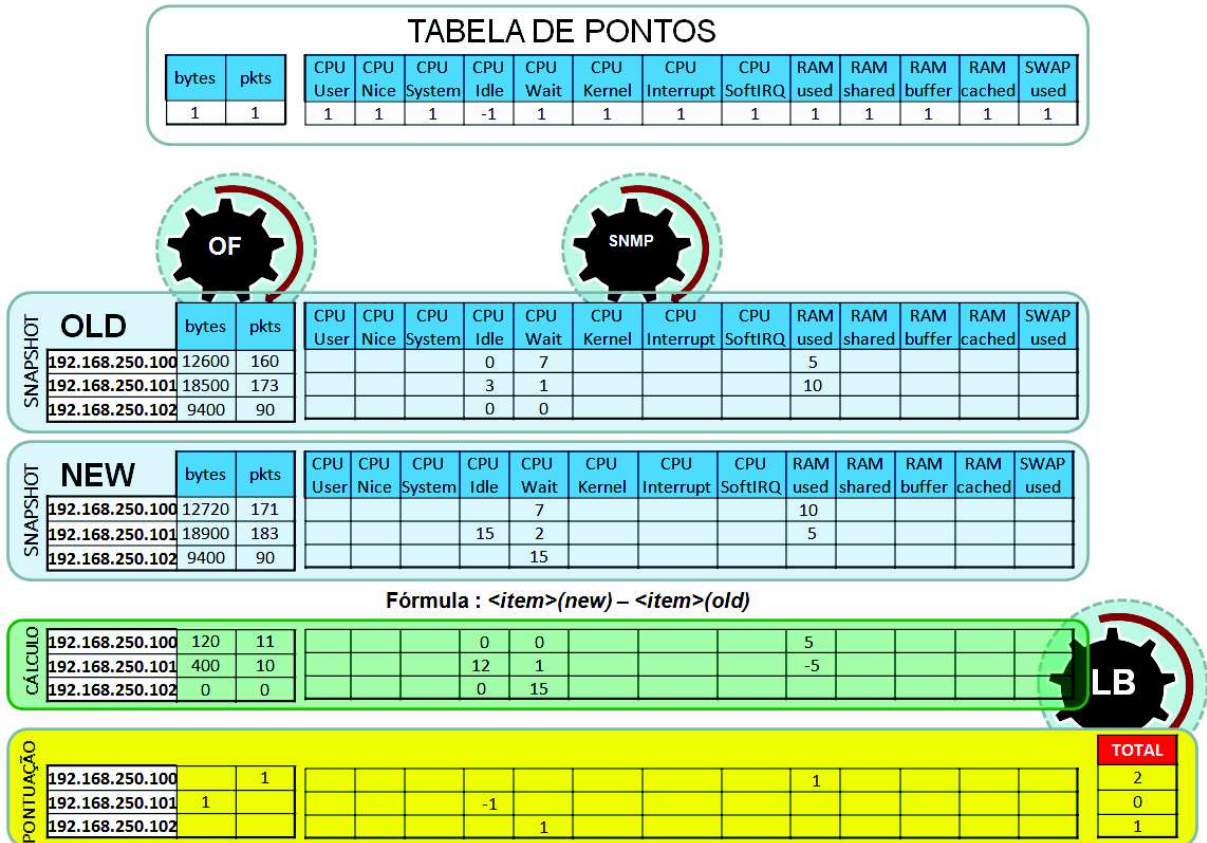
1  while Nodes{
2      if SNMP agent{
3          consulta CPU User consumed
4              Node(n)->resource(snmp).total = Resource(snmp).actual - Resource(snmp).old
5          consulta CPU System consumed
6              Node(n)->resource(snmp).total = Resource(snmp).actual - Resource(snmp).old
7          consulta CPU IDLE consumed
8              Node(n)->resource(snmp).total = Resource(snmp).actual - Resource(snmp).old
9          consulta CPU NICE consumed
10             Node(n)->resource(snmp).total = Resource(snmp).actual - Resource(snmp).old
11         consulta CPU WAIT consumed
12             Node(n)->resource(snmp).total = Resource(snmp).actual - Resource(snmp).old
13         consulta CPU KERNEL consumed
14             Node(n)->resource(snmp).total = Resource(snmp).actual - Resource(snmp).old
15         consulta CPU INTERRUPTED consumed
16             Node(n)->resource(snmp).total = Resource(snmp).actual - Resource(snmp).old
17         consulta CPU SOFTIRQ consumed
18             Node(n)->resource(snmp).total = Resource(snmp).actual - Resource(snmp).old
19         consulta MEMORY consumed
20             Node(n)->resource(snmp).total = Resource(snmp).actual - Resource(snmp).old
21         consulta SWAP consumed
22             Node(n)->resource(snmp).total = Resource(snmp).actual - Resource(snmp).old
23     }
24     if OF agent{
25         consulta BYTES consumed
26             Node(n)->resource(of).total = Resource(of).actual - Resource(of).old
27
28         consulta PACKETS consumed
29             Node(n)->resource(of).total = Resource(of).actual - Resource(of).old
30     }
31 }
32 }
33 if SNMP agent{
34     find NodeWorst of SNMP resource
35     NodeWorst = PontAtrib(r)
36 }
37 if OF agent{
38     find NodeWorst of OF resource
39     NodeWorst = PontAtrib(r)
40 }
41
42 find NodeWorst and NodeBest
43 if (NodeWorst - NodeBest) > DiscMax{
44     NoAtv = NodeBest
45     update OpenFlow(NoAtv)
46 }

```

5.2 Sistema de Pontuação

Cada um dos itens monitorados recebe uma pontuação chamada **PontAtrib**. O Agente LB atribui a pontuação aos nós com maior consumo de recursos. A figura 15 exemplifica o cálculo efetuado pelo agente LB baseado nos dados coletados pelo agente OF e SNMP. Os valores coletados na consulta anterior são comparados com a consulta atual e o nó computacional que teve maior consumo recebe a pontuação definida para o item específico. A tabela de pontos está definindo a mesma pontuação para todos os itens, porém esta pontuação pode ser diferenciada caso algum dos itens tenha maior importância na aplicação em execução. A pontuação ainda pode ser negativa, como CPU idle que significa recursos de CPU sem uso, com o intuito de reduzir a pontuação do nó computacional, aumentando a qualidade final do nó computacional.

Figura 15: Exemplo do cálculo de pontos



Esta pontuação é acumulativa até o momento em que todos os nós tem sua pontuação truncada em T3, conforme explicado na seção 5.1.1 na descrição do TimeKeeper.

6 RESULTADOS

Este capítulo está dividido em Metodologia de Avaliação e Resultados. Na metodologia está o conceito dos consumidores juntamente com sua eficácia na ação proposta, as técnicas de balanceamento comparadas com o *CoreLB* e o gerador de carga.

6.1 Metodologia de Avaliação

A metodologia de avaliação adotada entende que as pontuações de qualidade não devem divergir entre os nós computacionais ativos. Mantendo a pontuação de qualidade dos nós computacionais semelhante, faz-se o uso de cada nó computacional de forma semelhante, pois, as pontuações definidas são baseadas no consumo dos recursos utilizados. Garantindo o uso equilibrado dos recursos computacionais, o tempo de resposta ao cliente será o melhor, conforme esta avaliação demonstra.

Há algoritmos de balanceamento de carga que usam técnicas como *Time Slice Based Choice*: as requisições são divididas entre os servidores de maneira circular; *Random Time Slice and Choice*: as requisições são repassadas aos servidores de forma randômica; e *Load Based*: a requisição é repassada ao servidor com menor carga. O *CoreLB* é encaixado na categoria de *Load Based*, desta forma, a avaliação realizada compara o *CoreLB* com técnicas de *Time Slice Based Choice*, *Random Time Slice and Choice* e também a situações onde não é efetuado distribuição de carga. O processo de avaliação é baseado na comparação da eficiência entre os métodos de balanceamento. O melhor tempo de resposta para o usuário e a melhor distribuição de consumo de recursos operacionais define o método mais eficaz.

A avaliação dos resultados envolve a interpretação de dados quantitativos e qualitativos das situações obtidas nas ações de balanceamento de carga, considerando os distintos cenários com o uso de programas desenvolvidos que são chamados de Consumidores e que serão explicados a seguir. O universo heterogêneo de sistemas Web é, de certa forma, complexo de ser representado, assim sendo, o desenvolvimento de consumidores foi necessário para que os testes não se limitem a uma aplicação específica ou há um comportamento específico, tentando assim avaliar o balanceamento de carga de maneira mais ampla, por exemplo, avaliando se o balanceamento de carga em questão pode ser usado de maneira singular para aplicações distintas.

Inicialmente é explicado o que são os Consumidores e o leque de programas que o consiste com as características individuais de cada programa. Posteriormente, mais informações sobre os pontos avaliados e como foram avaliados serão apresentados.

6.1.1 Consumidores

Para avaliar o desempenho, foram desenvolvidos **Consumidores**, que são aplicações com diferentes propriedades de atuação para melhor representar o universo heterogêneo do compor-

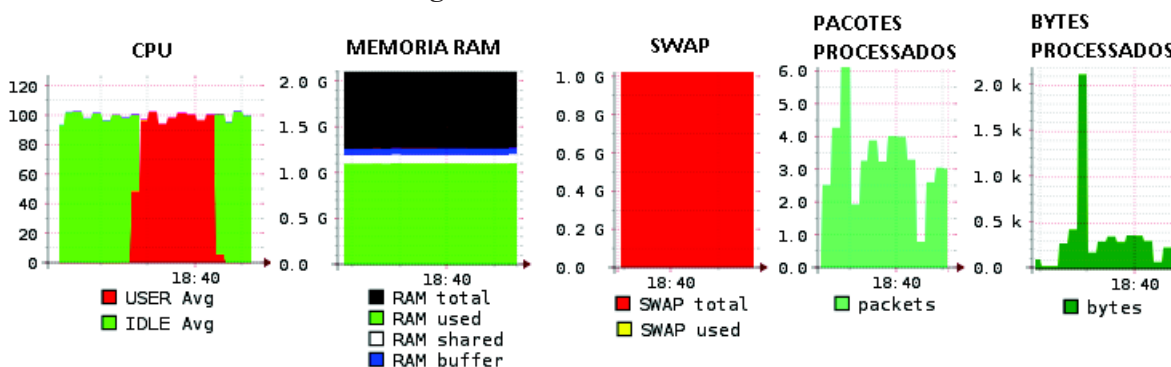
tamento de programas de Comércio Eletrônico, redes sociais ou qualquer outra aplicação Web. A seguir, estão explicados cada aplicação desenvolvida com suas características de comportamento representada em gráficos de monitoramento dos recursos computacionais como: CPU, memória física, swap, pacotes de rede e bytes de rede processados.

6.1.1.1 Consumidores de CPU

Para simular sistemas que efetuam consumo de CPU expressivo, foram desenvolvidos dois programas. O código destes programas executam cálculos em laço finito. O primeiro programa consome aproximadamente 1 segundo de CPU e o segundo aproximadamente 6 segundos de CPU nos servidores virtuais. Estes códigos foram desenvolvidos em linguagem C e são acionados a partir de uma requisição Web recebida pelo programa concentrador que será explicado posteriormente.

Nos gráficos apresentados na figura 16, observa-se o efeito que a execução deste consumidor causa no servidor . Estes consumidores se mostraram eficientes pois o acesso a esta aplicação causou consumo de CPU de *USER space* que são os processos executada pelo serviço Apache. Para representar este comportamento foi realizado solicitações de execução à este consumidor onde, durante aproximadamente 10 minutos, requisições foram enviadas ao nó computacional. Observa-se nos gráficos figura 16 que a execução do programa consumiu 100% dos recursos de CPU existentes.

Figura 16: Consumidores de CPU



Conforme observado, estes consumidores não alteram o consumo de memória do nó computacional e, com relação ao consumo de recursos de rede, observa-se picos menores de 2Kb/s. Este consumidor mostra-se eficiente para representar sistemas que consomem CPU de *user space* conforme observado na figura 16.

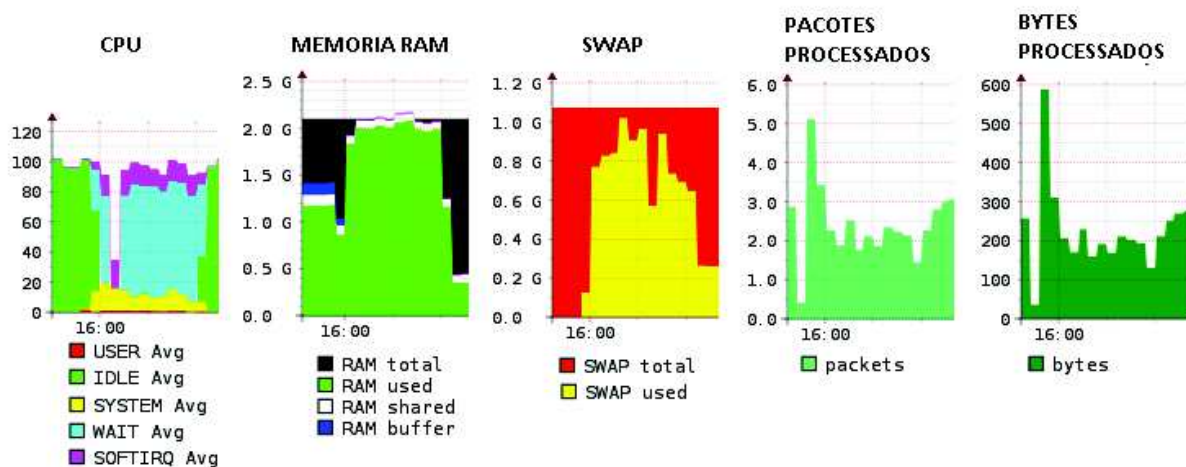
6.1.1.2 Consumidores de Memória

O consumidor de memória é um programa escrito em linguagem C que, em cada execução, faz a alocação de 1Gb de memória usando *malloc()*. Múltiplas execuções simultâneas deste pro-

grama no mesmo nó computacional podem levar a exaustão da memória disponível, neste caso, dependendo do Sistema Operacional e seus controles de exaustão de memória, é possível que o programa termine antes de conseguir alocar toda a memória desejada, porém, toda a memória disponível será consumida enquanto possível, o usuário, por outro lado, receberá o retorno HTTP 200 e não um erro por não conseguir alocar a memória desejada caso o consumidor seja terminado antes da alocação total de 1Gb.

Conforme observado na figura 17 uma requisição de consumo de memória não acarreta apenas no consumo do recurso de memória. Requisições *malloc()* também consomem CPU e solicitam IO para efetuar a operação, desta forma, percebe-se nos gráficos o aumento de CPU WAIT com porções de consumo de CPU para SOFTIRQ e também porções executadas em CPU SYSTEM. Os nós virtuais configurados com 2Gb de memória RAM tiveram o consumo total deste recurso e picos de 900Mb de SWAP neste exemplo de múltiplas execuções simultâneas deste consumidor durante 10 minutos com o intuito de demonstrar o funcionamento do programa. O Sistema Operacional Ubuntu, usado nestes testes, é dotado de mecanismos para prevenir exaustão total de memória, de outra forma o Sistema Operacional pode ser afetado ou mesmo travar com a inexistência de memória para ser consumida.

Figura 17: Consumidores de Memória



Este consumidor mostra-se eficiente para representar sistemas com grande consumo de memória conforme observado na figura 17. Não existe um consumo significativo de banda ou pacotes de rede pois o cliente efetua uma solicitação ao concentrador que executa o consumo de memória e quando o consumidor termina (tendo ou não sucesso na alocação da memória) o cliente recebe um retorno de solicitação concluída, sem transporte de dados significativos.

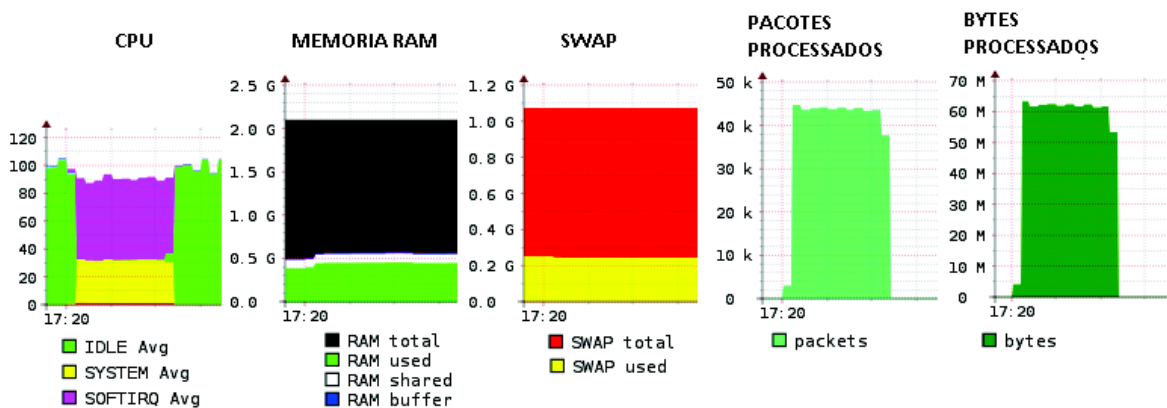
6.1.1.3 Consumidores de Rede

O consumidor de rede tem o intuito de consumir a maior quantidade possível de recursos de rede exaurindo a banda disponível ou mesmo levando a um gargalo na operação. Este consumi-

dor envia ao cliente um total de 3.1Mb de informação por solicitação de execução, assim sendo, quando o cliente solicita a execução deste consumidor, o retorno consome uma grande quantidade de banda de rede enquanto não concluído. A solicitação de execução deste consumidor gera uma grande quantidade de pacotes para serem transferidos através da rede, consequentemente uma grande quantidade de requisições para o adaptador de rede são criadas, ou seja, muitas solicitações de IO efetuadas. Requisições de IO tem prioridade de execução pois são interrupções de hardware e devem ser o mais rápidas o possível do ponto de vista de Sistema Operacional. Para que estas interrupções de hardware não sejam muito extensas, parte do trabalho é efetuado com o consumo de CPU SOFTIRQ que não é bloqueante, porém, ainda assim consomem CPU.

Na figura 18 é demonstrado o comportamento deste consumidor, durante a sua execução e é observado as requisições IO efetuadas e, consequentemente, o consumo de CPU SOFTIRQ que é diretamente usado neste processo. CPU SYSTEM também tem um consumo significativo pois são tarefas executadas pelo Sistema Operacional para alocar o hardware necessário. Nesta mesma análise, nota-se que o intuito principal de consumo da banda de rede foi atendido pois há o consumo constante de 63Mb de transferência durante aproximadamente 10 minutos de execução do consumidor. O número de pacotes transferidos está praticamente constante em 43000 pacotes por segundo. Por outro lado, não há alteração significativa de consumo de memória por parte deste consumidor.

Figura 18: Consumidores de Rede



O consumidor de rede projetado tem um algoritmo simples pois apenas transfere dados entre o cliente e o servidor, entretanto, mostrou-se eficiente para o propósito de exaurir recursos de rede e gerar requisições de interrupção de hardware conforme demonstrado na figura 18.

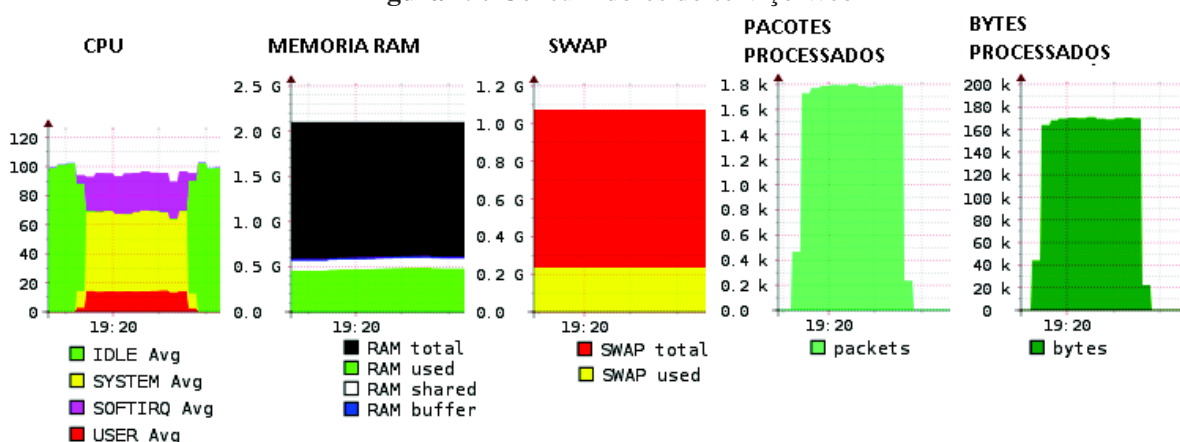
6.1.1.4 Consumidores do serviço Web

O consumidor de serviço Web não é exatamente um programa desenvolvido para executar alguma tarefa mas apenas uma resposta a solicitação. A solicitação é encaminhada pelo cliente ao concentrador que retorna um HTTP 200. O cliente envia a requisição GET que é atendida

pelo serviço Apache e o entregue ao concentrador, nenhum processamento adicional é efetuado, desta forma, apenas o serviço Apache executa processamento no nó computacional. O objetivo deste consumidor é estressar ao máximo com solicitações leves aumentando o número de requisições simultâneas aumenta.

A figura 18 demonstra o aumento no número de requisições simultâneas pois o programa consumiu em torno de 18000 pacotes por segundo. A ação deste consumidor não realiza alocação de memória e observa-se o consumo de CPU USER que é a gama de CPU usada pelo serviço do Apache. O alto número de requisições simultâneas também gerou consumo de SOFTIRQ e SYSTEM pois há muitas requisições de IO para cada solicitação recebida pelo cliente.

Figura 19: Consumidores do serviço Web



O consumidor Web é eficiente para o propósito de estressar o serviço Apache.

6.1.1.5 Concentrador

O concentrador é um código escrito em PHP e interpretado pelo serviço Apache. Este código tem o conhecimento de todos os consumidores possíveis de execução, assim sendo, a cada requisição recebida um consumidor é escolhido para ser executado. Os consumidores são escolhidos aleatoriamente pelo concentrador a cada requisição recebida. A figura 20 apresenta a simplificação do código do concentrador que é o ponto central para a execução dos consumidores de acordo com uma escolha aleatória realizada.

A seguir está apresentado as técnicas de balanceamento de carga implementadas para efetuar a comparação com o funcionamento do *CoreLB*.

6.1.2 Técnicas de balanceamento comparadas

O balanceamento de carga consiste em distribuir as requisições solicitadas pelos usuários entre os nós computacionais disponíveis. Todas as técnicas utilizadas nesta avaliação foram implementadas no *switch* OpenFlow, realizando a troca de fluxo das requisições usando algum

Figura 20: Algoritmo Concentrador

```

1  <?php
2  $chose = rand(1, 5);
3
4  switch ($chose){
5      case 1:
6          system("./consumeCPU1s");
7          break;
8      case 2:
9          system("./consumeCPU6s");
10         break;
11         case 3:
12             break;
13         case 4:
14             system("./memory");
15             break;
16         case 5:
17             include("bigtext.php");
18             break;
19     }
20  ?>

```

algoritmo que represente ou se assemelhe a definição da técnica. As técnicas de balanceamento de carga, que foram comparadas com o *CoreLB*, estão relacionadas abaixo com a devida explicação de como foi implementada.

6.1.2.1 Time Slice Based Choice

Neste algoritmo, a troca de tráfego é realizada a cada 60 segundos seguindo a listagem sequencial dos nós computacionais disponíveis, ao término, o ciclo se inicia novamente pelo primeiro. Inicialmente as conexões vão para o nó computacional número 1, após 60 segundos para o nó computacional número 2 e assim por diante. Na figura 21 há um exemplo de como pode ser implementado esta técnica. Controles da disponibilidade do nó foram abstraídos da figura 21 para fins didáticos, porém, a técnica Time Slice Based Choice não realiza avaliações precisas para analisar a disponibilidade e capacidade de processamento do nó computacional.

Figura 21: Algoritmo Time Slice Based Choice

```

1  while true
2  do
3      loadbalance node1
4      sleep 60
5      loadbalance node2
6      sleep 60
7      loadbalance node3
8      sleep 60
9  done;

```

6.1.2.2 Random Time Slice and Choice

O nó computacional é escolhido aleatoriamente de acordo com o número de nós. O algoritmo entra em uma situação de espera por um tempo que é definido de forma aleatória entre

0 e 300 segundos e novamente é escolhido randomicamente um próximo nó computacional. A figura 22 mostra um exemplo da construção deste algoritmo.

Figura 22: Algoritmo Random Time Slice and Choice

```

1 while true
2 do
3     node=$((RANDOM%<numero_de_nos>+1))
4     loadbalance $node;
5     chose=$((RANDOM%300))
6     sleep $chose
7 done;
```

6.1.2.3 CoreLB

O modelo de balanceamento adotado pelo *CoreLB* está explicado na seção Implementação. Este algoritmo é considerado Load Based pois avalia a situação de carga dos nós computacionais e a situação de rede para tomada de decisão de balanceamento. Os tempos de troca de trafego são dinâmicos pois dependem da situação da discrepância de qualidade encontrada entre os nós computacionais.

6.1.2.4 Com apenas um servidor

Os testes realizados neste cenário não foram efetuados propriamente com a distribuição de requisições entre os nós computacionais disponíveis. Neste cenário todas as requisições foram direcionadas para apenas um nó computacional. O objetivo deste cenário é observar o comportamento do nó computacional sem o auxílio do balanceamento de carga para proporcionar mais eficiência no processamento das requisições.

6.1.3 Gerador de carga

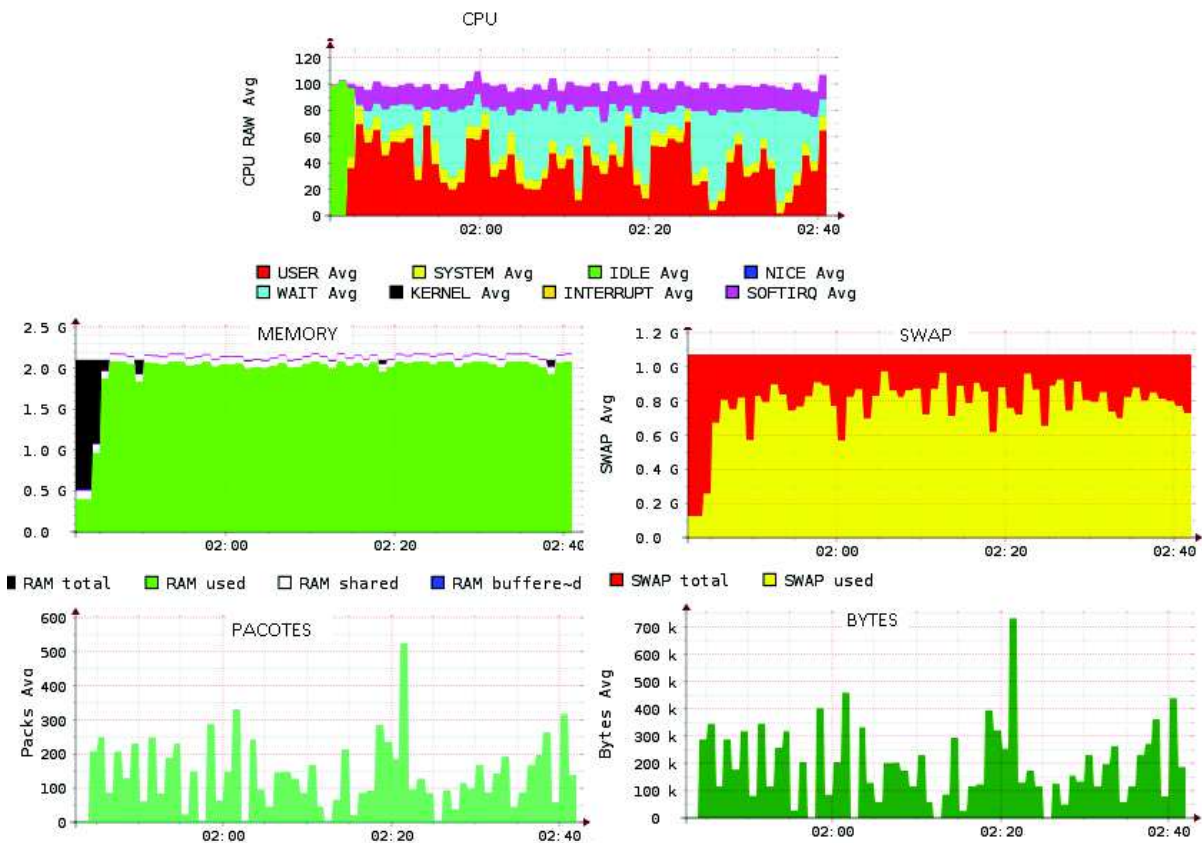
O gerador de carga é responsável pelo processo de gerar demanda de requisições e medir os tempos de resposta final. Este componente é executado para determinar o comportamento do sistema usando as diferentes técnicas de balanceamento de carga e avaliar a capacidade de cada técnica em melhorar a situação para cada cenário. O OpenLoad foi a ferramenta escolhida para executar esta tarefa pois atende a necessidade de gerar a demanda de requisições e interpretar os resultados de forma simples, retornando o número de transações por segundo processadas (TPS), tempos máximos e mínimos de resposta bem como gerar carga simulando um número distinto de clientes simultâneos.

Na figura 23 há uma representação gráfica do comportamento de um único nó computacional recebendo requisições geradas pelo OpenLoad simulando 5 clientes simultâneos durante 1 hora de execução. Os consumidores mostraram-se eficientes para estressar um nó computacional.

Tabela 6: Número de testes realizados com o OpenLoad e configuração de clientes

	Um servidor	TSBC	RTSC	CoreLB
1 cliente	1	1	1	1
3 clientes simultâneos	2	2	2	2
5 clientes simultâneos	4	4	4	4
8 clientes simultâneos	1	1	1	1
10 clientes simultâneos	1	1	1	1
Tempo de Execução	1 hora	1 hora	1 hora	1 hora

cional consumindo diversos recursos. A memória foi consumida e o swap usado para realizar paginação, o consumo de CPU também é constante principalmente com USER CPU que é a execução de processos de usuário, a fatia de SOFTIRQ mostra que o número de requisições IRQ foram frequentes consumindo praticamente 20% do tempo de CPU total.

Figura 23: Comportamento de um nó computacional recebendo toda a carga gerada pelo OpenLoad

Na tabela 6 está apresentado a distribuição de testes realizados com o OpenLoad. Os cenários são: com apenas um servidor, Time Slice Based Choice (TSBC), Random Time Slice and Choice (RTSC) e CoreLB. Para cada número de clientes simultâneos foram executados quantidades diferentes de testes, por exemplo, para execução com três e cinco clientes foram executadas duas e quatro baterias de testes respectivamente. A execução para cada cenário correu durante 1 hora de geração de carga.

Os resultados obtidos nas baterias de testes estão apresentados na seção de Avaliação dos Resultados onde, informações detalhadas, tempos de resposta e qualidade do balanceamento de carga estão explanados.

6.2 Avaliação dos Resultados

Durante a execução das baterias de testes as informações coletadas para análise foram representadas graficamente e também numericamente com dados estatísticos fornecidos pelas ferramentas RRDTool e OpenLoad. Os dados avaliados podem ser sumarizadas nos seguintes itens:

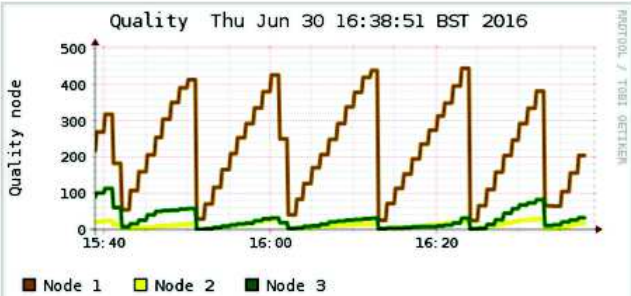
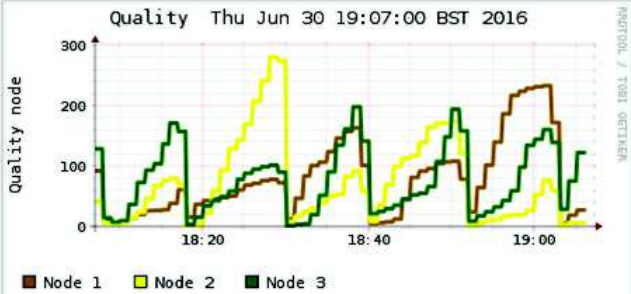
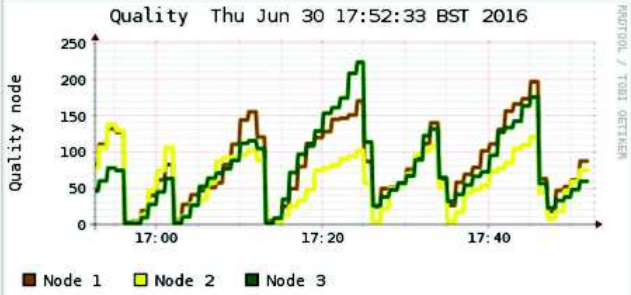
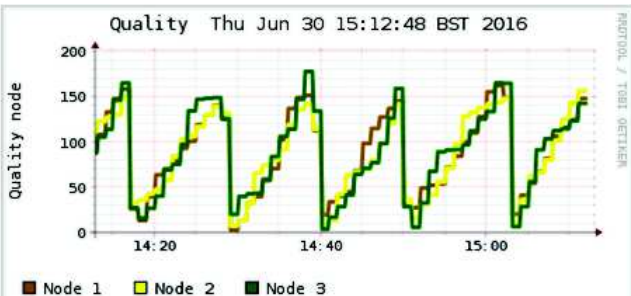
1. a pontuação atribuída pelo algoritmo de definição de qualidade do *CoreLB*. Durante todas as baterias de testes realizados o algoritmo de pontuação foi executando, entretanto, a decisão de balanceamento de carga foi executada pelo algoritmo da técnica avaliada e não pelo *CoreLB*. Desta forma, o algoritmo de definição de qualidade foi executado para avaliar este indicador sem interferir na execução de balanceamento da técnica sendo avaliada;
2. o número de transações processadas pelo gerador de carga juntamente com o respectivo tempo de resposta. Os dados estatísticos do OpenLoad são bastante úteis, pois, além de disponibilizar os tempos em execuções individuais, ainda fornecem as médias gerais no final da execução da bateria programada;
3. o *load average* de cada nó computacional é um indicador existente em Sistemas Operacionais Linux e representa os recursos consumidos no sistema. O calculo de carga é realizado através de um algoritmo do próprio Kernel do sistema avaliado pelo número de processos concorrendo por CPU.

Nas seções a seguir cada um dos três itens é detalhado com a devida análise técnica do comportamento observado com os dados que foram coletados durante as baterias de testes.

6.2.1 Pontuação definida pelo CoreLB algoritmo de qualidade

Os resultados referente à pontuação de qualidade de cada nó computacional estão apresentados na tabela 7. São dados gerados pelo algoritmo de calculo de pontuação do *CoreLB* durante a execução de cada uma das técnicas de balanceamento comparadas. Esta pontuação foi atribuída para cada nó computacional durante a execução de uma bateria de testes de uma hora com cada técnica de balanceamento para avaliar o comportamento e o desempenho de cada nó computacional. Um teste com apenas um servidor foi executado para comparar a execução de carga com apenas um nó computacional e a taxa de melhoria usando o balanceamento entre mais nós com o uso dos previamente mencionados algoritmos de balanceamento. Neste comparativo de

Tabela 7: Resultados de qualidade de nós computacionais

Gráfico de monitoramento	Descrição
 <p>Quality Thu Jun 30 16:38:51 BST 2016</p> <p>Quality node</p> <p>15:40 16:00 16:20</p> <p>Node 1 Node 2 Node 3</p> <p>Drawn at Thu 30 Jun 2016 16:38:00 BST</p> <p>1 min: 24.08 1 max: 444.30 1 ave: 224.31 2 min: 0.75 2 max: 28.84 2 ave: 11.45 3 min: 0.00 3 max: 112.83 3 ave: 27.56</p>	<p>Com apenas um servidor</p> <p>Neste ciclo de teste realizado, não existia balanceamento, todos as requisições estavam sendo direcionadas para o nó 1. Observa-se a qualidade do nó 1 piora constantemente enquanto o restante permanece com qualidade superior. A média de qualidade do nó 1 ficou em 224 pontos com pico máximo de 444 pontos.</p>
 <p>Quality Thu Jun 30 19:07:00 BST 2016</p> <p>Quality node</p> <p>18:20 18:40 19:00</p> <p>Node 1 Node 2 Node 3</p> <p>Drawn at Thu 30 Jun 2016 19:06:00 BST</p> <p>1 min: 4.54 1 max: 233.00 1 ave: 78.70 2 min: 0.00 2 max: 280.21 2 ave: 71.03 3 min: 1.00 3 max: 197.73 3 ave: 76.36</p>	<p>Random Time Slice and Choice</p> <p>Nesta situação percebe-se mais claramente nós computacionais consumindo muito mais recursos que outros, porém, a média de qualidade ficou muito mais semelhante entre os nós: 78, 71 e 76. Entretanto, o nó 2 teve um pico onde a qualidade chegou a 280 pontos, enquanto o restante ficou bem inferior.</p>
 <p>Quality Thu Jun 30 17:52:33 BST 2016</p> <p>Quality node</p> <p>17:00 17:20 17:40</p> <p>Node 1 Node 2 Node 3</p> <p>Drawn at Thu 30 Jun 2016 17:52:00 BST</p> <p>1 min: 3.12 1 max: 196.64 1 ave: 83.10 2 min: 2.01 2 max: 137.94 2 ave: 60.54 3 min: 2.00 3 max: 223.62 3 ave: 76.94</p>	<p>Time Slice Based Choice</p> <p>A média de qualidade dos 3 nós ficaram bastante discrepantes: 83, 60 e 76. Nota-se momentos em que a qualidade estava equilibrada, porém, outros momentos em que nós estavam consumindo muito mais recursos que outros. A pontuação máxima também teve uma discrepância ainda maior, o que provo uma execução muito mais intensa em alguns momentos em nós específicos.</p>
 <p>Quality Thu Jun 30 15:12:48 BST 2016</p> <p>Quality node</p> <p>14:20 14:40 15:00</p> <p>Node 1 Node 2 Node 3</p> <p>Drawn at Thu 30 Jun 2016 15:12:00 BST</p> <p>1 min: 2.80 1 max: 161.98 1 ave: 87.59 2 min: 7.29 2 max: 156.13 2 ave: 86.64 3 min: 3.58 3 max: 177.57 3 ave: 88.32</p>	<p>CoreLB</p> <p>A discrepância média não foi maior que 2 pontos, o que é um bom indicador para a equivalência do uso de recursos dos nós computacionais. Percebe-se que os 3 nós tiveram uma pontuação semelhante o tempo todo, fato este que prova a eficiência do TimeKeeper na tarefa de equilibrar a qualidade dos nós computacionais.</p>

qualidade computacional de cada nó, a menor discrepância na pontuação significa que os nós computacionais está atendendo uma demanda de solicitações semelhantes e, conseqüentemente, consumindo quantidade semelhante de recursos.

A tabela 7 também apresenta uma descrição técnica do que pode ser observado em cada uma das baterias de teste. É importante salientar que, segundo o algoritmo de cálculo de pontuação do *CoreLB*, em T3 a qualidade dos nós computacionais é truncada e o cálculo é recommçado com todos os nós tendo a mesma pontuação. Nas baterias de testes realizadas, o T3 estava definido em 300 segundos, momento em que a qualidade de todos os nós computacionais volta ao ponto inicial. Os gráficos da tabela 7 também apresentam a pontuação média (avg) recebida durante o tempo de execução e a pontuação máxima (max) recebida que são importantes itens para analisar o comportamento independente de cada nó computacional.

Além da análise sumarizada na tabela 7, é possível salientar como avaliação técnica os seguintes pontos:

- Na bateria realizada **com apenas um servidor**, o nó computacional 1 recebeu todas as requisições e teve a máxima pontuação que chegou há 444 pontos em 300 segundos. Comparando as três técnicas de balanceamento de carga, os tempos máximos e médios demonstram que o *CoreLB* teve as maiores médias (avg), porém, também teve menores máximas (max);
- O mais importante na análise de pontuação de qualidade é avaliar a discrepância na pontuação entre nós computacionais pois, uma discrepância grande significa que nós com maior pontuação tem um consumo maior de recursos computacionais. Um exemplo claro do problema de discrepância ocorre na bateria de testes com a técnica **Random Time Slice and Choice**, onde, por volta de 18:30 o nó número 2 recebeu 280 pontos, que representa 63% do máximo de 444 pontos recebidos pelo nó 1 na bateria de testes **com apenas um servidor**;
- Observa-se que a bateria de testes com **Time Slice Based Choice** o nó computacional 1 teve a média maior que o restante, entende-se que este nó teve mais consumo que os outros de uma maneira geral e, desta forma, as requisições processadas por este nó provavelmente tiveram o desempenho prejudicado por haver maior concorrência para alocação de recursos.

6.2.2 Resultados estatísticos do OpenLoad

Os gráficos de pontuação de qualidade atribuída pelo *CoreLB* por si só não demonstram a eficiência da técnica de balanceamento de carga, para isso as estatísticas fornecidas pelo gerador de carga OpenLoad complementam a análise estruturada e fornecem os resultados do ponto de vista de usuários acessando o serviço. A tabela 8 apresenta a média de resultados de todas as

execuções das baterias de testes conforme apresentadas no seção 6.1. Os resultados apresentados são referentes a todos os testes realizados conforme apresentado na tabela 6. Cada bateria de teste teve um resultado único para cada elemento apresentado na tabela 8, estes resultados estão apresentados representam a média obtida nestas execuções.

Tabela 8: Resultados da avaliação dos métodos

MÉTODO	TPS	AvgRT	MaxRT	Requests
Sem Load Balance	0,332	14,021s	78,157s	1177
Random Time Slice and Choice	0,376	12,502s	83,592s	1358
Randon Robin	0,373	11,746s	85,74s	1349
CoreLB	0,398	11,426s	101,118s	1446

Os resultados apresentados na tabela 8 demonstram o comportamento do balanceamento de carga do ponto de vista dos usuários, ou seja, representa os tempos que os usuário obtiveram em cada solicitação realizada para o serviço. O número de transações por segundo (**TPS**) revela que um único nó computacional foi capaz de executar 0,332 transações por segundo e, nesta mesma análise, percebe-se um aumento significativo com a adição de dois nós computacionais e a execução de balanceamento de carga que, nos testes realizados e com os consumidores desenvolvidos, possibilitou um aumento entre 12% e 19% da capacidade de processamento. Considerando que todos os testes tiveram o mesmo período de execução é possível verificar que o *CoreLB* atendeu mais requisições que o restante da técnicas com um total médio de 1446 requisições.

O tempo médio de resposta (**AvgRT**), calculado pelo OpenLoad, demonstra que o *CoreLB* teve os melhores tempos com 11,426s na média de todas as baterias de testes executadas, mesmo tendo processado requisições com o maior tempo de resposta (**MaxRT**) de 101,118s. O indicador **MaxRT** apenas representa a requisição que teve o maior tempo de resposta, ou seja, que mais demorou para ser atendida. Na tabela 8 o tempo médio é mais relevante para demonstrar o desempenho da técnica de balanceamento pois é uma estatística calculada a partir dos resultados de todas as requisições.

Os dados da tabela 8 representa o item número dois da relação de informações analisadas.

6.2.3 Resultados de *Load Average*

Para completar a análise proposta na seção 6.2, o terceiro e último item está apresentado nas três próximas figuras: **24** representando o metodo **Random Time Slice and Choice**, **25** representando o método **Time Slice Based Choice** e **26** representando o **CoreLB**. Estas representações gráficas representam o item *load average* gerado por cada um dos nós computacionais durante a bateria de avaliação de cada técnica de balanceamento. Os dados de *load* estão apresentados através de três contadores: **1 minuto de carga:** Estatísticas do último minuto de carga; **5 minutos de carga:** Estatísticas do último 5 minutos de carga; **15 minutos de carga:** Estatísti-

Tabela 9: Comparativo do valor de consumo médio

	load 1 min	load 5 min	load 15 min
Random Time Slice and Choice Method	1,48	1,55	1,30
Time Slice Based Choice Method	1,67	1,65	1,55
CoreLB Method	1,74	1,72	1,65

cas do último 15 minutos de carga; Estes contadores numéricos representam o estado da fila de processamento do sistema, se o sistema estiver completamente inativo (*idle*) o *load* deve ficar em zero.

Nas figuras representativas da situação de *load* de cada método de balanceamento de carga avaliado, existente, logo abaixo de cada conjunto de gráficos que representam os três nós computacionais, os dados estatísticos que representa os valores médios e máximos destes contadores. Estes dados foram coletados remotamente, com consultas regulares, através do protocolo SNMP. Para cada método avaliado há um conjunto com o *load* médio coletado durante o intervalo análise e também o *load* máximo coletado para cada um dos nós computacionais em uso. As cores desta tabela representam a linha dos gráfico, 1 minuto, 5 minutos e 15 minutos respectivamente.

O objetivo principal da análise deste indicador está em perceber como estava a concorrência de processos no Sistema Operacional durante a execução dos testes, tendo em vista que, este indicador é calculado e definido pelo próprio Kernel do Sistema Operacional. A análise deste indicador deve ser realizada de forma comparativa e a melhor situação de balanceamento de carga entende-se quando os valores de carga assemelham-se entre os distintos nós computacionais. Cada um dos nós computacionais estava sendo executado com um único processador dedicado, desta forma, o ideal é que o valor de *load* não ultrapasse de 1. Resultados até 1, significam que no momento havia não mais de 1 requisição necessitando de CPU. Valores maiores que não representam necessariamente um excesso de carga pois o Sistema Operacional é multi-tarefas então pode-se admitir valores superiores. Entretanto, quando mais acima deste valor, entende-se que o nó computacional está ocupado e enfileirando requisições.

Devido ao valor médio de carga representar a média de baterias de uma hora de testes, é importante também avaliar os valores máximos. Pode haver situações em que um nó computacional está com muito mais carga que outro e, desta forma, o cálculo final de média não vai representar a verdadeira situação do ambiente, pois o valor baixo de um nó computacional compensa o valor elevado de outro. Por mais que o valor de carga médio de um método seja inferior à outro, não significa que o método teve melhor desempenho. É necessário analisar os valores máximos na comparação. Entende-se que valores máximos menores representem uma situação mais eficiente de carga pois o sistema teve, nos piores picos de carga, menor processamento.

Para sintetizar os resultados obtidos nas figuras representativas do *load* coletado, os dados também estão apresentados na tabela 9 que representa o consumo médio obtido e na tabela 10 que representa o consumo máximo. Nestas tabelas, é possível comparar de maneira direta as

Tabela 10: Comparativo do valor de consumo máximo

	load 1 min	load 5 min	load 15 min
Random Time Slice and Choice Method	5,35	3,46	2,35
Time Slice Based Choice Method	4,57	2,65	2
CoreLB Method	3,96	2,51	1,99

estatísticas do valor de *load* obtidos na execução de cada um dos métodos de balanceamento. Percebe-se os resultados inversos entre o consumo médio e consumo máximo no comparativo de cada método de balanceamento.

Com a avaliação do comportamento plotado nos gráficos das figuras 24, 25 e 26 observa-se informações que não são possíveis de avaliar apenas analisando dados finais numéricos, como por exemplo, os momentos em que nós computacionais estavam sem atividade de processamento. A próxima relação de itens demonstra pontos possíveis de salientar com a observação dos dados numéricos finais e também da observação do comportamento de cada nó computacional nos gráficos:

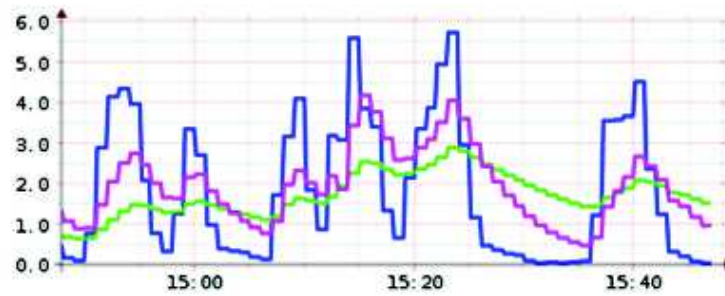
- A técnica Random Time Slice and Choice de balanceamento cria lapsos de inutilização de recursos enquanto, em contrapartida, outros nós computacionais ficam sobrecarregados de acordo com o que se consegue ver no indicadores de consumo máximo;
- O método de Time Slice Based Choice tem um comportamento mais equilibrado no uso dos recursos, mas os picos de **consumo máximo** continuam sendo altos conforme observado também na técnica randômica;
- O *CoreLB* teve as maiores médias no item de **consumo médio**, entretanto, teve os menores **consumos máximos**. Os picos de carga máximo foram menores nesta técnica pois todos os nós computacionais estavam sendo usados de maneira uniforme, conforme o objetivo principal do *CoreLB*. Desta forma, entende-se que a distribuição foi mais adequada evitando picos discrepantes de consumo em nós específicos enquanto outros não estão sendo utilizados.

Figura 24: Monitoramento LOAD durante o uso do método Random Time Slice and Choice

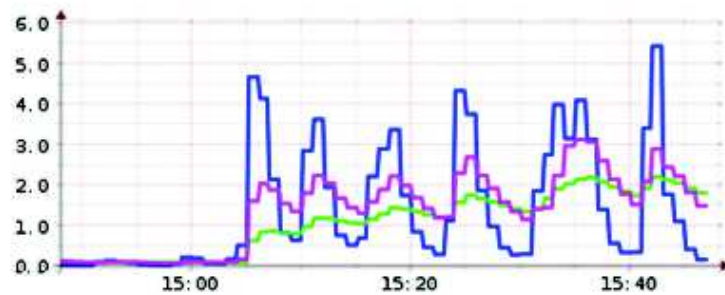
Random Time Slice and Choice

■ 1 Min Load Avg ■ 5 Min Load Avg ■ 15 Min Load Avg

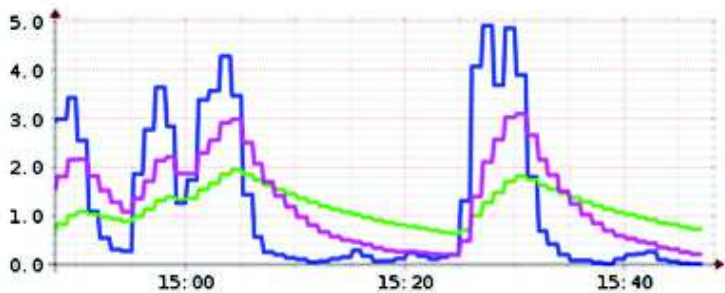
192.168.240.100 - NODE 1



192.168.240.101 - NODE 2



192.168.240.102 - NODE 3



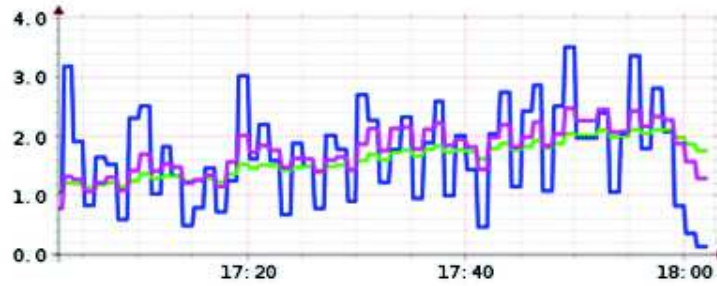
		192.168.240.100	192.168.240.101	192.168.240.102	MEDIA
consumo médio	1 min	1,81	1,38	1,24	1,48
	5 min	1,99	1,36	1,31	1,55
	15 min	1,72	1,07	1,118	1,30
consumo máximo	1 min	5,73	5,42	4,91	5,35
	5 min	4,17	3,12	3,09	3,46
	15 min	2,89	2,2	1,95	2,35

Figura 25: Monitoramento LOAD durante o uso do método Time Slice Based Choice

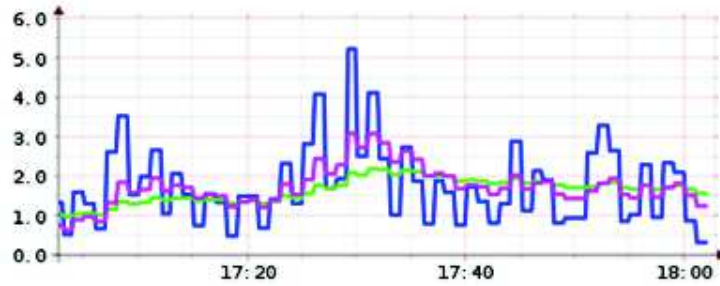
Time Slice Based Choice

■ 1 Min Load Avg ■ 5 Min Load Avg ■ 15 Min Load Avg

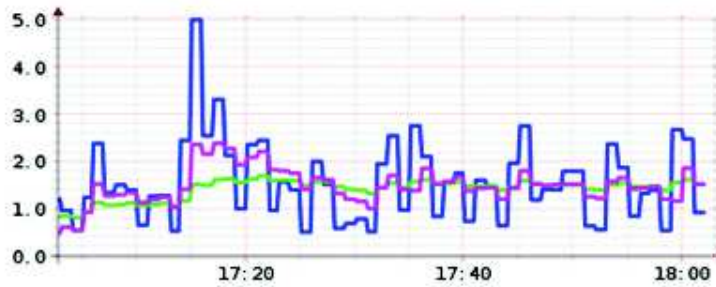
192.168.240.100 - NODE 1



192.168.240.101 - NODE 2



192.168.240.102 - NODE 3



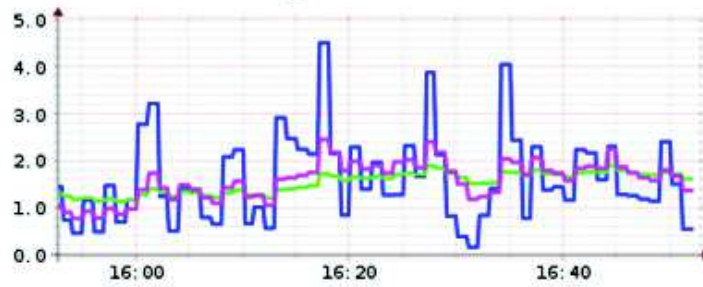
		192.168.240.100	192.168.240.101	192.168.240.102	MEDIA
consumo médio	1 min	1,7	1,76	1,55	1,67
	5 min	1,75	1,73	1,48	1,65
	15 min	1,61	1,63	1,4	1,55
consumo máximo	1 min	3,51	5,22	4,99	4,57
	5 min	2,47	3,09	2,38	2,65
	15 min	2,12	2,18	1,69	2,00

Figura 26: Monitoramento LOAD durante o uso do método CoreLB

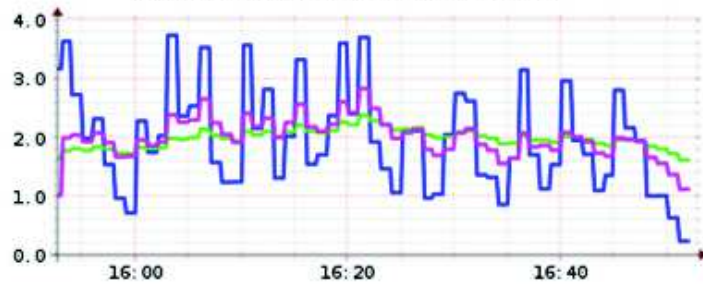
CORELB METHOD

■ 1 Min Load Avg ■ 5 Min Load Avg ■ 15 Min Load Avg

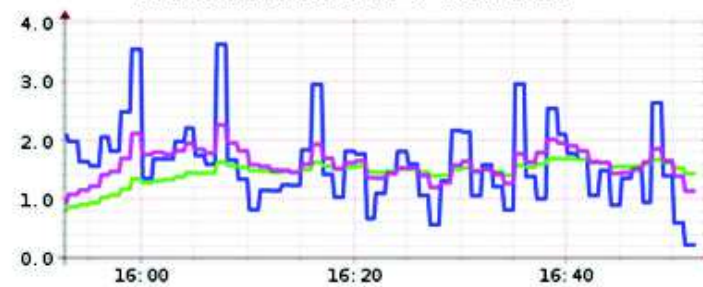
192.168.240.100 - NODE 1



192.168.240.101 - NODE 2



192.168.240.102 - NODE 3



		192.168.240.100	192.168.240.101	192.168.240.102	MEDIA
consumo médio	1 min	1,61	1,99	1,61	1,74
	5 min	1,58	2	1,59	1,72
	15 min	1,53	1,98	1,45	1,65
consumo máximo	1 min	4,51	3,73	3,63	3,96
	5 min	2,45	2,83	2,26	2,51
	15 min	1,9	2,37	1,69	1,99

7 CONCLUSÃO

Com os atrativos do modelo proposto por SDN, do bom desempenho da tecnologia OpenFlow demonstrados por trabalhos relacionados, e a capacidade de realizar balanceamento de carga de forma nativa na rede, removendo a necessidade de inclusão de hardware ou serviços adicionais para a realização desta tarefa, a metodologia apresentada mostra-se uma excelente proposta para execução de balanceamento de carga simples, em que a utilização do controle de contexto não se faz necessária. O processo de implementação desta tarefa é baseado na inclusão e alteração de regras para modificação de rotas dos pacotes. Esta tarefa pode ser automatizada e atualizada de forma dinâmica de acordo com a necessidade do serviço prestado e dos recursos computacionais disponíveis.

A solução de balanceamento de carga com o uso do OpenFlow funciona, e já foi testada e avaliada na literatura encontrada, porém, a metodologia proposta pelo *CoreLB* é mais abrangente e eficiente conforme apresentado nos resultados obtidos. O uso do protocolo de monitoramento SNMP auxilia o processo de análise de carga de cada *Nó Computacional*. O modelo do protocolo OpenFlow proporciona uma visão do processamento de fluxos, em que estatísticas são controladas e acessadas de forma centralizada. Com a determinação de qualidade obtida através das estatísticas providas por estes dois protocolos, os resultados avaliados provaram que a metodologia usada pelo *CoreLB* reduz a discrepância do uso de hardware de cada equipamento e aumenta o tempo de resposta de atendimento das requisições solicitadas pelos usuário. Os resultados obtidos mostraram-se mais eficientes e superiores há outras metodologias de balanceamento de carga.

Outro ponto importante de salientar é que a metodologia do *CoreLB* não está baseada no conhecimento da aplicação em uso pelo usuário e sim na situação do ambiente, o que possibilita usar a metodologia para diferentes aplicações e contextos pois a definição de balanceamento é independente dos serviços em uso. Na literatura estudada, percebe-se a maior eficiência do balanceamento quando se tem o conhecimento da aplicação em uso e sua operabilidade, no entanto, nem sempre este prévio conhecimento é possível e alguns ambientes há o uso de aplicações e serviços heterogêneos onde o balanceamento de carga não pode ser baseado apenas no funcionamento de um único serviço.

Com esta metodologia proposta para decisão de qualidade, mesmo nós computacionais com diferentes capacidades de hardware podem ter o balanceamento equilibrado de uma forma eficaz, pois, as requisições são balanceadas de forma proporcional com o uso dos recursos disponíveis em cada *Nó*. Outro fator positivo deste trabalho é que o balanceamento de carga é realizado na rede e este processo permite a adição, remoção e alteração de *Nós Computacionais* para atender um determinado serviço.

7.1 Contribuições Realizadas

Este trabalho não pretendeu apenas realizar contribuições, mas também comprovar a eficiência do balanceamento de carga sendo executado pela rede. Entretanto, entre as contribuições realizadas, é possível salientar:

- A possibilidade do uso do OpenFlow para ações de balanceamento de carga. Apesar de algumas possíveis limitações do modelo proposto por OpenFlow, como os limites para o número de fluxos possíveis de ser armazenado na memória do comutador (TCAM ou similar), a solução de balanceamento pelo comutador se mostra possível;
- Uma metodologia de avaliação de qualidade que determina a mudança de tráfego no comutador. O processo de qualidade avaliado pelo *CoreLB* é baseado no uso de recursos de hardware com um sistema de pontuação para nós que mais consomem recurso no período entre as últimas coletas de estatísticas;
- Uma técnica dinâmica para avaliação de qualidade dos nós computacionais baseada na discrepância de qualidade entre cada nó. O controlador que executa as coletas de dados estatísticos de consumo de hardware não é estático e seu acionamento acontece com maior frequência quando há nós computacionais consumindo mais recursos que outros. O intuito desta ação é trocar o destino do tráfego para nós com menos consumo e reduzir o uso de hardware de nós sobrecarregados.

7.2 Trabalhos futuros

Mesmo com resultados satisfatórios apresentados no Capítulo 6, existem fatores que necessitam de um estudo mais aprofundado da metodologia apresentada. Apesar dos avanços obtidos neste trabalho, ainda há itens necessários de serem avaliados para comprovar a eficácia da metodologia de balanceamento. É possível salientar:

- Avaliação da metodologia em ambientes reais e não apenas virtualizados. Todos os testes foram realizados usando redes virtuais e também sistemas virtuais. Apesar deste cenário representar exatamente o ambiente que muitas empresas utilizam, é importante avaliar a metodologia usando comutadores e servidores reais;
- Análise do desempenho usando nós computacionais com diferentes capacidades de hardware. Nas avaliações realizadas, todos os nós computacionais estavam usando a mesma capacidade de hardware. Para uma melhor avaliação é interessante usar equipamentos com diferentes configurações;

REFERÊNCIAS

- BELYAEV, M.; GAIVORONSKI, S. Towards load balancing in SDN-networks during DDoS-attacks. In: SCIENCE AND TECHNOLOGY CONFERENCE (MODERN NETWORKING TECHNOLOGIES) (MONETEC), 2014 INTERNATIONAL, 2014. **Anais...** IEEE, 2014. p. 1–6.
- BIANCO, A.; BIRKE, R.; GIRAUDO, L.; PALACIN, M. OpenFlow Switching: data plane performance. In: COMMUNICATIONS (ICC), 2010 IEEE INTERNATIONAL CONFERENCE ON, 2010. **Anais...** IEEE, 2010. p. 1–5.
- C.C. OKEZIE OKAFOR K.C, U. C. Open Flow Virtualization: a declarative infrastructure optimization scheme for high performance computing. **Academic Research International**, Nigéria, NG, UEMOA, v. 4, n. 4, p. 232–244, Jul 2013.
- HEO, G.; KIM, E.; CHOI, J. An Extended SNMP-based Management of Digital Convergence Devices. In: COMPUTER AND INFORMATION TECHNOLOGY (CIT), 2010 IEEE 10TH INTERNATIONAL CONFERENCE ON, 2010, Bradford, UK. **Anais...** IEEE, 2010. p. 2540–2547.
- JARRAYA, Y.; MADI, T.; DEBBABI, M. A Survey and a Layered Taxonomy of Software-Defined Networking. **IEEE Communications Surveys Tutorials**, [S.l.], v. 16, n. 4, p. 1955–1980, Fourthquarter 2014.
- KAUR, S.; SINGH, J.; KUMAR, K.; GHUMMAN, N. Round-robin based load balancing in Software Defined Networking. In: COMPUTING FOR SUSTAINABLE GLOBAL DEVELOPMENT (INDIACOM), 2015 2ND INTERNATIONAL CONFERENCE ON, 2015. **Anais...** IEEE, 2015. p. 2136–2139.
- KREUTZ, D.; RAMOS, F. M. V.; VERÍSSIMO, P. E.; ROTHENBERG, C. E.; AZODOLMOLKY, S.; UHLIG, S. Software-Defined Networking: a comprehensive survey. **Proceedings of the IEEE**, [S.l.], v. 103, n. 1, p. 14–76, Jan 2015.
- MARCON D., B. L. Flow Based Load Balancing: optimizing web servers resource utilization. **Journal of Applied Computing Research**, São Leopoldo, RS, BR, v. 1, n. 2, p. 76–83, Dec 2011.
- MCKEOWN, N.; ANDERSON, T.; BALAKRISHNAN, H.; PARULKAR, G.; PETERSON, L.; REXFORD, J.; SHENKER, S.; TURNER, J. OpenFlow: enabling innovation in campus networks. **SIGCOMM Comput. Commun. Rev.**, New York, NY, USA, v. 38, n. 2, p. 69–74, Mar. 2008.
- NAKAI, A.; MADEIRA, E.; BUZATO, L. On the Use of Resource Reservation for Web Services Load Balancing. **Journal of Network and Systems Management**, Online, v. 23, n. 3, p. 502–538, 2015.
- ONF, O. N. F. **Software-Defined Networking (SDN) Definition**. Disponível em: <<https://www.opennetworking.org/sdn-resources/sdn-definition>>. Acesso em: 1 Setembro 2015.

PFAFF, B.; PETTIT, J.; KOPONEN, T.; JACKSON, E.; ZHOU, A.; RAJAHALME, J.; GROSS, J.; WANG, A.; STRINGER, J.; SHELAR, P.; AMIDON, K.; CASADO, M. The Design and Implementation of Open vSwitch. In: USENIX SYMPOSIUM ON NETWORKED SYSTEMS DESIGN AND IMPLEMENTATION (NSDI 15), 12., 2015, Oakland, CA. **Anais...** USENIX Association, 2015. p. 117–130.

RAGALATHA P., M. C. S. K. K. Design and Implementation of Dynamic load balancer on OpenFlow enabled SDNs. **IOSR Journal of Engineering (IOSRJEN)**, New York, NY, USA, v. 3, n. 8, p. 32–41, Aug 2013.

SEHERY, W.; CLANCY, T. Load balancing in data center networks with folded-Clos architectures. In: NETWORK SOFTWAREZATION (NETSOFT), 2015 1ST IEEE CONFERENCE ON, 2015. **Anais...** IEEE, 2015. p. 1–6.

SHANG, Z.; CHEN, W.; MA, Q.; WU, B. Design and implementation of server cluster dynamic load balancing based on OpenFlow. In: AWARENESS SCIENCE AND TECHNOLOGY AND UBI-MEDIA COMPUTING (ICAST-UMEDIA), 2013 INTERNATIONAL JOINT CONFERENCE ON, 2013. **Anais...** IEEE, 2013. p. 691–697.

TECHOPEDIA. **Core Network**. Disponível em: <<https://www.techopedia.com/definition/6641/core-network>>. Acesso em: 24 Novembro 2015.

THEINSTITUTE.IEEE.ORG. SOFTWARE-DEFINED NETWORKS. **The Institute**, Online, v. 38, n. 4, p. 6–7, Dec 2014.

VAUGHAN-NICHOLS, S. OpenFlow: the next generation of the network? **Computer**, IEEE Computer Society, v. 44, n. 8, p. 13–15, Aug 2011.

WANG, P.; LAN, J.; CHEN, S. OpenFlow based flow slice load balancing. **Communications, China**, China, v. 11, n. 12, p. 72–82, Dec 2014.

ZELTSERMAN, D. **A Practical Guide to SNMPv3 and Network Management**. Upper Saddle River, NJ, USA: Prentice Hall PTR, 1999.