Um Orquestrador Flexível de Recursos de Rede e Computação para o aprimoramento de Qualidade de Serviço (*QoS*) em Aplicações Multimídia Baseadas em Funções Virtualizadas de Rede (*NFV*)

Rodrigo Moreira

Universidade Federal de Uberlândia
Faculdade de Computação
Programa de Pós-Graduação em Ciência da Computação

Uberlândia
2017

Rodrigo Moreira

Um Orquestrador Flexível de Recursos de Rede e Computação para o aprimoramento de Qualidade de Serviço (*QoS*) em Aplicações Multimídia Baseadas em Funções Virtualizadas de Rede (*NFV*)

Dissertação de mestrado apresentada ao Programa de Pós-graduação da Faculdade de Computação da Universidade Federal de Uberlândia como parte dos requisitos para a obtenção do título de Mestre em Ciência da Computação.

Área de concentração: Ciência da Computação

Orientador: Flávio de Oliveira Silva, Ph. D

Uberlândia
2017

Dados Internacionais de Catalogação na Publicação (CIP)
Sistema de Bibliotecas da UFU, MG, Brasil.

M838o
2017

Moreira, Rodrigo, 1993-
Um orquestrador flexível de recursos de rede e computação para o aprimoramento de qualidade de serviço (QoS) em aplicações multimídia baseadas em funções virtualizadas de rede (NFV) / Rodrigo Moreira. - 2017.
115 f. : il.

Orientador: Flávio de Oliveira Silva.
Dissertação (mestrado) - Universidade Federal de Uberlândia, Programa de Pós-Graduação em Ciência da Computação.
Inclui bibliografia.

1. Computação - Teses. 2. Redes de computadores - Teses. 3. Computação em nuvem - Teses. 4. Sistemas multimídias - Teses. I. Silva, Flávio de Oliveira, 1970-. II. Universidade Federal de Uberlândia. Programa de Pós-Graduação em Ciência da Computação. III. Título.

À minha família.

Agradecimentos

Ao Autor da Vida, "em quem estão escondidos todos os tesouros da sabedoria e da ciência" Cl. 2:3, por ter me criado à Sua imagem e semelhança; guardadas as devidas proporções com capacidade de pensar, criar e avaliar.

Aos meus pais Donizete e Cláudia pelo amor e suporte incondicional que me faz forte na execução dos planos. Eu devo tudo a vocês!

Aos meus irmãos Thiago e Donizete Jr. pelo incentivo; com palavras e olhares consoladores tornou os momentos turbulentos passíveis de superação.

À Larissa pelo companheirismo, paciência e estímulo.

Ao Prof. Flávio pelas orientações, conversas e ensinamentos que transcendem o âmbito acadêmico. Muito obrigado!

Aos professores Pedro Frosi e Rui Aguiar pelas contribuições, conversas e questionamentos.

Aos colegas da UFU por viabilizarem um ambiente de muito aprendizado, desafios, parcerias e boas conversas.

Aos amigos de perto e de longe que sempre possibilitam enxergar com humor as facetas da vida.

Aos colegas da Algar Telecom pelo incentivo e convivência.

Ao servidor Erisvaldo pela presteza e agilidade.

Aos docentes do PPGCO pelos ensinamentos e desafios.

Aos membros da banca pela disponibilidade, via de regra trazem grandes contribuições; professores Rodrigo Miani e Diogo Gomes.

"Também sei, que nada acontece sem a Tua vontade.
Mas preciso aprender a confiar em Ti.
Mas preciso aprender a descansar em Ti."
(Todas as Coisas – Fernandinho)

Resumo

Pessoas e organizações ao redor do globo utilizam aplicações multimídia para interação e comunicação. Existe notável crescimento de tráfego característico dessas aplicações, uma vez que há convergência de voz e dados para um modelo único de transporte. As redes de computadores, por suportar tais aplicações cadenciam desafios e oportunidades de negócios que os operadores têm aproveitado. Por outro lado, os usuários experimentam serviços com qualidades cada vez mais refinados, mesmo assim, existem aspectos que devem ser tratados para elevar a qualidade de experiência percebida pelos usuários ao consumir serviços ofertados sobre a internet. Métricas de *QoS* usualmente são baseadas nos usuários ou na rede, por isso, conceitos como *NFV* e *SDN* permitem aprimorar e expandir a perspectiva de oferta de uma camada de abstração para desenvolvimento de soluções flexíveis, que visam lidar com aplicações multimídia ofertadas em nuvem. Não obstante, a construção dessas soluções deve ser calcada em aspectos intrínsecos ao conceito de Internet do Futuro que preconiza soluções virtualizadas, flexíveis, convergentes, escaláveis, orientadas a contexto, seguras e melhor gerenciadas.

Este trabalho propõe uma abordagem para mitigar a distância das aplicações com os recursos de rede e computação. Usualmente as aplicações desconhecem características de *hardware* sobre o qual elas operam; quanto a rede, mesmo com esforços em vários níveis da pilha de protocolos, desconhece especificidades das aplicações em execução. Há também dificuldade de gerenciamento global de recursos, integração de políticas e acordos de serviço entre operadores. O trabalho visa adicionar uma camada de gerenciamento para elevar os níveis de *QoS* para aplicações multimídia baseadas em *NFV*. Para realizar isso, é proposto uma entidade que atua no plano de dados e controle, capaz de orquestrar recursos de rede e de computação simultaneamente para aprimorar *QoS* em aplicações multimídia. A incorporação dessa entidade conquista melhoramento de *QoS* por meio da oferta de resiliência na comunicação fim-a-fim de entidades consumidoras dessas aplicações, provê um balanceamento de carga adequado a fim de não comprometer parâmetros de *QoS* como tempo de resposta, provê escalabilidade sob demanda, promove o conceito de inspeção de pacotes para segmentação de políticas de rede orientada à aplicação, final-

mente realiza a separação de plano de dados e controle. A construção e experimentação da solução permitiu diminuir tempos de resposta nas aplicações multimídia; a solução reagiu adequadamente frente a cargas de trabalho heterogêneas no sentido de prover confiabilidade para garantia de *QoS*; também, o conceito de inspeção de pacotes foi capaz de atuar para diminuir a distância que as aplicações possuem da rede e o *hardware* sobre o qual elas operam.

Palavras-chave: Computação em Nuvem. Qualidade de Serviço. Aplicações Multimídia. Redes Definidas por Software. Virtualização das Funções da Rede..

Abstract

People and organizations around the globe use multimedia applications for interaction and communication. There is notable traffic growth characteristic of these applications since there is a convergence of voice and data for a single transport model. Computer networks, because they support such applications, meet the challenges and business opportunities that operators have taken advantage of. On the other hand, users use services with increasingly refined qualities, nevertheless, there are aspects that must be addressed to raise the quality of experience for users when consuming services offered over the internet. QoS metrics are usually user-based or network-based, hence concepts such as NFV and SDN allow to improve and expand the perspective of offering an abstraction layer for the development of flexible solutions that deal with multimedia applications offered in the cloud. Nevertheless, the construction of these solutions must be based on intrinsic aspects to the Future Internet concept that advocates virtualized solutions, flexible, convergent, scalable, context-aware, secure and better managed.

This dissertation proposes an approach to mitigate the distance of the applications with the resources of compute and network. Usually, the applications are unaware of the hardware characteristics on which they operate, as for the network, even with several efforts at various levels of the protocol stack, the specificities of running applications are unknown. There is also challenges of managing global resources, integrating policies and service agreements between operators. This work aims to add a management layer to raise the levels of QoS for multimedia applications based on NFV. To accomplish this, an entity that acts on the data and control plane is proposed, capable of orchestrating network and computing resources simultaneously to enhance QoS in multimedia applications. The incorporation of this entity achieves QoS improvement through resiliency in the end-to-end communication of the consumer entity of these applications, provides adequate load balancing in order to avoid compromising QoS parameters such as response time, provides on-demand scalability, improves the concept of package inspection for the segmentation of application-oriented network policies, and finally performs the separation of data and control plane. The construction and experimentation of the solution allowed

to reduce response times in the multimedia applications. The solution reacted adequately to heterogeneous workloads in order to provide reliability for QoS guarantee. Furthermore, the concept of packet inspection was able to act to decrease the distance that the applications have from the network and the hardware on which they operate.

Keywords: Cloud Computing. Quality of Service. Multimedia Applications. Software-defined Networking. Network function virtualization.

Lista de ilustrações

Lista de tabelas

Lista de siglas

API Application Programming Interface

BRAS Broadband Remote Access Server

CapEx Capital Expenditures

CLI Command Line Interface

DPI Deep Packet Inspection

DHCP Dynamic Host Configuration Protocol

EDCA Enhanced Distributed Channel Access

FIBRE Future Internet Brazilian Environment for Experimentation

HSS Home Subscriber Server

IP Internet Protocol

IMS IP Multimedia Subsystem

IaaS Infrastructure as a Service

IPTV Internet Protocol Television

ISP Internet Service Provider

LAN Local Area Network

LTE Long-Term Evolution

MTTF Mean Time to Failure

MAC Media Access Control

MOS Mean Opinion Score

NFV Network Function Virtualization

NAT Network Address Translation

NFVI Network Funcion Virtualization Infraestructure

NGN Next-generation Network

NFVO NFV Orchestrator

OpEx Operating Expenses

PoC Proof of Concept

PaaS Platform as a Service

PSTN Public Switched Telephone Network

QoE Quality of Experience

QoS Quality of Service

RAM Random-access Memory

RAID Redundant Array of Independent Disks

RTP Real-time Transport Protocol

SDN Software Defined Networking

SaaS Software as a Service

SIP Session Initiation Protocol

SNMP Simple Network Management Protocol

TCAM Ternary Content-addressable Memory

UE User Equipment

WAN Wide Area Network

VoIP Voice over Internet Protocol

VNF Virtual Network Function

VLAN Virtual Local Area Network

VM Virtual Machine

VPN Virtual Private Network

Sumário

Capítulo 1

Introdução

As redes de computadores, desde sua concepção aos dias atuais, com a disseminação de computação e nuvem e novos paradigmas de controle de rede, trazem consigo capacidades de comunicação sem precedentes, através de uma estrutura em conectividade de escala mundial (SIMSEK et al., 2016). A disseminação da Internet no âmbito comercial, no Brasil há cerca de duas décadas (PROCHNIK; UNE, 2003), foi catalizador de necessidades de novos serviços e oportunidade de negócios que utilizam a rede como baluarte (TARUTE; GATAUTIS, 2014). Também, há evidente evolução e padronização de *hardware* que viabiliza altas taxas de transferência, em consequência a comunicação alcança boas velocidades (NAKAMURA, 2011; BONETTO et al., 2009).

A evolução do conceito de Internet trouxe consigo meios para as pessoas se comunicarem por meio de uma diversidade de aplicações multimídia no mercado. Muitas companhias utilizam *Voice over Internet Protocol (VoIP)* como mecanismo padrão de comunicação dado o custo benefício que o conceito oferece (THORPE et al., 2016). No mesmo sentido a Internet permitiu novas formas das pessoas lidarem com questões rotineiras. Hoje, por exemplo, é possível comprar e vender pela Internet, pois existem comércios de variadas formas e públicos.

O paradigma da computação em nuvem (*cloud computing*) provê entrega e suporte de serviços computacionais sobre a internet com a premissa de consumo sob demanda. Ofertar recursos computacionais de processamento, memória e armazenamento como serviço exime os clientes de custos de aquisição de infraestrutura e subutilização dos recursos instalados. No mesmo sentido possibilita redução de custos de operação (*Operating Expenses (OpEx)*), redução de custos de mercado, provê facilidade de acesso e manutenção (ZHANG; CHENG; BOUTABA, 2010). O modelo de negócio de computação em nuvem converge para serviços de infraestrutura ou serviço. Esse nicho mercadológico é amplamente explorado por fornecedores como *Microsoft* (MICROSOFT, 2017), *Amazon* (AMAZON, 2017a) e *Google* (GOOGLE, 2017).

Acrescida essas capacidades, o conceito emergente de dissociação do plano de dados e controle possibilita redesenhar inúmeros aspectos das redes e a comunicação suportada por

ela. O conceito de Redes Definidas por *Software* (ou *Software Defined Networking (SDN)*) abriu as portas para a evolução no âmbito das redes pelo fato de tornar o plano de dados e controle disjuntos. É possível alcançar a ortogonalidade desses planos por meio do protocolo *OpenFlow* (MCKEOWN et al., 2008a), que efetivamente popularizou o conceito. Hoje é viável inúmeras pesquisas com internet, redes rurais, redes de *data center*, redes móveis e novas arquiteturas de rede (HU; HAO; BAO, 2014).

O modelo de computação em nuvem possibilitou a conceituação e materialização do paradigma de Virtualização de funções de rede (ou *Network Function Virtualization (NFV)* (ETSI, 2012a)). Com *NFV* é possível repensar o modelo de negócio dos grandes fornecedores de equipamentos de rede, com alternativa de virtualizar tais funções comercializada em equipamentos específicos sobre *hardware* comum e com isso lograr de benefícios como redução de *Capital Expenditures (CapEx)* (ETSI, 2012a; BOURAS; KOLLIA; PAPAZOIS, 2017). Conceitos de *SDN* e *NFV* são independentes e complementares, construir soluções à luz desses conceitos traz consigo inúmeras vantagens que vão desde negócios a operação (HERNANDEZ-VALENCIA; IZZO; POLONSKY, 2015; COSTA-REQUENA et al., 2015).

Todo esse elenco de tecnologias convergentes tem possibilitado a construção e oferta de inúmeras aplicações multimídia demandada pelos usuários sob uma nova perspectiva de qualidade. Ocorre que essas aplicações são demasiado sensíveis a fatores computacionais e de rede, e a não garantia de alguns parâmetros compromete a qualidade. Existem inúmeros esforços que objetivam aprimorar a qualidade de serviço (ou *Quality of Service (QoS)* de aplicações multimídia (SHANNON et al., 2016; HASAN; AL-RIZZO; AL-TURJMAN, 2017; PANDEY; KUMAR; KUMAR, 2016). Porém, é importante destacar que aplicações multimídia exigem uma camada de abstração sobre conceito de *QoS*, pois embora parâmetros de rede possam ser garantidos, é possível que o usuário não tenha a qualidade de experiência (ou *Quality of Experience (QoE)*) adequada. Por isso, é necessário soluções que pautem sua operação nesses desafios que a estrutura de rede atual impõe.

Nesse sentido, o presente trabalho contribui com o estado da arte ao conceituar e implementar uma solução compatível com *NFV* sob prova de conceito (ou *Proof of Concept (PoC)*) nas redes multimídias. Haja vista que é previsto tendência de crescimento em tráfego multimídia no mundo (CISCO, 2016a) até 2020. A solução é capaz de orquestrar recursos computacionais e de rede com objetivo de diminuir a distância entre as aplicações e os recursos físicos, e adiciona uma camada ao conceito de *QoS* para aplicações multimídia. Registros na literatura sugerem esforços nesse sentido (DUDOUET; EDMONDS; ERNE, 2015; KIM; PARK; SONG, 2015), porém são insuficientes em flexibilidade de lidar com diversas tecnologias e contexto da aplicação. Não lidam com aspectos de recursos computacionais e de rede simultaneamente, possuem operação condicionada ao domínio de rede. Possui balanceamento de carga dependente de protocolo de transporte e não oferecem manutenção da conectividade das entidades consumidoras de aplicações mul-

timídias, para obter uma conexão resiliente mesmo em cenários de falhas em elementos intermediários.

Efetivamente, a solução proposta utilizou o método de acomodar sobre questões pouco cobertas pela literatura no sentido de oferecer uma solução pertinente, que ofereça um nível de abstração ao conceito de *QoS* para aplicações multimídia. São explorados cenários similares aos ambientes reais, e isso eleva o nível de debate e relevância que a proposta versa. Sobretudo por considerar métricas válidas para os cenários propostos que garantem a conveniência da proposta. Assim, foi possível consolidar e implementar uma solução compatível com *NFV* que tem capacidades de influência nos domínios de recursos de rede e computação de forma simultânea sobretudo orientado ao contexto da aplicação e com *QoS* garantido nos dois domínios para elevar o nível de *QoE* nas sessões multimídia.

1.1 Motivação

Métricas tradicionais de *QoS* lidam com parâmetros que focam no usuário ou em parâmetros de rede (FIEDLER; HOSSFELD; TRAN-GIA, 2010). Com o advento do conceito das tecnologias de *NFV* e *SDN*, surge um novo conjunto de desafios e oportunidades (WANG; LI; XIA, 2015), uma vez que outras dimensões de gerenciamento de recursos devem ser consideradas para finalmente melhorar a experiência do usuário nas aplicações multimídia. No universo de virtualização, é necessário considerar os recursos de computação que hospedam os serviços multimídia.

Em um típico cenário de rede, a distância entre as aplicações e a rede é evidente (COSTA, 2013; SHARKH et al., 2013; LIU et al., 2015; DERAKHSHAN et al., 2013); aplicações de tempo real como *VoIP* utilizam a premissa de melhor esforço para transporte. O mesmo conceito é paralelo às aplicações que desconhecem limitações de *hardware* sobre o qual operam, o aumento excessivo de carga de trabalho compromete a performance das aplicações multimídia.

Em (MUELLER; MAGEDANZ, 2012) é proposto um mecanismo capaz de definir parâmetros em uma rede *wireless* baseado no congestionamento de *canais*, priorização de pacotes (possível por meio do protocolo *Enhanced Distributed Channel Access (EDCA)*), percepção do usuário (fornecida por meio de protocolos específicos) e políticas previamente definidas. A solução considera ganhos em aprimoramento de *QoS* ao definir tais parâmetros nos recursos de rede, no entanto negligencia o aspecto computacional que suporta a oferta de tais aplicações multimídia. Para o cenário, elevada carga de trabalho para consumo de *streaming* tornaria a oferta do serviço inconstante, e a qualidade de experiência é degradada.

Por outro lado, Dudouet, Edmonds e Erne (2015) contrasta o conceito de soluções orientado a rede com o discurso que *QoS* e *QoE* são garantidos quando há confiabilidade na performance das aplicações. Isso é evidenciado quando especifica-se o conceito

de confiabilidade com disponibilidade; uma aplicação pode responder a uma determinada solicitação (disponibilidade), mas é necessário que ela responda adequadamente (confiabilidade) sobretudo com tempo hábil para não comprometer parâmetros definidores de qualidade. O trabalho oferece uma aplicação que orquestra recursos computacionais para garantir confiabilidade em computação em nuvem. A solução é medida pelo conceito de tempo médio de falha (ou *Mean Time to Failure (MTTF)*). A solução negligencia a necessidade de gerenciamento e orquestração de recursos de rede, por isso é incompleta no seu domínio de influência.

A singularidade das soluções encontradas no estado da arte motiva uma abordagem capaz de orquestrar recursos computacionais e de rede simultaneamente, em cenários de aplicações multimídia ofertadas em ambientes virtualizados. Isso garante aprimoramento de *QoS* aos usuários, eleva o nível de resiliência das sessões multimídia através do conceito de garantia de conectividade, balanceamento de carga eficiente sobre a função de rede virtualizada, promove a técnica de inspeção de pacotes quanto a assertividade em descrever contextos de aplicações multimídia e finalmente a solução garante elasticidade para cenários heterogêneos de carga de trabalho.

1.2 Objetivos e Desafios da Pesquisa

O objetivo geral do trabalho é prover uma solução *NFV compliant* para gerenciamento simultâneo de recursos de computação e de rede para aprimoramento de *QoS* em ampliações multimídia sobre nuvem. Para pleitear êxito na proposta foi previsto a conclusão de micro objetivos conforme descrito:

- Projetar o *design* conceitual da solução à luz do conceito *NFV*;
- Consolidar um algoritimo de caráter não intrusivo para coletar métricas da função de rede virtualizada. Esse objetivo garante a flexibilidade da solução na interoperabilidade com sistemas *cloud computing* diversos;
- Configurar e Implementar módulos e regras de balanceamento carga na solução. A conclusão desse objetivo garante integridade e balanceamento adequado de carga de trabalho;
- Consolidar uma função de rede virtualizada modelo para viabilizar a política de elasticidade. A conclusão desse objetivo garante uma função de rede virtualizada matriz para ser utilizada pela solução no âmbito da elasticidade;
- Implementar módulos assíncronos para gerenciamento de recursos. A conclusão desse objetivo caracteriza a solução como orientada a *compute* e rede;

- ❑ Implementar uma camada de segurança para interface da solução com módulo de balanceamento de carga. A conclusão do objetivo agrega um nível de segurança à proposta ao garantir que terceiros não modifique destinos elegíveis para balanceamento de carga;

- ❑ Implementar módulo de inspeção de pacotes. A conclusão desse objetivo torna a solução orientada à aplicação;

- ❑ Configurar e Realizar experimentos com a solução em operação em ambientes similares a realidade;

- ❑ Analisar os resultados obtidos sob a ótica das métricas de qualidade definidas.

Sinalização de aplicações multimídia exigem configuração correta de inúmeras entidades para que elas comuniquem entre si, sobretudo de forma correta. Por questões de padronização, as funcionalidades são distribuídas de forma granular para as entidades, por exemplo na arquitetura para entrega de conectividade de dados nas redes móveis *IP Multimedia Subsystem (IMS)*. A interação entre tais entidades exige entendimento profundo do comportamento e padrão das interfaces. Nesse sentido o comportamento inadequado de uma entidade compromete a funcionalidade da aplicação de forma geral, por isso é considerado um desafio, sobretudo na curva de aprendizado. Soluções *open source* disponíveis na comunidade entregam soluções *hands-on* na modalidade *all-in-one* como em (OPENIMS, 2016; OPENSTACK, 2017a; HAPROXY, 2017; ONOS, 2017), mas esse modelo é pouco produtivo para cenários que exigem elasticidade e customização. Para lidar com isso, é necessário um *deploy* e configurações fim-a-fim das entidades.

Construir solução fora do domínio de acesso ou do núcleo da aplicação exige desdobramentos em termos de garantia de conectividade entre os dois mundos. No que tange a pesquisa, a integração do ambiente de acesso (recursos de rede) e o núcleo da aplicação (recursos de computação) concentrados na solução exigiu inúmeros esforços em configurações de endereçamento lógico. Outro desafio é a padronização da função de rede virtualizada para compor o catálogo, isso exige certificar de forma minuciosa funcionamento adequado da funcionalidade proposta em termos de configuração. É necessário garantir conectividade da função de rede virtualizada com a rede interna para possivelmente acoplar funções em série, e com a rede externa, para receber balanceamento de carga de trabalho.

1.3 Hipótese

No que tange a pesquisa, sustenta-se a hipótese que é possível construir uma solução compatível com *NFV* que provê esforços de *QoS* baseado no contexto da aplicação no ambiente de acesso e nos recursos computacionais onde operam aplicações multimídia si-

multaneamente. Dado as inúmeras vantagens de computação em nuvem e a materialização de programabilidade de comportamento da rede.

1.4 Contribuições

As contribuições gerais do trabalho são: (1) um mecanismo capaz de prover conectividade com resiliência entre entidade consumidoras de aplicações multimídia; (2) uma solução automática de elasticidade para aplicações multimídia para lidar com cenários diversos de carga de trabalho; (3) encorajar a utilização do conceito de técnicas de *Deep Packet Inspection (DPI)* para analisar contexto de aplicação para prover políticas assertivas no aprimoramento de *QoS*; (4) uma abordagem não invasiva de gerenciamento de recursos e telemetria em nuvens diversas; (5) solução capaz de lidar com redes multidomínios;

1.5 Método

A Ciência da Computação é definida por Bell e Newell (1971) em tradução literal como "o estudo dos fenômenos relacionados aos computadores". Como ciência, é necessário uma abordagem lógica para construir e validar argumentos que sustentam uma hipótese. Os passos lógicos para construção do trabalho é característico do método dedutivo hipotético (DODIG-CRNKOVIC, 2002), onde são consideradas referências específicas e relevantes na literatura, especifica-se um problema, propõe uma solução e a experimenta por meio de um protótipo de implementação. Na Figura 1 é ilustrado a estrutura dos métodos utilizados para construção do trabalho.

Figura 1 – Estrutura de Métodos.

Na primeira etapa foi previsto a composição de massa teórica com propósito de fundamentar a viabilidade da proposta. Nessa etapa foi realizado um estudo profundo do estado da arte, protocolos de sinalização multimídia, arquiteturas de redes multimídia, protocolos de transporte de dados multimídia, algorítimos de codificação e decodificação de media, arquiteturas de computação em nuvem e seus modelos de serviço, tecnologias de gerenciamento de recursos em nuvem, estudo de *frameworks* e ambientes reais de experimentos. O fim dessa etapa culminou inúmeros artefatos como experimentos funcionais de

arquiteturas multimídia, tutoriais de instalação, esboço de algorítimos e testes funcionais com sistemas gerenciadores de nuvem.

A etapa dois contempla atividades de refinamento da proposta e contextualização da contribuição. Nessa etapa foi possível elucidar as capacidades e limitações dos esforços presentes na literatura. Através de um estudo profundo do estado da arte a etapa permitiu contrastar inúmeros trabalhos da área com o proposto, em consequência a alocação exata da contribuição do trabalho. A etapa gerou artefatos como resumos conceituais, tabelas comparativas, *surveys*, especificação de critérios conceituais para fins comparativos.

Na etapa três foi possível construir o esboço conceitual da solução onde foi elucidado as capacidades da solução em preencher lacunas identificadas no estado da arte. A etapa permitiu definir interfaces, escopo inicial, modelo de serviço na nuvem e tecnologias. Inerente à etapa, houve necessidade de construir ilustrações estruturais e diagramas para compreensão dos fluxos e das relações entre entidades.

A etapa quatro diz respeito ao processo de codificação e configuração dos módulos. A implementação é condicionada ao modelo especificado na etapa anterior.

Na etapa cinco foi previsto a validação e testes da solução proposta. A validação consiste na percepção comportamental da execução da solução e se as saídas estão em consonância com o previsto em etapas anteriores. A atividade de testes permite validar a execução do código e tratar exceções a desvios incomuns que ocorrem em tempo de execução. As atividades dessa etapa permitiu construir avaliação experimental para discussão em *workshops* ou *symposiums*.

1.6 Organização da Dissertação

Essa dissertação é organizada em cinco capítulos. O Capítulo 2 detalha e discute aspectos teóricos que posicionam o trabalho proposto com o estado da arte. São detalhados tecnologias de sistemas de computação, Redes Definidas por *Software*, Virtualização de Funções de Rede, conceitos e tecnologias redes multimídia, telefonia sobre o protocolo *IP* e trabalhos relacionados com o proposto.

O Capítulo 3 descreve a proposta da dissertação, bem como detalhes da implementação. Neste capítulo é definido as características e a arquitetura da solução proposta.

O Capítulo 4 versa sobre a avaliação experimental da solução, é discutido as características do cenário de experimentos, métricas de avaliação; o capítulo discute e analisa os resultados alcançados.

O Capítulo 5 discute a conclusão obtida na construção e experimentação da solução proposta. No capítulo, é destacado as contribuições e aponta alguns aspectos que germinam trabalhos futuros.

Também há um apêndice que contém codificações complementares à solução.

Capítulo 2

Fundamentação Teórica

Nesse capítulo são apresentados aspectos teóricos essenciais para compreender e contextualizar a contribuição do trabalho. Serão discutidas tecnologias consolidadas e emergentes no âmbito das redes de computadores. O capítulo versa sobre conceitos de redes definidas por software, virtualização de funções de rede e computação em nuvem e o universo das aplicações multimídia. A apresentação desses assuntos pretende destacar meios tecnológicos que possibilitam que "Alice" e "Bob" se comuniquem como exemplificado em Schulzrinne e Wedlund (2000), e os esforços para que tal comunicação se caracterize como de qualidade.

Nos dias atuais se materializa a convergência tecnológica que Bernal (2007), Hallingby et al. (2016), Chung (2014) descreveram; onde texto, imagem, voz e vídeo se convergiriam para um modelo único de transporte, o *Internet Protocol (IP)*. A convergência do modelo de telefonia foi possível após a migração do conceito de comutação de circuitos para comutação de pacotes segundo Camarillo e Garcia-Martin (2007), e também pelas vantagens que sistemas em nuvem traz em termos de custo e qualidade para hospedar serviços de forma mais aprimorada (JAIN; PAUL, 2013). Também há tendência de migrar de serviços de rede ofertado por equipamentos legados comercializados por fornecedores históricos para a serviços totalmente baseados em *software*, principalmente serviços de comunicação de voz para lograr de soluções mais baratas e de qualidade (MIJUMBI et al., 2016; GLITHO, 2014; CARELLA et al., 2014).

2.1 Computação em Nuvem

Computação em nuvem (ou *cloud computing*) é um paradigma de computação bem definido em termos conceituais; termo proposto inicialmente nos anos de 1960 por Jonh McCarthy (FOSTER et al., 2008). Fundamentalmente, *cloud computing* traz consigo a possibilidade de utilizar de infraestrutura de computação em serviços sob demanda. O sistema de computação deverá ser capaz de responder com sensibilidade a determinadas cargas; tais cargas podem ser atemporais e com características bem definidas. Divide-se

a arquitetura de uma *cloud* em *back end* e *front end*. A sessão do cliente, onde acontecem as requisições de serviços são tratadas pelo *front end* e a administração da *cloud*, bem como as regras de negócio são tratadas na seção *back end* (JADEJA; MODI, 2012).

Define-se *cloud computing* como as aplicações e *hardware* entregues como serviços por meio de internet. Esses serviços são denominados como *Software as a Service (SaaS)*, *Infrastructure as a Service (IaaS)* e *Platform as a Service (PaaS)*. Uma nuvem será pública se ela compartilha esses recursos de *hardware* e *software* entregues como serviços com outros usuários, sobretudo ao utilizar o modelo de serviço pague pelo que usar. Por outro lado, nuvens dentro de organizações específicas, que atendem a demandas comuns de uma organização é denominada nuvem privada (ARMBRUST et al., 2010).

Segundo Hashem et al. (2015) *PaaS* são recursos diferentes rodando em nuvem com objetivo de oferecer plataforma de computação para os usuários finais, por exemplo *Google App Engine*[1]. Outro modelo de serviço *IaaS* refere-se aos recursos de *hardware* que são entregues a usuários finais sob demanda, como o *Amazon EC2* [2]. Por fim, *SaaS* são aplicações que estão em execução em *data centers* e são ofertadas pela internet, por exemplo *Google Docs* [3].

Uma *cloud* possui algumas características básicas como *on demand self-service*, inerente a capacidade de adquirir mais recursos de armazenamento, processamento, memória, máquinas virtuais; há vasta possibilidade de acesso para dispositivos heterogêneos; *pool* de recursos para serem consumidos sob demanda; elasticidade rápida, diz respeito ao tempo que é gasto para servir um requisitante de recursos; monitoramento do serviço, a nuvem deve ter interface para ser gerenciável (PUTHAL et al., 2015). O modelo de computação em nuvem consegue lidar com questões de ociosidade de recurso, principalmente pela característica escalável desse tipo de solução. Para diminuir *CapEx* e *OpEx*, é imprescindível que questões de eficiência energética sejam estruturadas. No trabalho de Mastelic et al. (2014) é discutido algumas estratégias que pretendem reduzir esses custos e as dificuldades de se obter consumo eficiente nesse tipo de contexto.

Utilizar alocação de infraestrutura em nuvem traz grandes vantagens, como as que seguem: (1) não exige investimento inicial, não é necessário que um prestador de serviço adquira toda infraestrutura que planeja utilizar, antes pode-se adotar a estratégia de alocar recursos dinamicamente e pagar pela quantidade que usar; (2) redução de custo operacional, existe flexibilização para alocar recursos, oferece economia pois recursos são desapropriados quando não utilizados; (3) altamente escalável, de acordo com a demanda pode-se alocar infraestrutura. Por exemplo, escalabilidade no que diz respeito a alocação de recursos para suprir a demanda de um servidor que em épocas específicas, lida com grande quantidade de requisições de serviços; (4) acesso fácil, por se tratar de um serviço disponível na rede (ou Internet) existe a característica de facilidade de acesso. Todavia,

[1] https://cloud.google.com/
[2] https://aws.amazon.com/pt/ec2/
[3] https://www.google.com/docs/about/

o termo facilidade se aplica também a *smartphones*, *tablets* dentre outros equipamentos móveis; (5) redução de risco e despesas: falhas no *hardware* da infraestrutura eventualmente ocorrem, assim é transferido para o provedor da infraestrutura, que deverá lidar com desafios de disponibilidade. Também, no que diz respeito a despesas, a contratadora de infraestrutura exime-se de oferecer treinamentos para lidar com a infraestrutura (ZHANG; CHENG; BOUTABA, 2010).

Coutinho et al. (2015) aborda aspectos inerentes a elasticidade e as métricas comuns no estado da arte para medir qualidade de serviços hospedados em *cloud*. As métricas são classificadas em (1) alocação, relaciona-se com a capacidade da nuvem provisionar elasticidade; (2) capacidade, métricas para avaliar por exemplo a capacidade máxima, capacidade de expansão, capacidade de cargas e outras; (3) custo, há métricas para avaliar o custo dado uma performance, custos de migração, custos de *bandwidth* e outros; (4) qualidade de serviço, há métricas que avaliam percentual de violação, custos de reconfiguração da infraestrutura, ganho de performance; (5) utilização de recursos, percentual de utilização, demanda, número de máquinas virtuais e baixa utilização; (6) escalabilidade, há métricas como sistema de escalabilidade efetiva que medem a performance do sistema de expansão e retração de recursos; (7) tempo, inicialização, exclusão, média de tempo para expandir a capacidade do serviço e outras.

Elasticidade é um conceito de destaque em *cloud computing*, que difere de forma clara de escalabilidade. Elasticidade é a capacidade do sistema de *cloud computing* prover e gerenciar recursos para lidar com a demanda corrente possivelmente de forma automática. Por outro lado, escalabilidade é a capacidade de adicionar recursos computacionais à infraestrutura para alcançar uma certa escala para lidar com determinada carga de trabalho (AGRAWAL et al., 2011).

Características de elasticidade podem ser descritas de forma sistemática. Política, diz respeito ao comportamento do redimensionamento da escala, se ocorre de forma manual ou automática. Escopo, que diz respeito ao modelo de serviço ofertado; plataforma, serviço ou infraestrutura. Propósito, relacionado ao objetivo da elasticidade; redução de energia ou aprimoramento de performance. Método, é relacionado ao mecanismo de elasticidade; se ocorre pelo conceito de réplicas, redimensionamento ou migração. Essas características viabilizam a migração de inúmeros serviços, plataformas e funções para *cloud computing* (GALANTE; BONA, 2012).

A computação em nuvem na modalidade *IaaS* suporta hoje iniciativas abstratas para virtualização de recursos de rede. Hoje o operador de rede tem dificuldades em acomodar uma grande variedade de equipamentos de rede para suportar a operação. Há considerável custo com energia, custos com operação e treinamento de profissional para operar tais equipamentos; esses equipamentos alcançam fim da vida e tem dificuldades de receberem atualizações inerentes a serviços. No entanto, há abordagens conceituais para migrar equipamentos, sobretudo suas funcionalidades para serem executadas e entregues como

serviços em *cloud computing* (ETSI, 2012a).

2.1.1 Gerenciadores de Nuvem

Para gerir recursos de infraestrutura é necessário ferramentas capazes de fazer o provisionamento e operação de forma eficiente dos recursos disponíveis. Nesta seção são apresentados alguns gerenciadores e suas características.

2.1.1.1 *Eucalyptus*

A ferramenta se apresenta como um *framework* simples e modular para uso computacional e de armazenamento. Os usuários podem iniciar instâncias, controlá-las e encerrá-las, por meio de uma interface uniforme similar a utilizada por outros gerenciadores de nuvem (NURMI et al., 2009). Os módulos que compõe o ecossistema são descritos conforme abaixo:

- *Node Controller (NC)*: é a primeira interface entre a instância e os recursos de infraestrutura. Responsável pela descoberta de estado, memória, *CPU*, espaço de disco e números de *core*. Por meio deste controlador é possível iniciar ou terminar instâncias, pois ele trabalha diretamente com o *hypervisor*;

- *Cluster Controller (CC)*: é responsável pelo escalonamento de instâncias por meio de interação direta com o *NC*, controla a camada de rede de cada instância e traz informações sobre todos os *NC*. O módulo *CC* possui um conjunto de *NC* para que na medida que houver solicitação de alocação de recursos ele seja capaz de consultar o módulo *NC* e repassar requisições de criação de instância ou destruí-la;

- *Storage Controller (SC)*: é responsável pelo serviço de armazenamento de dados das instâncias e também para o caso de *streaming* de dados entre instâncias. Com esse serviço de armazenamento não há inconsistência de dados em situações de escrita e leitura de um determinado objeto, por segurança é utilizada a soma de verificação *MD5*;

- *Cloud Controller (CLC)*: é uma coleção de *web services* para gerenciamento dos recursos que compreende alocação, propriedades das instâncias e da rede; serviços de dados para persistência dos dados, configuração do ambiente; e a interface para oferta dos serviços para que os usuários possam interagir com os recursos por meio de interfaces definidas e autenticação;

A parte de destaque do *Eucalyptus* é a de rede, pois ele provê uma camada de rede sobreposta. Com isso obtém-se segurança, pois as máquinas são conectadas diretamente ao *software Ethernet bridge*, é possível que o administrador defina o *Media Access Control*

(MAC) das instâncias e também é possível isolamento de instâncias por meio de *Virtual Local Area Network (VLAN).*

2.1.1.2 *OpenNebula*

É uma alternativa *open source* para gerenciamento de nuvens privadas, públicas ou hibridas, se propõe a oferecer gerenciamento de serviços de virtualização, rede e armazenamento. A solução sustenta teses de flexibilidade, interoperabilidade, estabilidade, escalabilidade, controle, simplicidade e velocidade. A arquitetura do *OpenNebula* define: (1) *front-end* responsável pela execução da oferta dos serviços de *cloud*; (2) *Datastores* é o serviço de armazenamento de imagens das *Virtual Machines (VMs)*; (3) e o serviço de rede que provê conectividade por meio de *VLAN* (MILOJIčIć; LLORENTE; MONTERO, 2011).

2.1.1.3 *Xen Cloud Platform*

Plataforma *open source* para gerenciamento de nuvens públicas, privadas ou híbridas que utiliza *Xen Hypervisor* para oferta de serviço de virtualização. Especificamente na modalidade *Iaas* (BIST; WARIYA; AGARWAL, 2013). A solução é altamente escalável, pois seu *hypervisor* consegue escalar 4095 *hosts* com 16Tb de memória *Random-access Memory (RAM).* A solução diferencia-se dos seus pares, pois suporta migração de máquinas virtuais para outras soluções de nuvem como o *OpenStack* (XEN, 2016).

2.1.1.4 *Microsoft Windows Azure Platform*

Solução comercial para gerenciamento de nuvem pública e oferta de *PaaS.* A arquitetura da solução é composta pelos componentes: *compute*, que provê uma interface para programabilidade, instanciação de máquinas virtuais; o serviço *Fabric* que trabalha como o controlador das funções da nuvem. Com ele é possível provisionar, armazenar, monitorar as máquinas virtuais e os servidores físicos da *cloud* (ROLOFF et al., 2012). A solução oferta serviços de armazenamento especificamente plataformas para *big data* e serviços de *backup.* Também há oferta de serviços de rede nas modalidades de redes virtuais, balanceador de carga, *gateway* de *Virtual Private Network (VPN)*, gerenciador de tráfego e serviço de distribuição de conteúdo (MICROSOFT, 2017).

2.1.2 *Amazon EC2*

Plataforma comercial para entrega de computação como serviço, destaca-se por ser a maior em termos de mercado. Em 2006, introduziu a modalidade *IaaS* e possui amplo portfólio de serviços de computação, como *S3* que oferece serviço de armazenamento; *EC2* para entrega de servidores virtuais sob a ótica de elasticidade e balanceamento de carga; e *Cloudfront* para serviços de distribuição de conteúdo de vídeo (VOORSLUYS;

BROBERG; BUYYA, 2011). Características marcantes dos serviços computacionais oferecidos pela *Amazon* é o controle completo e integrado das instâncias pela interface *web*, segurança para funcionalidades de redes e recursos de computação (AMAZON, 2017b).

2.1.2.1 *OpenStack*

Plataforma *open source* escrita em *Python* para gerenciamento de infraestrutura de nuvem inicialmente proposta pela *NASA*. Possui característica de ser escalável, pois consegue instanciar cerca de 60 milhões de máquinas virtuais e armazenar bilhões de objetos. Também é compatível com virtualizadores *ESX*, *KVM*, *XEN* e outros (SEFRAOUI; AISSAOUI; ELEULDJ, 2012). A solução consiste em serviços de *computing*, rede e armazenamento. Os módulos principais são: *Swift*, para armazenamento de objetos; *Keystone*, serviço de identidade; *Nova*, serviço de *compute* especificamente para gerenciamento de virtualização; *Neutron*, serviço de rede, onde é possível customizar redes e serviços como *Dynamic Host Configuration Protocol (DHCP)*; *Cinder* é o serviço de gerenciamento do espaço de disco alocado para determinada instância; *Glance*, serviço de imagens dos sistemas operacionais disponíveis para as instâncias, também é possível criar e armazenar *snapshots* de instâncias para serem utilizados posteriormente como cópia exata de uma futura instância (OPENSTACK, 2017c).

O *OpenStack* se destaca dos seus pares no critério compatibilidade com virtualizadores, performance no provisionamento, modelo de rede (WEN et al., 2012; BIST; WARIYA; AGARWAL, 2013; VORAS et al., 2011; PARADOWSKI; LIU; YUAN, 2014; LASZEWSKI et al., 2012). No que diz respeito ao modelo de rede há esforços que objetivam integrar o modelo convencional de rede de infraestruturas *cloud* por meio da incorporação do conceito de redes definidas por *software* (JAIN; PAUL, 2013; SHARKH et al., 2013; BAKSHI, 2013).

2.2 Redes Definidas por Software

As redes de computadores, desde sua disseminação no âmbito comercial, operam sistematicamente com protocolos bem definidos, com evoluções discretas e certificados por entidades reguladoras. No arcabouço de telecomunicações, existem muitos equipamentos – responsáveis por tarefas bem definidas na rede de diversos fornecedores, como: *Cisco*, *Hewlett-Packard (HP)*, NEC, *Juniper*, *Ericsson* e etc., cada qual, fornece seu produto com sistema operacional especializado no *hardware* hospedeiro. Desta forma, a gerência e manutenção da rede se torna onerosa, quando considerado redes maiores, pois é exigido pessoal conhecedor da solução do fornecedor. Por outro lado, o operador estabelece um laço de dependência com o fornecedor, principalmente em termos de suporte, oferta de novas soluções. Em última instância, percebe-se uma barreira aos operadores para inserção de novos serviços e no padrão operacional da rede.

SDN é um paradigma para arquitetura de redes que propõe desacoplar o plano de dados do plano de controle (FOUNDATION, 2012). Plano de dados é o encaminhamento dos pacotes, plano de controle é a inteligência da rede, no que diz respeito a forma com que o pacote é encaminhado, sujeito a interesses definidos. Nesse sentido, novos serviços podem ser ofertados, e existe redução expressiva de custo de *OpEx* dos operadores (HERNANDEZ-VALENCIA; IZZO; POLONSKY, 2015), pois a possibilidade de uma visão geral da rede, habilita esquivar se de enormes quantidades de linhas de comando para configuração dos *switches*. É possível automatizar a escalabilidade, no que diz respeito a integração novos dispositivos.

Em tempo real é possível habilitar serviços específicos, como balanceamento de carga, *firewalls*, inspetores de pacotes, gerenciamento de banda, qualidade de serviço (*QoS*), gerenciamento de uso de energia e etc., o que torna a rede flexível. No trabalho de (FARHADY; LEE; NAKAO, 2015) é descrito inúmeras aplicações categorizadas nas camadas conceituais de *SDN*. A característica de controle centralizado provida por *SDN* traz consigo evolução positiva em atividades de gerenciamento, provisionamento, otimização e configuração dos recursos da rede. Tais atividades podem ser feitas de forma dinâmica sobretudo logicamente centralizadas em um único equipamento, de forma transparente a rede se sensibiliza sobre nova forma que deve operar.

A Figura 2 ilustra a arquitetura do paradigma *SDN*. Existem três camadas bem definidas, aplicação, controle e infraestrutura. Os serviços de redes são mapeados para o controlador através de *Application Programming Interface (API)* que os controladores definem, serviços podem ser *firewall* ou controle de acesso por exemplo. A camada *Control Layer*. Essa camada contempla a inteligência da rede, onde também se mantém uma visão global da rede, essa camada opera os serviços de rede. A *Infrastructure Layer* contempla os dispositivos da rede, que operam segundo regras e comportamentos definidos na camada superior.

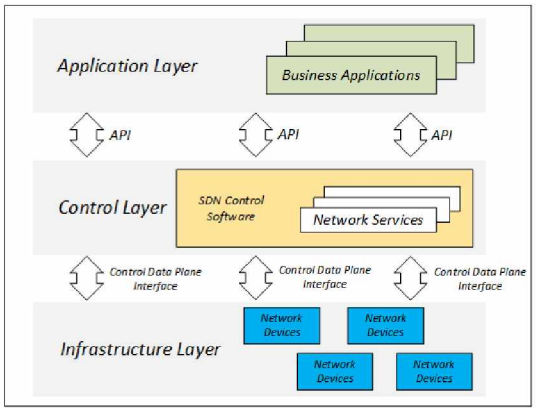

Figura 2 – Arquitetura *SDN*.

Fonte: (FOUNDATION, 2012).

A comunicação entre *Control Layer* e *Infrastrucure Layer* se dá por interfaces de controle do plano de dados, conceitualmente denominada *Southbound API*, por exemplo o protocolo *OpenFlow* e *Forces* (DORIA et al., 2010). De forma análoga tem-se *Northbound API*, que é a classe de *Application Programming Interfaces (APIs)* que possibilitam a comunicação entre *Application Layer* e *Control Layer*, por exemplo *REST API* que formaliza um padrão de interface usada por aplicações para expressar requisitos da rede para o plano de controle.

Há questões de segurança inerentes a todo modelo conceitual de *SDN*, principalmente nas interações dessas camadas. No trabalho de Hu et al. (2015) é sugerido alguns princípios a serem considerados na construção de projeto baseado em *SDN*. Segurança no nível arquitetural, serviços de checagem de pacotes, garantia da integridade da comunicação de mensagens de regras os fluxos. Em adição, questões básicas de segurança como autenticação, autorização, confidencialidade e integridade.

Para suportar customização por meio de *software* é necessário a interação das camadas do modelo *SDN*. Ao considerar uma abordagem *button-up* tem se a interação por meio de protocolos da classe *southbound* como por exemplo *OpenFlow*.

2.2.1 O Protocolo *OpenFlow*

A dificuldade de experimentar novos serviços de rede culminou na comunidade científica a necessidade de um mecanismo capaz de separar tráfego de produção do tráfego de testes. Também o fato dos equipamentos serem fechados tanto no âmbito de *hardware* quanto *software* inviabiliza a execução de testes. Ou seja, os fornecedores entregam um produto funcional, totalmente modelado para atividade fim – por exemplo encaminhamento de pacotes (*switch*) incapaz de receber influência operacional externa por alguma interface.

Ao introduzir o paradigma *SDN*, o gerenciamento dos elementos de rede *L2* pode ser concebido de duas formas. Tradicionalmente, tem-se *Command Line Interface (CLI)*, uma interface de configuração e gerenciamento desses elementos. Por meio de *CLI* não se aplica separação do plano de dados e controle, pois as regras de comportamento são inseridas diretamente no circuito interno do elemento. Entende-se como *CLI* o modelo tradicional de configuração de elementos, onde são inseridos comandos nas entidades *L2* por sessões *telnet* ou *SSH*, o circuito interno do elemento compreende esses comandos e os assume como regras de operação. Por outro, com o paradigma de redes *SDN* materializa-se uma interface padrão que provê inteligência no controle, configuração e gerenciamento da rede, por meio do protocolo *OpenFlow*. Esse protocolo é a interface entre o plano de controle e dados.

Elementos das redes *SDN* (embora fisicamente compatíveis) não são gerenciados e configurados pelo modelo tradicional (*CLI*), antes, as regras são centralizadas em uma entidade central, o controlador *SDN*. Essa entidade opera segundo requisitos de alto nível,

comumente expressadas em linguagem de programação. Quando o controlador recebe esses requisitos, ele os converte em mensagens *OpenFlow* e as descreve nos *switches*, que por sua vez assume como regra de comportamento. *OpenFlow* é a oferta de um protocolo aberto que difundiu o conceito *SDN*, presente na região *southbound* do modelo conceitual do conceito de *SDN* capaz de definir comportamentos específicos, baseado em necessidades ou regras de negócio nos equipamentos como *switches* e roteadores (MCKEOWN et al., 2008a).

O conceito de *switch OpenFlow* inspira-se em *Ternary Content-addressable Memory (TCAM)*, que é memória física dos equipamentos legados que contém informações por exemplo da interface que deve encaminhar o pacote, dado seu endereço de *MAC* destino. O protocolo *OpenFlow* se aloja na similaridade de comportamento que os equipamentos dos fornecedores devem adotar, ou seja, todos devem armazenar em memória física informações para a encaminhamento do pacote. Nesse sentido, a necessidade do *OpenFlow* vem de encontro a capacidade de editar as regras escritas nas *TCAMs*, e estabelecer em tempo de execução, isto é, sem parar a rede, o novo destino que o pacote deve se sujeitar (MCKEOWN et al., 2008a). Quando o *switch* recebe um pacote para encaminhamento, se constatar que a regra não está definida na sua tabela interna, será feito imediatamente uma consulta no controlador *SDN*. Para o protocolo *OpenFlow*, esse cenário inspira o evento *Packet-In* (DIXIT et al., 2013), após, o destino para encaminhamento será concedido conforme as regras previamente estabelecidas.

A Figura 3 ilustra a proposta de controlar o *switch* segundo regras implementadas externamente. Na Figura tem-se o controlador – agente externo, responsável por comunicar através de um canal seguro com o *software* do equipamento. A comunicação com controlador se dá pelo protocolo *OpenFlow*, e o conteúdo da comunicação são regras determinadas em *software*, na qual o equipamento deve cumprir. A camada de *software* do equipamento trabalha diretamente nas políticas da tabela de fluxo do *hardware*. Os clientes possuem a interface de comunicação com o *switch*, que por sua vez atende pela a camada de *hardware* do equipamento (MCKEOWN et al., 2008a).

A expectativa é que a disseminação do protocolo *OpenFlow* permita experimentar os conceitos *SDN* em redes de variadas características, sobretudo em larga escala. Hoje existem diversas plataformas que operam o protocolo *OpenFlow*. Essas por sua vez fazem interface entre os requisitos das aplicações e a execução em nível operacional desses requisitos.

2.2.2 Controladores

Controlador é a inteligência da rede, precisamente uma entidade de *hardware* que é consultada quando acontece eventos por exemplo *Packet-In*, possui uma visão global da rede, pois todos os *switches* comunicam com o controlador através de uma interface segura para operarem corretamente o fluxo de encaminhamento pacotes (MCKEOWN et

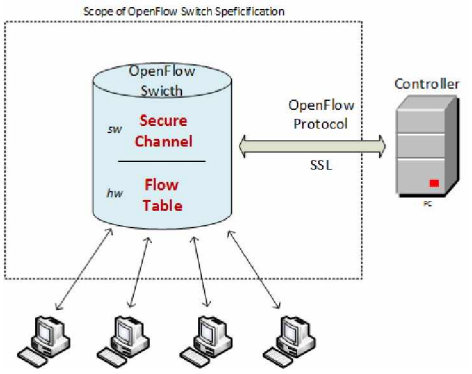

Figura 3 – Idealização de um *switch OpenFlow*.

Fonte: (MCKEOWN et al., 2008b).

al., 2008b). O projeto e design do controlador é pautado por algumas metas, como: liberar o desenvolvedor da aplicação de conhecer detalhes intrínsecos do *hardware* proprietário. Liberdade do operador de rede ao lidar com interfaces e protocolos proprietários. Dissociação da evolução dos componentes de *hardware* e *software*, isto é, cada um poderá seguir a linha do tempo de mercado de forma independente (MCKEOWN et al., 2008b).

Há esforços para que o controlador tenha comportamentos similares a um sistema operacional – principalmente no que diz respeito a gerenciamento de recursos entre as entidades. Também há grande quantidade de controladores cada qual projetado a luz de critérios específicos. Encontra-se na literatura especificações de controladores como: *POX*, *ONOS* (ONOS, 2015), *Ryu* (RYU, 2015), *FloodLight* (FLOODLIGHT, 2015), *OpenDayLight* (OPENDAYLIGHT, 2015), *Trema* (TREMA, 2015), *Beacon* (BEACON, 2015) e outros.

Mais acima, no modelo conceitual *SDN* é encontrado a interface aplicações-controlador presente na classe *northbound*. Essas interfaces provêm a comunicação do motor de controle com as aplicações. Comumente, as aplicações publicam e consomem funcionalidades operacionais da rede por meio de serviços *REST* (RICHARDSON; RUBY, 2008). Os Controladores acima mencionados ofertam serviços por meio dessa *API*, salvo o *POX*.

É possível juntar dois paradigmas emergentes e alcançar maiores benefícios em termos de comportamento nas redes de computadores. Como sugere (ETSI, 2012a) a possibilidade de virtualizar funções de rede em *data centers* em conjunto com a possibilidade de programar funcionalidade viabiliza a construção de serviços altamente especializados.

2.3 Network Function Virtualization (NFV)

Por meio do conceito de virtualização é possível pensar em formas alternativas de executar funções de rede. Tais funcionalidades são comumente disponíveis por meio de equipamentos específicos de fornecedores consagrados do mercado. Segundo Jain e Paul

(2013) virtualização tem se tornado comum nos dias de hoje e é possível virtualizar coisas rotineiras; como por exemplo: compras, comunicação, educação e etc. Por isso, a proposta de virtualizar recai sobre algumas características positivas e isso possibilita transpor essas capacidades também para o universo de redes de computadores.

Características destacadas por Jain e Paul (2013) são: (1) compartilhamento, que é a capacidade de dividir um recurso demasiado grande entre vários usuários; (2) isolação, diz respeito à segurança, onde cada instância virtualizada deve ter isolação assegurada (*slice*); (3) agregação, possibilita somar recursos computacionais pequenos, por exemplo, armazenamento; agregação de discos com preços populares possibilita consolidar um sistema *Redundant Array of Independent Disks (RAID)*; (4) dinamismo, recursos podem ser realocados quando não estão em pleno uso, por outro lado, novas instâncias virtualizadas podem ser criadas afim de suprir eventual demanda; (5) gerenciamento fácil, pelo fato de ser baseado em *software* é mais fácil de gerenciar, devido a interface uniforme.

Virtualização de funções de rede, *NFV* propõe mudar a forma como os operadores de rede constroem e operam suas arquiteturas de rede, devido a tecnologia de virtualização que operam no *hardware* hospedeiro. Agrupar equipamentos de rede de vários tipos e funções em *data centers*, onde cada um desses equipamentos podem ser virtualizados em *hardware* comum é uma solução que traz inúmeros benefícios. Cada instância ou função de rede conceitualmente oferece uma funcionalidade de rede, que pode inclusive operar em série (ETSI, 2012a).

A virtualização apresenta-se como uma solução para diversos cenários e demandas, por exemplo, escalabilidade de redes móveis (BASTA et al., 2014); com virtualização pretende-se abordar os desafios das redes atuais de qualidade de serviço experiência, também inúmeros serviços. Martins et al. (2014) propôs um ambiente para virtualização de entidades baseadas em *software*, como prova de conceito foi implementado funcionalidades como *firewall*, *load balancer*, *Broadband Remote Access Server (BRAS)*, monitor de fluxo, sistema de detecção de intrusão e *Network Address Translation (NAT)*.

Segundo (ETSI, 2012a) podem se destacar algumas benefícios das técnicas de virtualização de equipamentos de rede: (1) redução de energia; (2) redução de custos; (3) possibilidade dos operadores de rede seguirem o ritmo de mercado; (4) disponibilidade de equipamento, vários equipamentos sob uma única plataforma; (5) oferta de serviços segmentados; (6) encoraja a distribuição de equipamentos virtualizados na comunidade aberta.

As redes convencionais são implementadas com a ligação em série de equipamentos que possuem funções bem definidas. A proposta de *NFV* traz consigo diferenças para o design atual como sugere (ETSIGS, 2013): (1) desacoplamento, os elementos de rede deixam de ser compostos pelo conjunto de *hardware* e *software*, por fim possibilita a evolução de cada separadamente; (2) flexibilidade, é alcançada na implantação de funções de rede pois o *hardware* e *software* executam funções diferentes em cada momento; (3) operação

dinâmica, o fato de haver instâncias dos equipamentos alcança-se maior flexibilidade para adaptar o desempenho da função de rede virtualizada, por exemplo de acordo com o tráfego atual adota-se determinada estratégia.

A Figura 4 ilustra o *framework NFV* que é composto por entidades que executam atividades de gerenciamento e orquestração. Nesse sentido, *Network Funcion Virtualization Infraestructure (NFVI)* controla os recursos físicos e suporta a execução das funções de rede virtualizadas; *Virtual Network Functions (VNFs)*, implementa uma função de rede que executa sobre *NFVI*; gerenciador *NFV* (ou *NFV MANO*) garante o gerenciamento das funções específicas de *hardware* e *software* e virtualização da infraestrutura (ETSIGS, 2013).

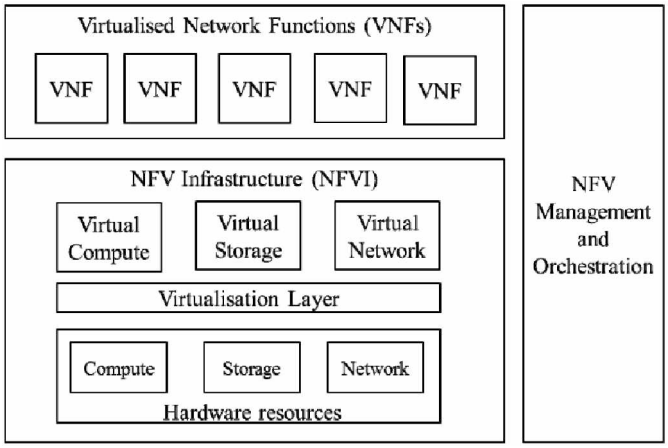

Figura 4 – Abstração do *framework NFV*.

Fonte: (ETSIGS, 2013).

NFV e *SDN* são propostas distintas, entretanto *NFV* contribui com *SDN* para alcançar benefícios, por exemplo, em uma proposta de rede escalável (ETSI, 2012a). Nesse sentido é possível destacar que *NFV* transfere funções de rede de equipamentos dedicados para equipamentos baseados em *software*, sobre *hardware* comum. Por outro lado, *SDN* possibilita um controle centralizado e torna a rede programável; em adição, desacopla os planos de dados e controle. Mesmo conceitualmente distintas, há benefícios no uso dessas tecnologias em conjunto. Por exemplo, *SDN* atende *NFV* com oferta de conectividade programável entre as *Virtual Network Functions (VNFs)*, essa conectividade programável é gerenciada por um orquestrador. Com mesmo objetivo é possível executar a funcionalidade de um controlador *SDN* e migrar entre nuvens mediante a necessidade da rede (HAWILO et al., 2014).

2.4 Telefonia *IP*

Nos dias atuais a comunicação é fundamental para as pessoas e organizações; há vários meios para se comunicar, por meio de texto, imagem, gestos, voz, toque e outros. No cenário globalizado, a comunicação por voz se tornou amplamente utilizada devido sua eficiência e praticidade. Desde quando o telefone foi proposto, no final do século XIX e até os dias atuais ocorreram expressivas evoluções tecnológicas no sentido de melhorar a qualidade e preço do serviço. O modelo de telefonia amplamente utilizado é *Public Switched Telephone Network (PSTN)*, que demanda elevado custo de *OpEx* por parte dos operadores (JOSEPH, 2010). A arquitetura da rede *PSTN* se baseia na alocação de recursos pelos elementos intermediários no sentido de prover um caminho com garantias quando uma chamada está em curso. Por isso, essa seção dedica-se a esclarecer detalhes da aplicação multimídia *VoIP*, dado sua popularização e custo benefício.

A convergência tecnológica de comunicação por voz através de redes legadas para redes *Next-generation Network (NGN)* sugere maior economia de recursos de *CapEx* e *OpEx* Joseph (2010) e consumo eficiente de energia (BOLLA et al., 2011). Existem algumas vantagens inerentes à utilização do transporte *IP* para amostras de áudio utilizadas no processo de comunicação. Economia para os usuários, as empresas podem construir seu próprio sistema de ramal utilizando sua *Local Area Network (LAN)* ou *Wide Area Network (WAN)*, dado que telefonia de longa distância a considerar internacionais são caras; eficiência na utilização de recursos; utilização por diversos dispositivos não necessariamente dispositivos fixos e também a possibilidade de integração entre si, ou seja, serviço de *VoIP* possui interface com a telefonia fixa legada. No mesmo sentido a adoção do transporte *IP* cadencia a utilização de diversas aplicações multimídia como: vídeo conferência, *live streaming*, *stored streaming*, *Internet Protocol Television (IPTV)*, *WebRTC*[4] e outras.

A utilização do transporte baseado em *IP* pelas aplicações multimídia recai sobre alguns desafios, pois aplicações multimídia são sensíveis a atraso e pouco tolerantes à perda (TANENBAUM, 2002). O modelo conceitual de rede é dividido em camadas que não receberam adaptações significativas em sua implementação e evolução para lidar com os requisitos das aplicações multimídia. De forma que a pilha de protocolos, de maneira geral, é agnóstica a aplicação, portanto, a premissa do serviço de melhor esforço não supre as demandas dessas aplicações adequadamente. Embora seja possível destacar alguns avanços que propõe lidar com o desafio de *QoS* para aplicações multimídia: alocação de banda definitiva, abordagem pouco perene pois estatisticamente não se utiliza o canal reservado todo o tempo; diferenciação de serviço *DiffServ* que basicamente categoriza os pacotes transportados pelos *Internet Service Providers (ISPs)* entre pacotes de primeira e segunda classe – conceito de prioridades; Carvalho e Mota (2013) destacam algumas

[4] Serviço de multimídia via *browser* padronizado pela *W3C* que possibilita interação em diversos aspectos entre os usuários.

classes de esforços encontrados na literatura: abordagens de mudança de *codec* em tempo de execução (ALSHAKHSI; HASBULLAH, 2011; MYAKOTNYKH; THOMPSON, 2009; TEBBANI; HADDADOU, 2008), reconfiguração da taxa de codificação (MKWAWA et al., 2010; ZHOU; SHE; CHEN, 2010; JAMMEH et al., 2012), reconfiguração da forma de empacotamento (WEI; WU; WU, 2009; YUHE; JIE, 2009), abordagens que lidam com algoritmo de controle de erro de encaminhamento (LI et al., 2010; LIZHONG et al., 2010; JUNG; IBANEZ, 2010).

2.4.1 Qualidade de Serviço em Telefonia *IP*

O serviço de melhor esforço por não suprir a demanda das aplicações multimídia é intrinsecamente ofensor de *QoS* em chamadas *VoIP*. Há dois extremos que são discutidos, de um lado sugere a necessidade de garantir banda pelo caminho em que os dados de uma chamada utilizam. Abordagem pouco usual no cenário da Internet, pois cada *ISP* em tese pode utilizar um critério próprio de roteamento entre seus dispositivos e finalmente não garantir banda. Outro caminho são as propostas de solução baseadas em aplicação ou em protocolos específicos, para todos os efeitos são abordagens liberais que não exigem modificações conceituais das redes de computadores.

Existem métricas que medem o *QoS* de uma chamada de voz, a *Mean Opinion Score (MOS)* proposta pelo (ITU-T, 2003b) que faz um ranking da qualidade da chamada onde 5 é excelente, 4 bom , 3 médio, 2 ruim e 1 péssimo. Tal técnica é subjetiva, pois ela é baseada na percepção dos usuários e assim cada pessoa pode perceber e avaliar de uma forma. Apesar de subjetiva, essa abordagem é amplamente utilizada para avaliação de *QoS* (STREIJL; WINKLER; HANDS, 2016). Outras abordagens não subjetivas são: *Perceptual Speech Quality Measure (PSQM)* (ITU-T, 1995), *Perceptual Evaluation of Speech Quality (PESQ)* Rix et al. (2001) e *E-MODEL* (BERGSTRA; MIDDELBURG, 2003). Todas métricas citadas acima visam avaliar a *QoS* que conforme Carvalho e Mota (2013) tem impacto direto com a qualidade de experiência *QoE* do usuário; essa métrica está ligada a satisfação do usuário, avaliação mediante percepção, ao passo que *QoS* é diretamente ligada a garantia em de recursos de tecnologia (KIM et al., 2008). Comparado com a rede *PSTN* a modalidade *VoIP* é mais flexível quanto a escolha do *codec* que faz a amostragem e compactação da voz em *datagrams*. Há na literatura diversos *codecs*. A Tabela 1 detalha alguns e suas características:

As colunas das tabela são detalhadas como segue:

- *Codec & Bit rate* diz respeito nome padronizado do *codec* e o número de *bits* por segundo que são gastos para a entrega da voz na chamada;
- *Codec Sample Interval* é o intervalo de amostra que o *codec* trabalha.Por exemplo, o *codec* G.729 opera na construção de amostras a cada 10ms, e utiliza 10 *bytes* por amostra. Portanto o *bit rate* equivale ao tamanho da amostra *80 bits (10 bytes)*,

Tabela 1 – Especificações de *codecs* de Voz.

Codec Informations			
Codec & Bit Rate (Kbps)	*Codec Sample Interval (ms)*	*MOS*	*Bandwidth Ethernet (Kbps)*
G.711 (64 Kbps)	10 ms	4.1	87.2 Kbps
G.729 (8 Kbps)	10 ms	3.92	31.2 Kbps
G.723.1 (6.3 Kbps)	30 ms	3.9	21.9 Kbps
G.723.1 (5.3 Kbps)	30 ms	3.8	20.8 Kbps
G.726 (32 Kbps)	5 ms	3.85	55.2 Kbps
G.726 (24 Kbps)	5 ms		47.2 Kbps
G.728 (16 Kbps)	5 ms	3.61	31.5 Kbps
G722_64k (64 Kbps)	10 ms	4.13	87.2 Kbps

Fonte: (CISCO, 2016b).

dividido tempo que o codec gasta construir uma amostra. Para o G.729 tem-se: $codec\ bit\ rate = \frac{80}{10} = 8kbps$;

- *MOS* é a métrica subjetiva utilizada para medição da qualidade de voz. Para esse tipo de métrica os usuários avaliam a qualidade da voz mediante uma escala;
- *Bandwidth Ethernet* é o custo final que o transporte de dados de voz sobre a rede *IP* causa no enlace. A considerar o *over head* de identificação do pacote, como cabeçalhos e campos de verificação de erros.

2.4.1.1 Atraso

Atraso é o tempo que o pacote contendo a amostra de voz gasta para chegar até seu destino. Os atrasos podem ser de propagação, compreensão, serialização e chaveamento, descompressão, congestionamento, armazenamento e *jitter*. Boas taxas de empacotamento tem influência significativa no atraso fim a fim, para tanto, há uma variedade de *codecs* conforme mencionados que trata questões de eficiência de empacotamento à sua maneira. *Jitter* é a variação da latência, são atrasos variáveis na rede que ocorre devido ao tamanho e enfileiramento imprevisíveis; no contexto *VoIP* é predominante importante para qualidade da chamada (BERNAL, 2007). A (ITU-T, 2003a) classifica níveis de atraso como: (1) atraso fim a fim: máximo 0 a 50 ms geralmente em contextos de chamadas de ótima qualidade; (2) 50 a 150 ms: boa qualidade; (3) 150 a 400 ms impactos de qualidade; (4) 400 ms não aceitável.

2.4.1.2 Variação do atraso (*Jitter*)

O atrasos que ocorrem em aplicações *VoIP* não são constantes, pois em diversos pontos e elementos da rede podem ocorrer atrasos. Para lidar com eventual atraso é proposto um *buffer* para armazenamento dos pacotes a medida que chegam, para posteriormente serem

ordenados conforme o protocolo *Real-time Transport Protocol (RTP)*[5]. Quanto maior for a variação do atraso (ou *jitter*) maior é o tempo gasto para *buffering*, assim se esse tempo for muito grande não será possível entregar os pacotes em ordem e a mensagem trocada será incompreensível (BERNAL, 2007; TANENBAUM, 2002).

2.4.1.3 Perda de Pacotes

É o percentual (%) de perda de dados em uma transmissão, obtido por número de pacotes perdidos pelo número de pacotes esperados pelo destino. Por motivos físicos ou lógicos os pacotes de um fluxo *RTP* não alcançam o destino, ou chegam demasiado tarde, a chamada prossegue portanto com blocos de voz com falhas. Conforme Bernal (2007), existem técnicas de inserção de silêncio nos blocos de pacotes perdidos, suspensão de silencio que significa quando não estiver falando nada, o protocolo envia pacotes de silêncio ao invés de fazer amostras de ruídos do ambiente – essa técnica reduz o consumo de banda em 40%; mas para os interlocutores há um desconforto auditivo no funcionamento do algoritmo, entre o reconhecimento da fala e momentos de silencio. Em termos percentuais, é esperado que em uma sessão *VoIP* ocorra perdas máxima de 5% para garantia de qualidade superiores a satisfatória.

2.5 Redes Multimídia

Com a oferta de novos serviços de telecomunicações torna-se evidente as possibilidades de comunicação sobre diversos meios e plataformas, no entanto usuários utilizam a plataforma que melhor se encaixa a sua expectativa. O modelo de comunicação passou por mudanças importantes, parte-se de um contexto baseado em comutação de circuito – existente desde o surgimento dos telefones – para a comutação baseada em pacotes. A rede móvel $2G$ e suas precedentes utilizam o modelo de comutação baseado em circuito (BOGINENI et al., 2009), tal modelo de comunicação é capaz de oferecer voz e vídeo e mensagens instantâneas. Porém, o modelo de comutação baseado em pacotes é mais eficiente (HOLMA et al., 2006), e é fortemente marcado pela característica de flexibilidade, portanto, foi considerado na padronização da rede $3G$ (DAHLMAN et al., 2010).

A comutação baseada em pacotes oferece aos usuários finais endereços *IP* para conectividade com a Internet, assim há possibilidade de navegação pela internet, leitura de *e-mails*, transporte de voz e vídeo, dentre outras possibilidade que a rede *IP* oferece. De forma nativa, a rede $3G$ trabalha com comutação de pacotes, ao passo que a rede $2G$ necessita de um modem – que faz interface da rede baseada em circuito com a rede *IP*. Na maioria das vezes tais dispositivos são os equipamentos dos usuários finais que usualmente fazem o trabalho de conversão digital-analógico.

[5] Protocolo para entrega de áudio e vídeo sobre *IP* (SCHULZRINNE et al., 2003).

IMS é uma *framework* de arquitetura acima da camada de transporte do modelo *TCP*, necessário para que se tenha garantia de qualidade em nível minimamente superior ao melhor esforço nos serviços multimídia que são transportados sobre *IP*. Por isso, *IMS* tem sido amplamente trabalhado para suportar demandas emergentes de redes da nova geração, redes $3G$ e predecessoras são baseadas em necessidades humanas de conectividades, no entanto $5G$ estão sendo desenhadas sob ótica de *cloud computing* para oferecerem de 10 a 100 vezes mais dispositivos conectados e consumindo dados, inclusive conectividades no contexto *Machine to machine (M2M)* (ABU-LEBDEH et al., 2016).

As redes atuais, sobretudo $3G$ e $4G$, possuem o modelo comutação baseada em pacotes, por isso não garantem plenamente *QoS*, carga e integração com novos serviços. A entrega de pacotes com a premissa de melhor esforço prejudica a qualidade de serviço, por exemplo em uma sessão de vídeo conferência – pode haver percepção de atrasos, distorções e interrupções. Existem dificuldades de garantir o controle de fluxo e carga em um domínio de comutação baseado em pacotes, mesmo com transporte sobre *TCP*. Os operadores de rede possuem uma visão mais quantitativa do que qualitativa dos dados. Ou seja, os usuários são cobrados pela quantidade de *bytes* transferidos, e não exatamente pela característica dos dados – se são dados de vídeo conferência, *voice mail*, *streaming* por exemplo (CAMARILLO; GARCIA-MARTIN, 2007).

Com *IMS* objetiva-se integrar por meio da abstração de controle novos serviços multimídia. Também a possibilidade dos operadores basearem suas políticas de qualidade para gerenciar melhor os serviços. É possível por exemplo executar a migração de um cliente para um perfil de tarifação menos oneroso – no caso da utilização de mensagens instantâneas; quando considerado a baixa demanda por *bitrate*. Embora a rede de comutação baseada em pacotes ofereça numerosas possibilidades, algumas questões como as mencionadas acima carecem de atenção. Para tanto, a arquitetura *IMS* provê o tratamento de tais pontos em aberto. Existem numerosos problemas no contexto de *IMS* que são desafiadores conforme apontado por (ABU-LEBDEH et al., 2016), como o fato das entidades desse modelo arquitetural oferecem seus serviços no modelo *steteful*, isso significa que um registro de usuário na arquitetura é amarado a uma entidade específica, assim não é possível migra-lo em tempo de chamada, por exemplo. Ou também propor maior granularidade nas funções das entidades *IMS* com objetivo de facilitar escalabilidade. Mesmo assim, *IMS* torna-se um a interface para introdução de novos serviços específicos de multimídia sobretudo com *QoS* garantido (ou superior a tentativa de melhor esforço) (CAMARILLO; GARCIA-MARTIN, 2007).

A Figura 5 ilustra a arquitetura *IMS* padronizada pelo *3GPP* [6]. Na Figura tem-se o *Home Subscriber Server (HSS)* , responsável por armazenar dados dos os usuários, como: informação de localização, informações de segurança – para autenticação, informações de

[6] Entidade desenvolvedora de padrões, fornece aos seus membros ambiente estável para produzir relatórios, e especificações de novas tecnologias(3GPP, 1999).

perfil e etc. Em uma rede *IMS* pode haver mais de um elemento *HSS*, quando é o caso de espelhamento.

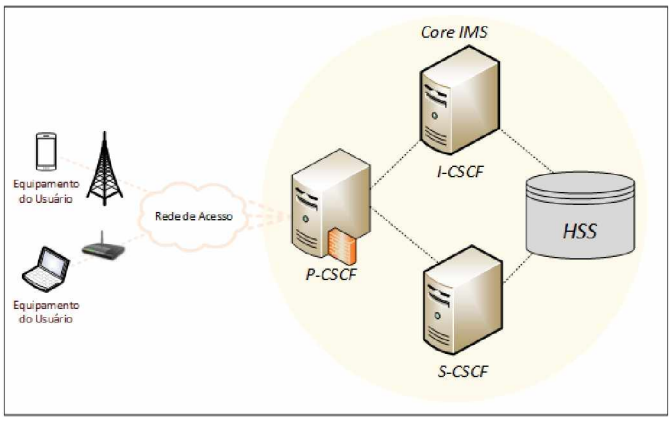

Figura 5 – Arquitetura do *3GPP IMS*.

Outro elemento de uma rede *IMS* é o *Call Session Control Function (CSCF)*, responsável pelo processo de controle e sinalização na rede. Divide-se em três categorias, cada qual para um fim específico: *P-CSCF*, *I-CSCF* e *S-CSCF*. A entidade *P-CSCF* é responsável por registrar o equipamento de borda na rede *IMS*. Provê um serviço de interface da borda como o *proxy* da rede *IMS*. O processo de registro, leva em conta políticas de segurança. No processo inicial de autenticação é estabelecido um número de *IPsec* para o terminal requisitante da autenticação. Após autenticado, o *P-CSCF* propaga a identidade íntegra do novo elemento na rede. Nesse módulo também é realizado as atividades de comprimir e descomprimir mensagens – a utilizar critérios de qualidade de serviço – pois cada canal de comunicação meio possui restrições de *bitrate*, por exemplo. Conceitualmente o módulo *Policy Decision Function (PDF)* é construído para trabalhar em cooperação com *P-CSCF*, gerencia especificamente *QoS*, por exemplo em atividade de autorização dos recursos do plano de dados.

I-CSCF provê um serviço de ponto de conexão para redes externas. Também é a entidade responsável pelo controle e direcionamento do fluxo das mensagens internas, também da manutenção do domínio *Domain Name System (DNS)*. Interage com a entidade *HSS* e é capaz de criptografar mensagens trocadas no domínio, determina um *S-CSCF* para tratamento das mensagens de sinalização que são recebidas. Finalmente, a entidade *S-CSCF* localizado no *core* da rede *IMS* trabalha no plano de sinalização. Mantém o registro do equipamento do usuário – neste caso *IP*, e no *core* da rede, um endereço único para controle de sinalização das mensagens *Session Initiation Protocol (SIP)*[7]. Também interage com outros elementos da rede como *HSS* para carregar o perfil do usuário, e faz roteamento das mensagens para outros equipamentos, o que depende do modelo de serviço ofertado. Em uma rede *IMS* pode haver execução de outros protocolos bem como

[7] Protocolo para sinalização e controle de sessões multimídia (ROSENBERG et al., 2002)

formatos de mensagens. Os protocolos disponíveis são destacados na *Release 5* do *3GPP* (3GPP, 1999).

Existem diversas arquiteturas na comunidade que oferecem serviços no modelo *IMS*. Cada uma atendendo a requisitos específicos de comportamento, possuem implementações, abstrações e possibilidades de customização distintas.

2.5.1 Arquitetura *Clearwater*

Abordagens mais recentes de arquiteturas de redes multimídia são propostas com intuito de resolver problemas de escalabilidade, convergência fixo-móvel e contribui no custo benefício, ou seja, aumento de receita pautada com otimização de recursos (KOUKAL; BESTAK, 2006). Os operadores deixam de utilizar a rede legada – baseada em comutação de circuitos, e oferecem soluções sobre IP. *Clearwater* (CLEARWATER, 2015) é uma arquitetura de rede multimídia que segue princípios da padronização *IMS*, entretanto diferencia-se das demais pelo fato de ser orientada a nuvem – ambientes *cloud*. *Clearwater* possui todas entidades tradicionais do *IMS – P-CSCF, I-CSCF e S-CSCF*, também oferece controle de chamada baseado no protocolo de sinalização *SIP*.

A Figura 6 ilustra a arquitetura do *Clearwater*. A entidade *Bono (Edge Proxy)*, é a interface inicial de comunicação dos *User Equipment (UE)* responsável por direcionar investidas de autenticação para os elementos responsáveis no *core* da rede. *Bono* é altamente escalável, em uma rede virtualizada pode admitir vários. Quando a comunicação de um *UE* falha, outra entidade similar associa-se de forma transparente para a oferta da conectividade, as conexões trocam mensagens baseadas em *SIP-UDP* ou *SIP-TCP*. *Sprout (SIP Router)* executa as atividades de roteamento e autenticação, realiza as funções previstas para o *S-CSCF*. Mensagens *SIP* em seções multimídia são processadas por essa entidade, que possui interface direta com *HSS*, por exemplo para requisitar informações de perfil de um determinado usuário. *Sprout* é capaz de manter entidades redundantes para viabilizar o processo de balanceamento e escalabilidade.

Homestead (HSS Mirror) é o banco de dados *Cassandra*[8], que recebe solicitações de recuperação de credenciais de usuários e perfil dos usuários. Essa entidade possui uma interface *web* para provisionamento, e internamente trata requisições de dados através de interface *Cx* (interface padrão de comunicação prevista no *3GPP*). A entidade *Ralf (RF CTF)* realiza serviço de *billing* (ou faturamento), recebe parâmetros das entidades *P-CSCF*, *I-CSCF* e *S-CSCF* para executar a tarifação. A entidade *Ralf* consegue fazer isso após consultar informações disponíveis no *HSS*, também as especificidades recebidas pelas entidades *CSCF* e finalmente executa a lógica de tarifação através da funcionalidade *Charging Data Function (CDF)*. A entidade *Homer (XDMS), XML Document Management Server (XDMS)* é um servidor de armazenamento de dados em formato *XML* capaz de armazenar dados dos serviços e configurações de cada usuário do sistema. *Elis* é

[8] Sistema de Banco de Dados distribuído altamente escalável(CASSANDRA, 2015).

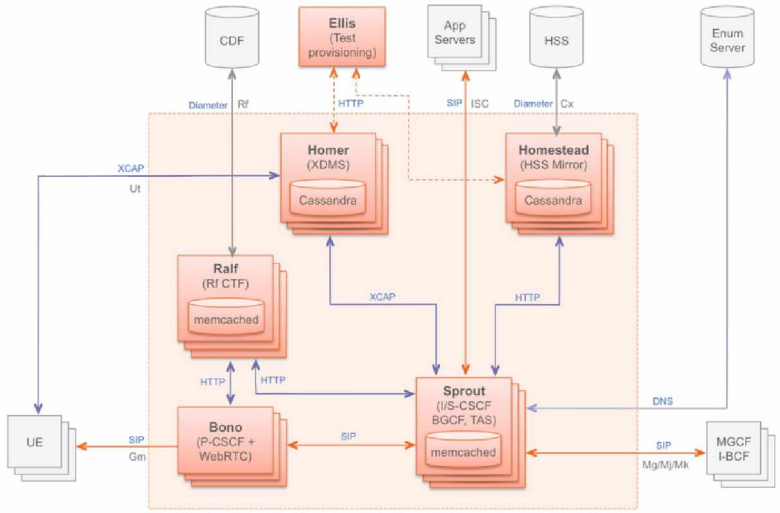

Figura 6 – Arquitetura do *Clearwater*.

Fonte: (CLEARWATER, 2015).

um portal de provisionamento, gerenciamento e configurações dos *Multimedia Telephony (MMtel)*.

2.5.2 Arquitetura *OpenIMS Core*

Projeto iniciado em 2004 pelo instituto de pesquisa *Fraunhofer FOKUS*, objetiva simular e avaliar performance de redes *NGN* por meio de entidades virtualizadas. A solução é composta por elementos de controle *CSCF* especificamente *Interrogating, Serving* e *Proxy* também um repositório de dados *HSS*. A implementação seguiu critérios estabelecidos pela entidade padronizadora *3GPP* (OPENIMS, 2016).

Muitos trabalhos encontrados na literatura realizaram experimentos utilizando essa arquitetura. Trabalhos que vão desde experimentos de soluções de vídeo sobre *IP* a soluções intermediadoras que gerenciam aspectos e comportamentos do *IMS cloud computing* (MAGEDANZ; SCHREINER, 2014; FAROOQI; MUNIR, 2008; LU; DOUSSON; KRIEF, 2015).

2.5.3 Arquitetura *Kamailio*

Outra abordagem de arquitetura de rede baseada em *IMS*, escrita em linguagem de programação *C*, é denominada *Kamailio*. Essa tecnologia é basicamente um *Open Source SIP Server*, capaz de lidar com centenas de chamadas por segundo. Possui boa integração com tecnologias como *WebRTC*, extensível para trabalhar com entidades *Long-Term Evolution (LTE)*, balanceamento de carga, operação com banco de dados *MySQL*, *Postgres*, *Oracle*; autenticação por tecnologia *Radius* possibilidade de monitoramento por *Simple Network Management Protocol (SNMP)* (KAMAILIO, 2016).

A característica principal dessa arquitetura *IMS* é que ela é modular, isto é, funcionalidades podem ser incorporadas apenas com chamadas de módulos específicos e configuração de parâmetros que implementam a funcionalidade desejada. Entidades de *CSCF* especificamente *Interrogating, Serving* e *Proxy* são possíveis de serem configuradas de forma distinta. Nesse sentido é possível incorporar soluções de escalabilidade com objetivo de tratar com eficiência determinada carga de trabalho.

A Figura 7 ilustra a arquitetura e seus relacionamentos. Devido essas arquiteturas serem padronizadas pelo *3GPP* existem interfaces que são comuns aos seus pares de implementações *IMS*, inclusive as vistas acima, nas subseções 2.5.2 e 2.5.1. O que ocorre são diferenças básicas de abstrações e granularidade de funcionalidades.

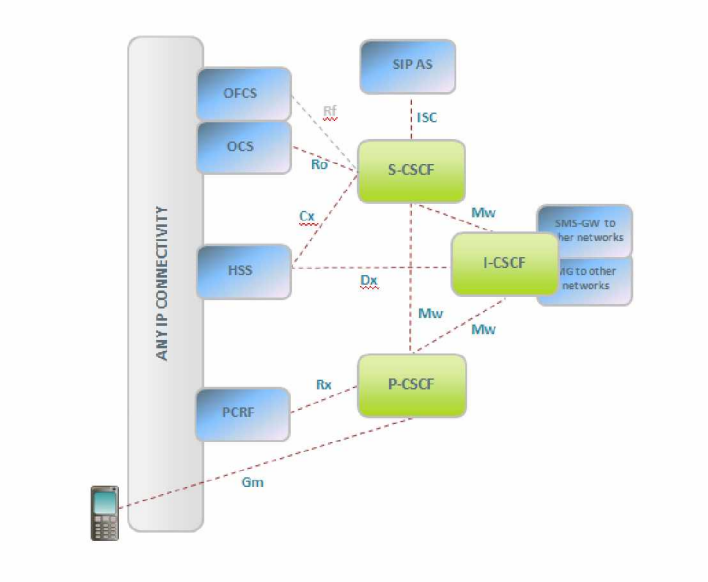

Figura 7 – Arquitetura do *Kamailio*.

Fonte: (KAMAILIO, 2016).

2.5.4 Interfaces das entidades *IMS*

As entidades da arquitetura *IMS* interagem entre si com trocas de informações para a continuidade e manutenção do plano de dados e controle. A seguir é descrito algumas interfaces comumente observadas em implementações:

- Interface *Gx*, localizada entre o *Policy and Charging Rules Function (PCRF)*[9] e *Policy and Charging Enforcement Function (PCEF)* com objetivo de instruir provisionamento e remover regras de qualidade de serviço (ETSI, 2008);

[9] Entidade responsável por aplicar políticas de *QoS* em seções *multimídia*

- Interface *Rx*, localizada entre *PCRF* e o *P-CSCF*, dentre outras funcionalidades basicamente é responsável pela escrita da política de *QoS* na seção alocada para o cliente (ETSI, 2013a);
- Interface *Rx/Dx*, são as interfaces de comunicação entre *S-CSCF* e *I-CSCF* para troca de informações com o *HSS*, dentre outras informações basicamente são trocados informações como localização de informação, autorização de acesso *IMS*, encaminhar informações da autenticação, carregar informações do usuário da base de dados (ETSI, 2013b);
- Interface *Ro* e *Rf*, presente na interligação do *S-CSCF* entre *Online charging system (OCS)* e *Off-line charging system (OFCS)*, sistema de faturamento *online* e *off-line*. A interação dessas entidades dentre outras funcionalidades tem objetivo de subsidiar informação para faturamento baseado na política do operador (ETSI, 2006);
- Interface *ISC*, presente na interligação entre *Application Server (AS)* e *S-CSCF* para subsidiar troca de informações para manutenção e execução das aplicações. Dentre outras funcionalidades, basicamente nessa interface é informado as aplicações em detalhes da seção e usuário (ETSI, 2009);
- Interface *Mw*, presente na conexão entre *S-CSCF*, *I-CSCF* e *P-CSCF* permite a comunicação e encaminhamento de mensagens de sinalização durante processos de registro de usuário e controle de seção. A troca dessas mensagens segue o padrão *SIP* (ETSI, 2015a).

Todas entidades trabalham com fim específico, seja no encaminhamento do plano de dados ou controle. Dependendo do que é requisitado na arquitetura *IMS* a tratativa pode ser diferente, o que implica em interações com entidades específicas. Comumente é requisitado registro de usuário, convite para chamada, publicação de presença e outros. Assim essas interfaces são essenciais na troca de informações para manter a arquitetura em operação.

2.5.5 Interação das Entidades *IMS*

As entidades *IMS* interagem entre si por meio de interfaces com mensagens contendo descrição de métodos para o serviço requisitado. Conforme descrito na padronização (IETF, 2002), são métodos do protocolo *SIP*: *INVITE*, *REGISTER*, *CANCEL*, *PUBLISH* e outros. Os diagramas ilustrados nas Figuras 8, 9 e 10 permitem compreender exemplos de interações comuns entre entidades *IMS* quando requisitado algum serviço.

A interação para registro de um usuário é descrita:

1. O usuário encaminha ao *P-CSCF* uma mensagem contendo o método *REGISTER*, nesse método é encaminhado o *password* do usuário (quando houver), parâmetros

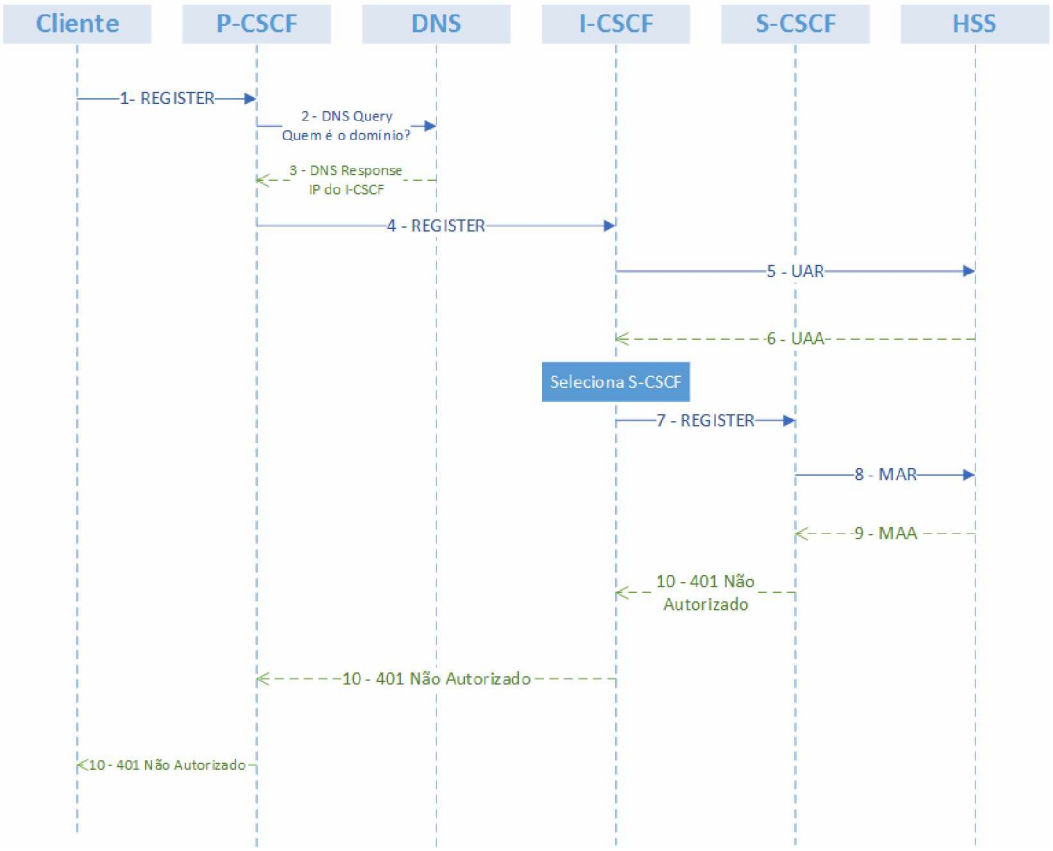

Figura 8 – Insucesso de Registro de Usuário.

Fonte: Adaptado de (ETSI, 2001; EVENTSTUDIO, 2007).

de domínio de rede e também é informado sua identidade pública de usuário. Na mesma mensagem, de forma encapsulada segue a identidade privada do usuário;

2. A entidade *P-CSCF* faz uma consulta no *DNS* com intuito de descobrir o *IP* do *Interrogating* que responde pelo domínio *IMS*;

3. O *DNS* responde com o endereço *IP* do *I-CSCF*;

4. Imediatamente a entidade *P-CSCF* encaminha novamente a mensagem, após modificar detalhes no cabeçalho o método *REGISTER* para a entidade *I-CSCF* tratar;

5. A entidade *I-CSCF* encaminha ao *HSS* a mensagem de requisição de autenticação do usuário – existem vários métodos segurança que o *HSS* pode adotar para a autorização, *Sha1*, *MD5*, *Radius* e outros (IETF, 2006);

6. Posteriormente o *HSS* responde com uma mensagem de resposta da autorização do usuário, na mesma mensagem é retornado um conjunto de endereços de *S-CSCF* capazes de efetivar o registro do usuário;

7. Ao receber o endereço do *S-CSCF* (ou conjunto deles) a entidade *I-CSCF* encaminha a mensagem contendo o método *REGISTER* para o *S-CSCF* escolhido;

8. Ao receber a mensagem com o método *REGISTER* o *S-CSCF* encaminha uma mensagem de requisição de autenticação de multimídia para o *HSS*;

9. Tão logo para o cenário descrito o *HSS* responde a com uma mensagem resposta de autenticação de multimídia onde é negado a consolidação do registro (para efeitos de exemplo);

10. A mensagem de não autorizado é propagada respectivamente aos elementos *S-CSCF*, *I-CSCF* *P-CSCF* finalmente ao usuário – assim, o registro não é efetivado.

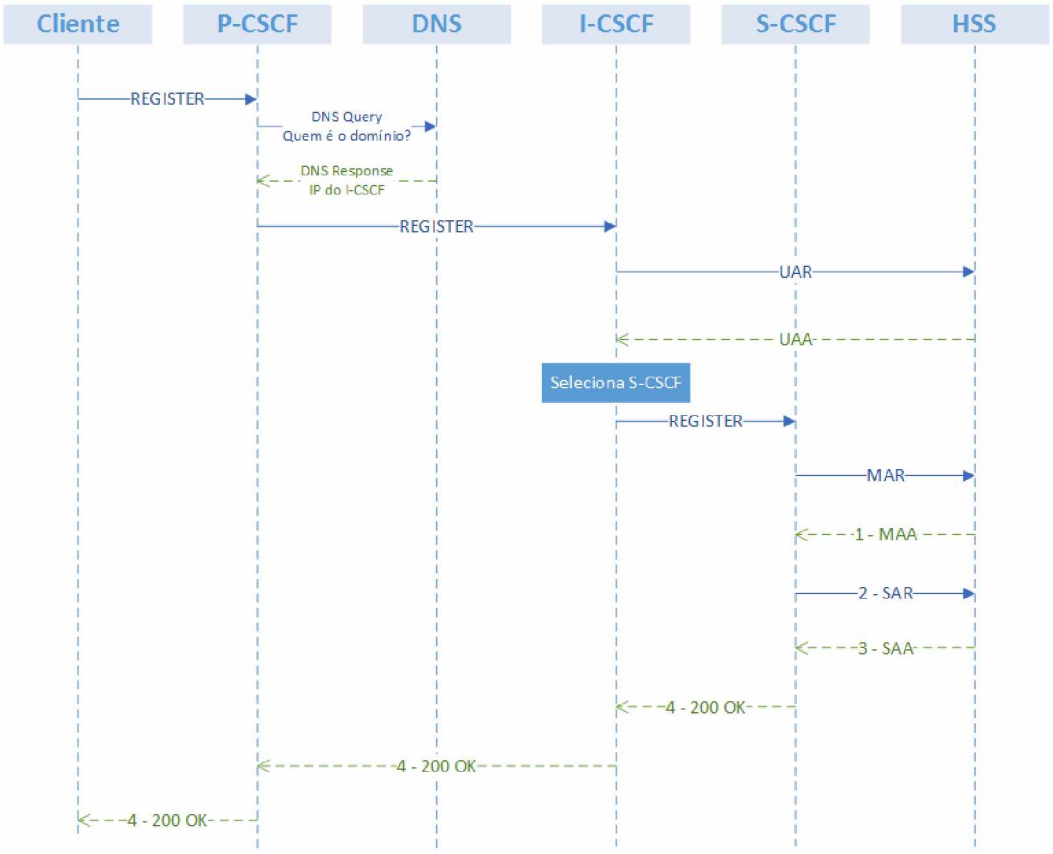

Figura 9 – Sucesso de Registro de Usuário.

Fonte: Adaptado de (ETSI, 2001; EVENTSTUDIO, 2007).

De forma análoga, um caso de sucesso de registro ilustrado na Figura 9 (detalhes anteriores foram suprimidos) é descrito conforme os passos:

1. A mensagem de resposta de autenticação é respondida ao *S-CSCF* onde é autorizado prosseguir com registro;

2. Posteriormente a entidade *S-CSCF* encaminha uma mensagem de solicitação de atribuição do servidor ao *HSS*;

3. Uma resposta de atribuição de servidor é devolvida ao *S-CSCF*;

4. Mensagem de sucesso 200*OK* é encaminhada respectivamente as entidades *S-CSCF*, *I-CSCF*, *P-CSCF* e cliente.

Outro exemplo de interação e troca de mensagens pelas entidades *IMS* se dá para estabelecimento de chamada entre dois usuários – efetivamente a execução do método *INVITE*. A Figura 10 ilustra por meio de um diagrama a sequência de troca de mensagens.

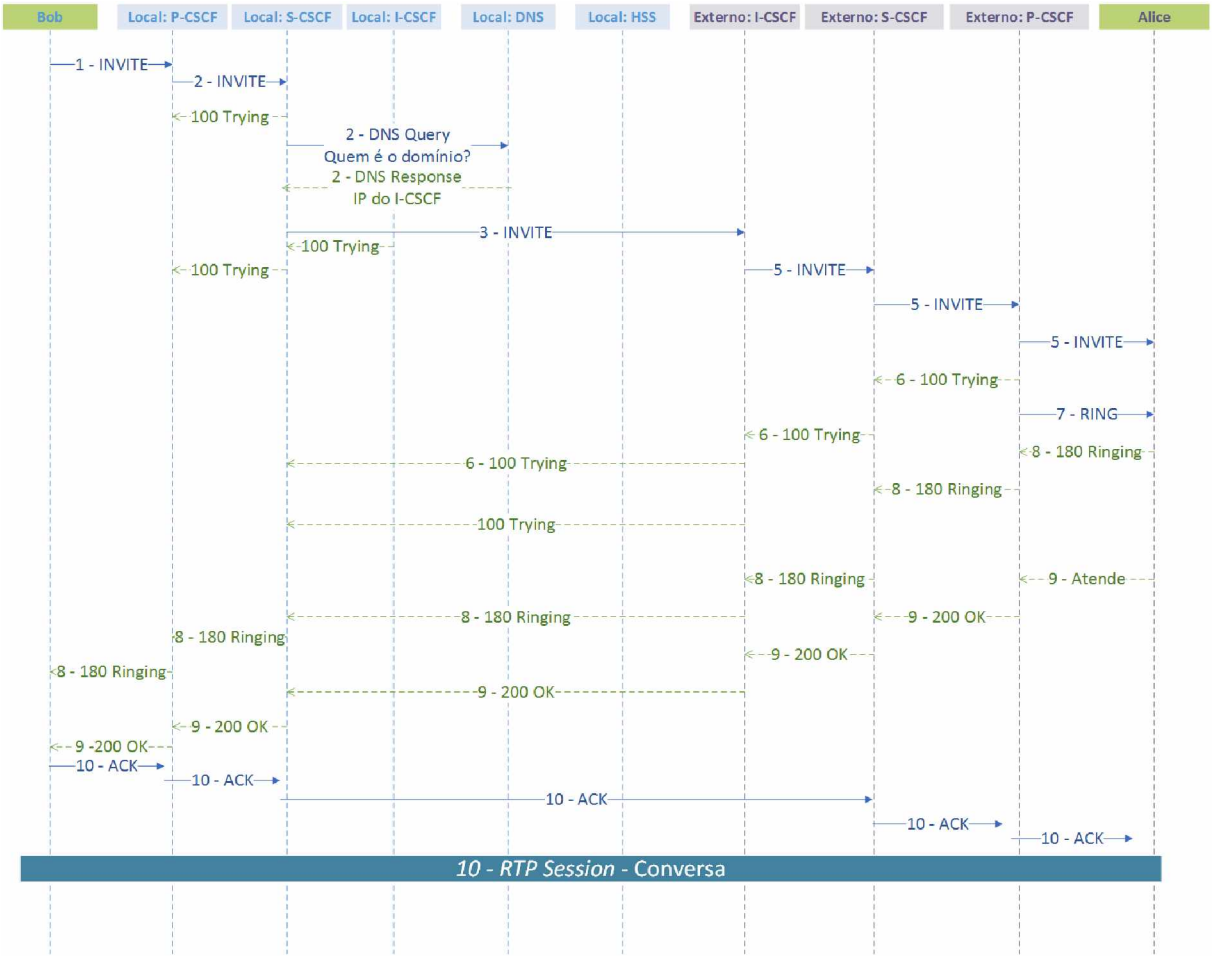

Figura 10 – Sucesso de Chamada entre dois Usuários.

Fonte: Adaptado de (ETSI, 2001; EVENTSTUDIO, 2007).

1. Uma vez registrado (conforme ilustração anterior), o usuário *Bob* encaminha ao *P-CSCF* um convite contendo a identidade pública de *Alice*;

2. A entidade *P-CSCF* ao receber a mensagem contendo o método *INVITE* encaminha para *S-CSCF* mediante consulta ao *DNS* para recuperar o endereço *IP* alvo;

3. Conforme caso exemplo, *Alice* não está no mesmo domínio *IMS*, desta forma, a entidade *S-CSCF* encaminha o convite ao *Interrogating* da rede externa;

4. A entidade "Externo: *I-CSCF*" recebe a mensagem contendo o método *INVITE* e a encaminha para o "Externo: *S-CSCF*" de seu domínio;

5. De forma análoga o roteador "Externo: *S-CSCF*" encaminha a mensagem recebida para o "Externo: *P-CSCF*" que a repassa ao usuário *Alice*;

6. Após, a entidade "Externo: *P-CSCF*" encaminha a mensagem 100*Trying* respectivamente para as entidades "Externo: *S-CSCF*", "Externo: *I-CSCF*", "Local: *S-CSCF*";

7. Assim, a entidade "Externo: *P-CSCF*" inicia o processo de chamada ao usuário *Alice*;

8. O usuário *Bob* deve ser informado desse evento, portanto mensagem 180*Ringing* é encaminhado respectivamente pelas entidades "Externo: *P-CSCF*", "Externo: *S-CSCF*", "Externo: *I-CSCF*", "Local: *S-CSCF*", "Local: *P-CSCF*" e finalmente ao chamador;

9. Após a chamada ser atendida pelo usuário *Alice* uma mensagem de confirmação contendo 200*OK* deve ser encaminhada ao chamador através da interação das respectivas entidades "Externo: *P-CSCF*", "Externo: *S-CSCF*", "Externo: *I-CSCF*", "Local: *S-CSCF*", "Local: *P-CSCF*" finalmente entregue ao usuário *Bob*;

   Ao recebe-la o usuário *Bob* envia uma mensagem de confirmação *ACK* com destino ao usuário *Alice*, respectivamente através das entidades "Local: *P-CSCF*", "Local: *S-CSCF*", "Externo: *S-CSCF*", "Externo: *P-CSCF*";

10. Após, uma sessão *RTP* entre os usuários *Bob* e *Alice* é estabelecida.

2.6 Trabalhos Relacionados

Vários trabalhos investigam a construção de sistemas multimídia sobre nuvem (CHAMORRO; CASTILLO; LOPEZ-PIRES, 2016; PODDAR; VISHNOI; MANN, 2015; REGO et al., 2015). Em Chamorro, Castillo e Lopez-Pires (2016) os autores apresentam uma solução *auto scaling* de serviços *VoIP*. Os serviços ofertados são sinalização e *proxy* de chamadas em curso. Os componentes são balanceador de carga *Kamailio Load Balancer* e os roteadores de chamada *Kamailio Router*, todos esses componentes são instâncias orquestradas em uma infraestrutura *OpenStack*. A objetividade da solução basicamente consiste na utilização dos serviços adicionais do *OpenStack*. São eles *Heat* para orquestração de instâncias e o *Ceilometer* para monitorar estatística de *hardware* da instância, a saber utilização de *CPU* e memória.

Estaticamente é definido um limite para o consumo de *CPU* das instâncias que executam serviço de roteamento de chamada. Todas as instâncias disponíveis compõem um *pool* de elegíveis para receberem a carga balanceada. O *auto scaling* ocorre por meio do módulo *Ceilometer* do *OpenStack*, uma vez extrapolado o consumo de *CPU* por período predeterminado de tempo o módulo *Ceilometer* dispara uma rotina de alarme. Uma vez alarmado, o agente *Python* desenvolvido invoca o serviço *Heat*, através de uma *API* do *OpenStack* para criação de um novo roteador de chamada. O *down scaling* das instâncias criadas mediante a carga de trabalho utiliza o critério que considera, se um roteador de chamada está com 5% ou abaixo disso de utilização, a máquina deverá ser excluída do *pool* de instâncias elegíveis para tratar chamadas.

Poddar, Vishnoi e Mann (2015) propõem um balanceador de carga intra *cloud*. O balanceador faz interação direta com módulos do *OpenStack*, como *Neutron* e *Ceilometer*. Efetivamente o trabalho consiste numa aplicação *SDN* sobre o controlador *OpenDayLight*. Portanto a solução apresentada executa no mesmo domínio *L2* do serviço de rede e das aplicações cliente-servidor, isso que permite alcançar a característica holística que o trabalho sustenta. Uma vez que é possível monitorar via *API* todos os recursos de *network* e *compute* do *OpenStack*.

A aplicação possui algoritmos que avaliam o consumo de cada recurso de *network* e *compute*. O consumo de recursos de rede é monitorado pelo módulo *Statistics Manager*, que recebe *reports* de consumo de *links* entre os *switches OpenFlow*. O consumo de recursos de *compute* é monitorado via consumo de *API* com *Ceilometer*, *reports* de consumo de *CPU* e memória são ofertados para o módulo *Data Orchestrator*. Todos esses *reports* são tratados por um algoritmo que calcula o *score* (especificação quantitativa de consumo) de cada recurso.

O recurso que tiver o menor *score*, será informado ao módulo *LBaaS Provider* que por sua vez proverá escrita de regras com intuito de efetivar balanceamento, tanto de *network* no que diz respeito a *links* e *switchs*, quanto *compute* onde as aplicações cliente e servidor executam. No mesmo sentido, *reports* do consumo de *compute* são tratados pela aplicação desenvolvida onde o algoritmo avalia por meio do nível de utilização a necessidade efetivar *up scaling* ou *down scale* de instâncias que ofertam serviços.

Em Rego et al. (2015) é explorado uma solução de balanceamento de carga para servidores de mídia. A solução é comparada com outra abordagem de *proxy* denominada *HAProxy* (TARREAU, 2012), cujo o qual trabalha como *proxy* sobre pacotes *TCP* e aplicações baseadas em *HTTP*. O trabalho sustenta a hipótese da dificuldade de promover balanceamento de carga de pacotes *UDP* em cenários de nuvem, para tanto, objetivamente propõe balanceamento de carga *UDP* por meio de *SDN*. A solução se limita em operar com infraestruturas orquestradas pela tecnologia *OpenNebula* (OPENNEBULA, 2017).

A arquitetura da solução consiste em um *load balancer* que faz intermediação das requisições de *streaming* de vídeo solicitadas pelos clientes, e efetivamente a delegação de

qual servidor de mídia irá tratar a requisição. Também, o *load balancer* possui interface com a infraestrutura de nuvem e outra entidade denominada orquestrador. Esse por sua vez, interage por meio de *API* com os *media servers* e com *load balancer* simultaneamente.

A tecnologia de serviço de *streaming* utilizada é *Darwing Media Server*. Essa tecnologia, de forma nativa, fornece dados da aplicação ao orquestrador; dados como quantidade de clientes recebendo *streaming*, *jitter* e taxas como *packet lost*. O orquestrador por sua vez, ao utilizar o modelo de referência *MAPE-K* Huebscher e McCann (2008) recebe estatísticas do consumo de *hardware* das instancias, e da aplicação de *streaming*, para que consiga promover *scaling up* ou *scaling down* baseado na política definida. A política que foi definida no trabalho consiste em se um *media server* estiver com 95% de utilização, execute uma nova instância de *media server*. Por outro lado, se tiver com utilização abaixo de 60%, exclua ou suspenda a instância.

O trabalho de Carella et al. (2015) propõe a construção de um *VNF* capaz de realizar balanceamento de carga entre *application servers*, encaminhamento de tráfego de sinalização e monitoramento de consumo de *CPU* e memória dos *Application Servers*. A considerar a topologia padrão *IMS*, a solução desenvolvida propõe uma entidade *VNF* intermediária ao *S-CSCF* e *Application Servers*. Desta forma, todo o tráfego destinado aos *Application Servers* transpassa a solução que por sua vez trabalha pela sustentação da hipótese de garantia de *QoS* para aplicações multimídia.

O *VNF* desenvolvido trabalha em conjunto com uma entidade externa de controle. De forma serial, a entidade de controle recebe estatísticas de consumo de *CPU* dos *Media Servers*, trabalha essas estatísticas no sentido de avaliar mediante a *KPIs* definidos se é necessário executar ação de *up scaling* ou *down scaling* de *Media Servers*. Uma vez avaliado, a entidade externa interage com *NFV Orchestrator (NFVO)* que fará orquestração dos *Media Servers*, que poderá ser agregação de mais recurso ou *release* de recurso alocado.

O trabalho de Tranoris et al. (2013) concentra numa abordagem *SDN* que provê *QoS* na comunicação entre dispositivos que consomem serviços de multimídia. A solução desenvolvida prioriza tráfego *SIP* transportados sobre infraestrutura de rede *OpenFlow*. E provê garantia de banda, a carga na rede consiste em tráfego *RTP*. A solução está sobre uma infraestrutura de *test bed OSIMS* com entidades *IMS* como *Policy and Charging Rules Function (PCFR)*, *Policy and Charging Enforcement Function (PCEF)* que trabalha em cooperação com o *IMS Core*.

O *Proxy IMS* ao receber requisições de chamada por meio do método *SIP INVITE*, encaminha ao *PCRF* características do *codec* de voz e requisitos de rede, que são transportadas no corpo *SDP* do pacote. Por sua vez, o *PCRF* interage com *PCEF* que avalia a requisição e descreve como regras de *flows* para o controlador *SDN* via *REST API*. O controlador *FloodLight* materializa a regra no contexto da rede.

Está organizado na Tabela 2 alguns trabalhos e aspectos conceitualmente similares a

presente proposta. O critério de Escalabilidade é importante para definir o quão robusta é uma solução no sentido de prover continuidade na oferta do serviço. Consequentemente garantir satisfação dos usuários mesmo em cenários de alta utilização, pois os recursos podem ser alocados sob demanda. Métricas de *QoS* que as entidades centrais das soluções utilizam para ponderar a operação de suas funcionalidades, as métricas devem ser pertinentes para o contexto de aplicações multimídia e seus requisitos, tanto métricas centradas na rede quanto nos usuários Huong-Truong et al. (2013). Visão Global, refere-se ao escopo da solução em termos de holística e gerenciamento integrado de recursos. Por fim, o aspecto Flexibilidade diz respeito ao quanto a solução é acoplada a tecnologia e principalmente se é capaz de exercer influência baseado no contexto da aplicação nos recursos de rede e de computação simultaneamente.

Tabela 2 – Propostas do Estado da Arte.

Trabalho	Provê Escalabilidade	Métricas de QoS	Visão Global	Provê Flexibilidade	Critério para tomada de decisão
(PODDAR; VISHNOI; MANN, 2015)	Não, a solução não provê escalabilidade.	Latência do controlador *SDN*.	Sim.	Não, trabalha apenas com controlador *SDN* específico e com *OpenStack Neutron*. Não realiza quaisquer influência em recursos baseados no contexto da aplicação.	*Threshold* do uso da memória, *CPU* e *bandwidth*.
(MUELLER; MAGEDANZ, 2012)	Não, a solução não provê escalabilidade.	*Bitrate* em *streaming* de media.	Não, a solução não conhece consumo de *CPU* ou memória das entidades ofertantes do serviço.	Parcial, baseado no contexto da aplicação a solução exerce influência nos recursos de rede, somente.	Condições do tráfego de rede, políticas previamente definidas, e coleta de métricas através do equipamento do usuário enviadas a entidade *GARC*.
(REGO et al., 2015)	Sim, a solução provê escalabilidade.	*Jitter* e taxa de perda de pacotes.	Sim, todas entidades da solução são monitoradas por uma entidade central.	Não, a solução é fortemente acoplada ao *OpenNebula* e não exerce nenhuma influência em recursos baseada no contexto da aplicação.	*Threshold* do uso de *CPU*.
(CARELLA et al., 2015)	Sim, a solução provê escalabilidade.	Taxa de Registro e Chamadas por segundo	Não, os componentes *IMS* executam em diferentes *VMs* e a solução não consegue ver isso.	Parcial, baseado na aplicação a solução consegue exercer influência nos recursos de *compute*, somente.	*Threshold* do uso de *CPU*.
(HAHN; GAJIC, 2015)	Sim, a solução provê escalabilidade.	Transações por segundo, latência e *bandwidth*.	Não, a solução apenas conhece a rede interna do *data center*.	Não, a solução não exerce nenhuma influência em quaisquer recursos baseado no contexto da aplicação.	Predição de tráfego com padrões definidos *a priori* e *threshold* do uso de *CPU*.

Baseado nos conceitos apresentados, ferramentas, tecnologias e nos aspectos destacados no estado da arte, é possível identificar oportunidade de contribuições no âmbito de gerenciamento de recursos para aplicações multimídia em nuvem. Posteriormente serão discutidas as características do presente trabalho e seu contexto cientifico.

Capítulo 3

O Orquestrador Multimídia

Este Capitulo descreve a especificação conceitual da proposta desenvolvida para lidar com aprimoramento de *QoS* em aplicações multimídia e gerenciamento eficiente de recursos. Para tanto são discutidos detalhes dos componentes e suas funcionalidades.

Conceitualmente há uma barreira entre aplicação e rede (COSTA, 2013; SHARKH et al., 2013; LIU et al., 2015; DERAKHSHAN et al., 2013), uma vez que as aplicações para transmitirem dados estabelecem um canal (ou *socket*) e entregam os dados para a pilha de protocolos de rede que por sua vez proverá a tentativa de entrega. O cenário descrito, reforça que a camada de rede é agnóstica em relação à aplicação que a utiliza. De forma análoga, a aplicação comumente desconhece limitações de *hardware* sobre o qual elas operam (SHARKH et al., 2013). Alta utilização de processamento faz com que as aplicações concorram por recursos escassos, o que ocasiona queda de desempenho. Nesse sentido, o presente capítulo descreve a implementação de uma solução pertinente; cujas características são flexibilidade, orientação a recursos de *compute* e *network* em aplicações multimídia ofertadas em nuvens.

3.1 Características

O trabalho desenvolvido foi construído à luz do conceito *NFV Management and Orchestration (MANO)*. O *MANO* é composto de três blocos fundamentais propostos para o *framework* (ETSI, 2015b):

- (1) *NFV Orchestrator* que diz respeito ao gerenciamento global dos recursos e gerenciamento de políticas das instâncias virtualizadas. O trabalho desenvolvido lida com gerenciamento global dos recursos e com políticas específicas para balanceamento de carga;
- (2) *NFV Managers* que cuida do ciclo de vida dos recursos virtualizados. O trabalho desenvolvido controla do ciclo de vida dos servidores multimídia e da função de rede *proxy RTP*;

- (3) *VIMs* para controle e gerenciamento dos recursos de computação, rede e armazenamento. O trabalho desenvolvido lida com aspectos do tipo de instâncias a serem iniciadas, tamanho do volume (serviço de *storage* das instâncias) e a conectividade por meio do conceito de *IPs* públicos.

Como pode ser visto na Figura 11, a solução desenvolvida se comporta como orquestrador na perspectiva *NFV*. Assim, o trabalho desenvolvido será referenciado com a terminologia *Orchestrator*.

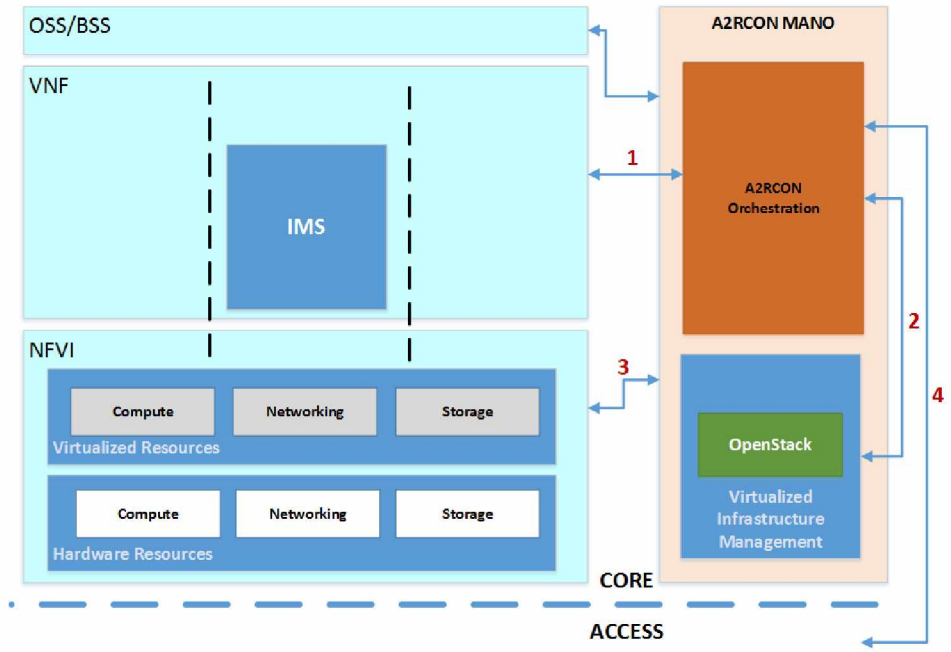

Figura 11 – *Orchestrator* contextualizado na arquitetura de referência *NFV ETSI*.

Na Figura 11, a ligação 1 representa a atuação do *Orchestrator* para gerenciar as *Virtual Network Functions (VNFs)*, para isso, ele mantém uma lista de servidores e funções em seu repositório e aplica políticas de despacho de carga a cada um deles. A ligação 2, representa a atuação do *Orchestrator* no controle dos recurso de computação (*storage, compute* e *network*) por meio da interface disponível do sistema gerenciador de nuvem. Também, o *Orchestrator* define característica do *flavor* e o sistema operacional hospedeiro da *VNF*, define grupos de segurança e *IP* público para cenários de serviços multi-domínio. A ligação 3 representa a operação do *Orchestrator* ao gerenciar o ciclo de vida das *VNFs*, em termos de elasticidade e escalabilidade, termino de instância e gerenciamento de integridade. Finalmente, a ligação 4 refere-se a interação que o *Orchestrator* possui com a rede de acesso, essa interação é detalhada posteriormente.

A Figura 12 ilustra o contexto do *Orchestrator*. A figura é fatiada em quadrantes de duas camadas. As camadas contemplam controle e carga útil de aplicações. No centro se localiza o *Orchestrator* que eventualmente pode prover interação com a camada de rede por meio de *API* com controladores *SDN*, interação com sistema gerenciador de

nuvem para orquestrar recursos de *compute* e também aproxima a aplicação com esses dois universos ao prover balanceamento de carga dentre o *pool* de recursos disponíveis e efetuar influência baseado no contexto da aplicação multimídia.

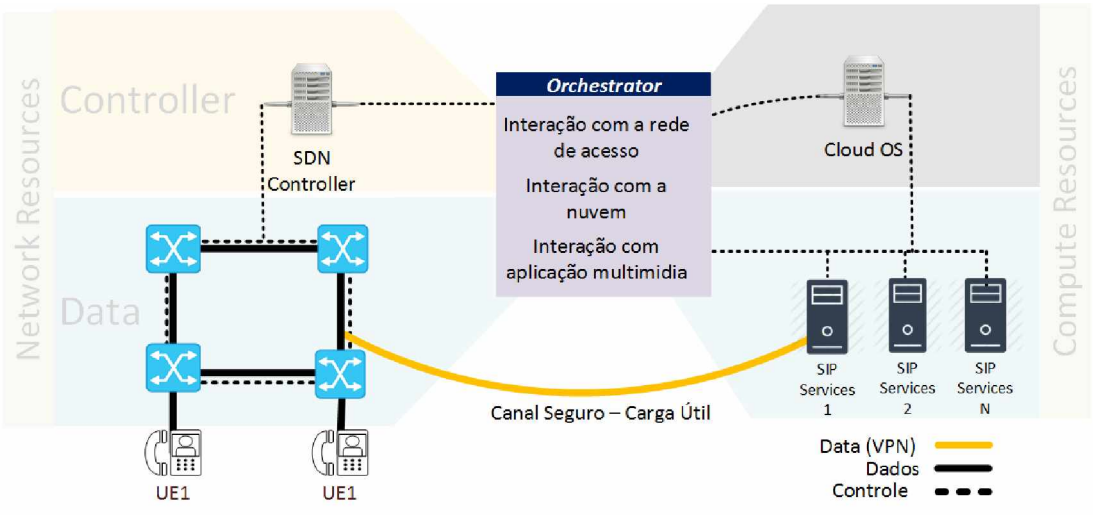

Figura 12 – O *Orchestrator*.

Do lado esquerdo da Figura 12 concentram-se os recursos de rede que são *hosts*, *switches*, elementos *WiFi* e controlador *SDN*. O *Orchestrator* trata os recursos de rede pela perspectiva de controle e dados. Neste ambiente, *hosts* são entidades que podem consumir ou ofertar serviços para seus pares, uma vez que tenham conectividade entre si. No mesmo sentido os *switches* do ambiente devem operar em compatibilidade com protocolos *OpenFlow* para possibilitar controle por meio de controlador *SDN* para efetivamente realizar a separação do plano de dados e controle.

A parte direita da Figura 12 ilustra os recursos de *compute* divididos em nível de dados e controle. Recursos de *compute* são instâncias que ofertam serviço de multimídia no modelo *IaaS*. O controle desses recursos de forma centralizada e externa ao ambiente da *cloud* logra da vantagem de ser flexível pois facilmente pode ser adaptada para trabalhar com diversos sistemas gerenciadores de nuvem e multiplicidade de domínios de acesso. Isto é, a solução não depende dos sistemas de telemetria que são ofertados pelos gerenciadores de nuvem convencionais para por exemplo examinar estatísticas de consumo de hardware das instâncias. Também, não necessita de configuração de alarmes para prover *scale up* ou *scale down* em cenários de *over load*. O plano de dados é onde ocorre a transmissão de carga útil consumida pelas aplicações multimídia.

3.1.1 Capacidades do *Orchestrator*

O *Orchestrator* é uma solução que propõe encurtar a distância entre os recursos de *compute*, rede e as aplicações. Como mencionado anteriormente, há esforços concentrados em gerenciamento de recursos de *compute* com objetivo de prover *QoS* para aplicações

multimídia, no mesmo sentido, há esforços dedicados com o mesmo desafio no âmbito de rede. Portanto, concentrar essa capacidade de influência em uma entidade externa possibilita definir uma solução de rede orientado à aplicação. O *Orchestrator* propõe exercer influência de forma simultânea nos recursos de rede, *compute* e aplicações multimídia, tais capacidades distingue-o de demais propostas do estado da arte, que são incompletas nesse sentido.

Assim é necessário destacar questões importantes que o *Orchestrator* lida como *QoS* para as aplicações multimídia que são executadas sobre a rede. Ademais, é possível por intermédio do *Orchestrator* definir banda de *links* onde há fluxo multimídia. Essa influência pode ocorrer em tempo de execução mediante verificação da aplicação que está a executar sobre a rede. No mesmo sentido, a carga sobre o *hardware* dos servidores das aplicações multimídia compromete a garantia de *QoS*, pois em situações de *over load* não é possível garantir a integridade do processamento de sinalização multimídia, e há de se constatar elevados tempo de resposta.

É possível alcançar resiliência na sessão multimídia através da definição de *Intents* sobre os recursos de rede. Controladores *SDN* conhecidos do mercado já implementam o *framework*, como exemplo o *OpenDayLight* (OPENDAYLIGHT, 2015) e *ONOS*(ONOS, 2015). O *framework Intents* se empenha em proteger a operação da rede em cenários de configurações errôneas, falhas de entidades intermediarias e mudanças de topologia (GKOUNIS et al., 2016; BONDKOVSKII et al., 2016; SCOTT-HAYWARD, 2015). Com uso de *Intents* o controlador *SDN* calcula caminho alternativo para encaminhamento de fluxo já definido. Assim, o *Orchestrator* avalia o conteúdo da troca de mensagens de sinalização onde ocorrerá tráfego multimídia sobre os *switches* e interage com controlador *SDN* por meio de *north-bound interface (NBI)* com objetivo instalar uma *Intent* e lograr de garantias mencionadas.

Outra funcionalidade do *Orchestrator* é a capacidade de inspecionar pacotes. Este comportamento foi implementado com objetivo viabilizar a tomada de decisões automáticas baseado nas aplicações que estão a executar. A inspeção de pacotes é importante para definir como o controlador *SDN* deve atuar (ou outra entidade de controle dependendo do cenário), isto é, se mediante análise do tráfego de sinalização for constatado o término de fluxo multimídia a *Intent* é destruída de forma que não sobrecarregue o controlador.

Também, a característica de flexibilidade consagra o *Orchestrator*, pois não se faz necessário modificação estrutural da solução para interoperabilidade entre controladores *SDN* e também de sistemas gerenciadores de nuvem. No âmbito dos recursos de computação a solução independe de estatísticas de consumo de *CPU* ou memória fornecidas pelo serviço de telemetria da nuvem. Tais estatísticas das instâncias são reportadas por uma aplicação *daemon* desenvolvida. Isto possibilita monitoramento mais preciso da saúde das instâncias e acompanhamento histórico de consumo de forma gráfica.

3.1.2 Interface com a rede

As interações com os recursos de rede são necessárias pois elas modificam o comportamento da rede em função da aplicação em execução. O *Orchestrator* utiliza uma *API Rest* para requisitar serviços do Controlador *SDN*. O *ONOS* é um Controlador que desde sua primeira versão 1.0.0 traz consigo maior quantidade de serviços que podem ser consumidos por *API*. A partir da *release* 1.4.0 foi incorporado a possibilidade de instalar *Intents* por meio da *API Rest*, o que era possível somente via linha de comando nas versões anteriores (mesmo assim de forma incompleta). Hoje é possível recuperar várias informações dos *links*, estatísticas de *flows*, topologia da rede, *clusters* de controladores, menor caminho entre elementos entre outras. A atual versão do trabalho desenvolvido utiliza *release* 1.5.0 do *ONOS* por ser estável em relação a seus pares e também por possuir boa documentação disponível para comunidade.

Dado o fato do *Orchestrator* ser uma aplicação externa ao ambiente de acesso onde estão os recursos de rede bem com o controlador *SDN*, é necessário que haja conectividade *L3* com o controlador *SDN*. A conectividade pode ocorrer através de *gateway* de rede; *DNS*, para cenários de domínios *IMS*, nesse caso o *DNS* retorna o *IP* da entidade responsável para garantir a conectividade; *Virtual Private Network (VPN)* ou pela rede pública, internet. Após estabelecida uma conectividade, a principal interface do *Orchestrator* com os recursos de rede é por meio de chamadas na *API Rest* do controlador, nesse caso o *ONOS*. As chamadas são feitas por meio de *socket HTTP* com os parâmetros adequados entre o *Orchestator* e o *ONOS*. Um exemplo, para o caso do *Orchestrator* requisitar informações sobre métricas de um dispositivos de rede, é necessário enviar uma mensagem com cabeçalho *HTTP* contendo método de requisição *GET* e utilizar a *URL*: *http://<ONOS_IP>:8181/onos/v1/meters/{deviceId}*, onde *deviceId* representa o identificador do dispositivo mapeado pelo *ONOS*.

Conceitualmente, a interface do *Orchestrator* com os recursos de rede são especificamente para dados de controle em tempo de execução. Para o caso de sessões multimídia nenhuma carga útil irá transpor o *Orchestrator*, somente sinalização. Também há ocorrências de interações com os recursos de *compute* e aplicações, que são desencadeados por meio de percepções que o *Orchestator* tem da rede dado as características dos pacotes.

3.1.3 Interface com os recursos de *compute*

O *Orchstrator* possui três interfaces bem definidas com os recursos de *compute*. A primeira, possibilita acompanhamento do consumo de *hardware* de cada instância em operação. Esta interface foi desenhada com objetivo de ofertar uma solução de telemetria agnóstica aos sistemas gerenciadores de nuvem. Esta interface é baseada em *socket UDP* na porta 7070 que as instâncias abrem para se comunicar com o *Orchestrator*. Os dados trocados estão contidos em mensagem no formato *JSON* que contém o endereço *IP* da

instância virtualizada, a utilização de memória *RAM* seguimentada em: memória total, memória livre e memoria em *cache*; utilização de *CPU*: media de utilização em porcentagem, utilização corrente em porcentagem e ociosidade em porcentagem. Tais métricas foram escolhidas para possibilitar maior flexibilidade de incorporar futuramente outras funcionalidades no *Orchestrator*.

A segunda interface com os recursos de *compute* ocorre por meio do consumo da *API Rest* com sistemas gerenciadores de nuvem. A atual versão do trabalho desenvolvido incorpora o Kit de Desenvolvimento de Software (*Software Development Kit (SDK)*) do *OpenStack* na *release Mitaka* e *OpenNebula release* 4.12, entretanto extensões de Classe e Interfaces estão completamente funcionais para o *OpenStack*. Assim, para viabilizar a interação é necessário autenticar-se com credenciais no serviço de identidade do *OpenStack*, denominado *Keystone*. Uma vez autenticado, é devolvido junto com a mensagem de autorização um *JSON* com demais endereços dos serviços que o *OpenStack* oferece e o *token* da sessão. Endereço dos serviços: gerenciamento de imagem, gerenciamento de *compute* (*reset*, *pause*, *suspend*), gerenciamento de rede (alocação de *IP* flutuante) e gerenciamento de volume são armazenados pelo *Orchestrator* para utilização posterior. O *OpenStack* implementa uma camada de segurança ao exigir que toda requisição do *Orchestrator* entregue com a mensagem *HTTP* o *token* da sessão.

O *Orchestrator* se comunica com uma arquitetura *IMS* que oferta funções de rede disponíveis em nuvem que suportam aplicações multimídia. O protótipo desenvolvido considerou a utilização da arquitetura *Kamailio* e seus módulos para implementar esses serviços. Assim, o *Orchestrator* precisa se comunicar com os elementos dessa arquitetura, nesse sentido foi especificado a funcionalidade dessa interface.

A terceira interface do *Orchestrator* ocorre com o processo em execução que é responsável pelo balanceamento de carga entre os recursos de compute onde ocorre a oferta de aplicações multimídia. O *Kamailio* em conjunto com módulo *dispatcher* provê balanceamento de carga entre os recursos de *compute* disponíveis, servidores multimídia concebidos como função de rede virtualizada. O balanceamento é feito ao encaminhar a mensagem de sinalização via interface de rede, com destino ao recurso de *compute* escolhido. O módulo *dispacher* possibilita adicionar instâncias alvo para balanceamento, basta adicionar entradas na tabela *dispacher* do banco de dados com o endereço *IP* alvo. Mesmo que as instâncias não estejam em operação, o módulo busca em intervalos regulares de tempo resposta a mensagem *Internet Control Message Protocol (ICMP)*[1] por meio do utilitário *ping* que é endereçado ao conjunto *IPs* cadastrados.

[1] Protocolo utilizado para enviar relatório de erros à fonte original quando o destino é inacessível (DEERING, 1991).

3.2 Arquitetura

A arquitetura da solução é ilustrada na Figura 13 onde as principais funcionalidades, módulos e as interações do *Orchestrator* são destacadas. Para a solução foi utilizado o módulo *Kamailio Load Balancer* que executa como processo *daemon* no *Orchestrator*; o fato de ser *open source*, modular, versões atuais e possuir bom desempenho justifica a utilização. Assim, uma aplicação Java propõe executar rotinas no sentido de orquestrar recursos em cenários de utilização de aplicações multimídia. O *Orchestrator* utiliza a mesma placa de rede para saída de tráfego de balanceamento de carga, interação controlador *SDN* e interação com o gerenciador de nuvem. Nas seções subsequentes são tratados especifidades dos módulos, interfaces e funcionalidade do *Orchestrator*.

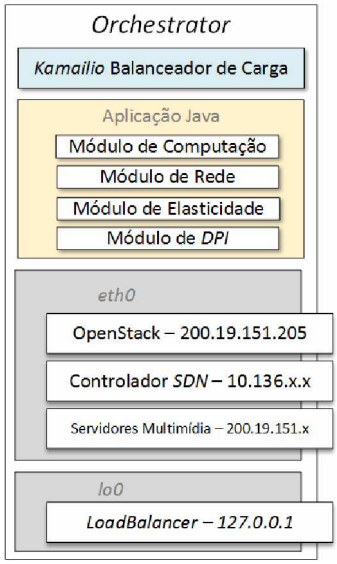

Figura 13 – Arquitetura do *Orchestrator*.

3.2.1 Módulo de Balanceamento de Carga

A principal funcionalidade do módulo *Kamailio* é prover tratamento de mensagens de sinalização multimídia endereçadas a ele. Isto permite que o *Orchestrator* se comporte como *proxy* por onde mensagens de sinalização multimídia deve transpassar e também atue como balanceador de carga. O Algoritmo abaixo descreve o comportamento geral do processo *Kamailio* frente ao recebimento de mensagens de sinalização *SIP*:

Para toda mensagem recebida pelo *Orchestrator* o processo *Kamailio* executa a rotina descrita no algoritmo acima. Na linha 2 é verificado se o método da mensagem de sinalização é *CANCEL*, quando for, na linha 3 a mensagem de sinalização será descartada. Na linha 5 ocorre a verificação de demais possibilidades de métodos na mensagem de sinalização. Se for *INVITE*, *SUBSCRIBE* ou *BYE* a mensagem de sinalização receberá na linha 6 em forma de *tag* o endereço *IP* do *Orchestrator*. Isso ocorre para que posteriormente

Algoritmo 1 Regra de Roteamento do *Kamailio*

```
1: route_method() {
2:         if (rcv_msg == CANCEL) {
3:             Drop the message;
4:         }
5:         if (rcv_msg == INVITE or rcv_msg == BYE) {
6:             Tag the message with Orchestrator IP (record_route);
7:         }
8:         dispatch_method();
9: }
10: dispatch_method() {
11:         if (Dispatch with round-robin algorithm to available destination == success) {

12:             LogPrint("Dispatch: going to (selected destination) via Orchestrator");
13:         }
14:         else {
15:             LogPrint("503 - Service Unavailable");
16:         }
17: }
```

quando a mensagem de sinalização alcançar o destino, a mensagem pode ser rastreável, isto é, saber por onde ela passou. E no caso de réplicas, o remetente poderá saber para onde devolver a mensagem (conceito *stateful*). Na linha 8 ocorre a chamada da função para balanceamento de carga entre os destinos disponíveis.

Uma vez chamada a função para balanceamento de carga, ocorre na linha 11 a escolha pelo critério *round-robin* qual será o destino da mensagem. Mesmo no critério *round-robin* que encaminha carga de sinalização para cada destino de forma intercalada isto é, um pouco para cada, é possível definir uma política de prioridade, onde 0 (zero) é o destino que possui maior prioridade – no desenvolvimento do *Orchestrator* essa característica foi explorada. É possível escolher outras políticas como distribuição de carga considerando atributos da mensagem de sinalização, distribuição de carga de forma aleatória, primeiro destino (*FIFO*) e outros. Quando houver sucesso na escolha e encaminhamento, na linha 12 é exibido em forma de *log* o registro do encaminhamento, contendo o destino para o qual a mensagem foi encaminhada e por quem foi encaminhada. Casos de insucesso no balanceamento são considerados na linha 15, que faz o *Orchestrator* devolver a mensagem ao requisitante informando o insucesso no encaminhamento da mensagem.

Dentro do contexto de balanceamento de carga é necessário definir as prioridades dos servidores multimídia em tempo de execução. Em cenários de alto desempenho, cargas de trabalho aumentam e diminuem, tal fato deve ser refletido de forma a expandir ou retrair o arsenal de servidores sem comprometer a oferta mínima em casos de baixíssima utilização. Portanto é necessário que o *Orchestrator* ajuste as entradas para o balanceamento em tempo de execução. Nesse sentido, a regra construída para ajuste de prioridades em

tempo de execução é dada como:

1. o serviço de sinalização multimídia conta com uma infraestrutura mínima de serviço, disponível a todo momento, mesmo em cenários de total ociosidade;

2. se no serviço multimídia a carga de trabalho aumentar, o *Orchestrator* de forma responsiva irá avaliar e iniciar a alocação de mais recursos, uma vez iniciado é atribuído ao servidor recém criado a maior prioridade para receber carga de trabalho;

3. se no serviço multimídia for constatado servidores com baixíssima utilização, o *Orchestrator* desapropria os recursos e ajusta as prioridades dos servidores remanescentes;

4. o ajuste de prioridades no cenário de retrair o arsenal de servidores decrementa em 1 (um) a prioridade dos servidores disponíveis, salvo os que já são 0 (zero), nesses por sua vez é mantida a prioridade.

Os destinos elegíveis para balanceamento são entradas no banco de dados *MySQL* presente no *Orchestrator* e contidas na tabela denominada *dispatcher*. Os campos da tabela são: *destination* conceitualmente representa o *Uniform Resource Identifier (URI)*, isto é o endereço completo do servidor destino. O campo *flag* armazena um valor verdade na forma de inteiros binários (zero ou um), zero torna a entrada disponível e um indisponível. O campo *priority* armazena inteiros que representam a prioridade da entrada. As modalidades de balanceamento de carga baseado em atributos da mensagem *SIP* utilizam do campo *attrs* para validar critérios válidos que condicionam o despacho da carga de trabalho. Cada entrada na tabela possui o campo *description* que armazena o nome do servidor.

Após o ajuste de prioridades o *Orchestrator* precisa formalizar o novo cenário por meio da edição das tabelas de despacho do processo *Kamailio*. O *Kamailio* traz para memória *RAM* todos os destinos disponíveis para o balanceamento similar a um algoritmo de roteamento de pacotes, que armazena em tempo de execução qual é caminho mais barato para ser considerado em um posterior repasse. Portanto, para atualizar a tabela *Orchestrator* faz uma chamada remota de procedimento ao processo *Kamailio* por meio de uma mensagem *HTTP* destinada a interface *loopback*, ou seja a mensagem é endereçada ao *host* 127.0.0.1.

A mensagem de formato *HTTP* segue a descrição textual para que o processo *Kamailio* recarregue para a memória os novos endereços de despacho bem como suas prioridades. De forma complementar foi implementada uma camada de transporte segura de mensagens *HTTP* por meio do protocolo *Transport Layer Security (TLS)*. Para elevar o nível de segurança do módulo de forma que intrusos não envie mensagens destinadas ao *Orchestrator* com o objetivo de dissuadir o comportamento adequado. Somente entidades que possuem certificado são capazes de solicitar ao processo *Kamailio* que recarregue sua

tabela de balanceamento de carga. Dado o fato do *Orquestrator* estar fora do domínio é necessário uma camada de segurança adicional.

O padrão de infraestrutura de chaves públicas utilizado para complementar a segurança no ajuste das tabelas de balanceamento de carga foi o *X.509* proposto pelas entidades *ISO* e *ITU* (MYERS et al., 1999). O *X.509* foi considerado no trabalho por possuir um formato de dados padrão, possuir campos básicos para verificação do certificado (LEE; LEE; SONG, 2007) e possibilidade de revocar o certificado a qualquer momento (HOUSLEY et al., 1998). Também pela diversidade de aplicações que utilizam o padrão principalmente para segurança de mensagens *web* e *e-mail* e *IPsec* (HOUSLEY et al., 1998).

Uma vez organizado no banco de dados os destinos de balanceamento e suas prioridades o *Orchestrator* faz uma chamada remota de procedimento ao processo *Kamailio*. Isto é, ao enviar a mensagem por meio do método *PUT* do protocolo *HTTP*; uma mensagem típica para redefinir o balanceamento de carga é como: *https://127.0.0.1:5061/http_rpc/dispatcher/dispatcher.reload?=arg=dispatcher.reload*. Uma vez recebida o processo *Kamailio* refaz as regras de balanceamento sem prejuízos ao serviço.

3.2.2 Módulo de *Compute*

O *Orchestrator* gerencia logicamente instâncias hospedadas nos recursos de *compute*, por meio do Módulo de Computação. O gerenciamento físico dos recursos de *compute* é feito no *OpenStack* na versão 13 *Mitaka*. Para maior praticidade ao alocar um novo recurso, foi utilizado o conceito de *snapshot* do *OpenStack*. O conceito se traduz na criação de uma imagem exata e momentânea de uma instância em execução, ao fazer isso, todas as características (tamanho de disco, memória, alocação de processadores) são mantidas, além disso, dados dos usuários. Nesse sentido foi construído uma instância com a funcionalidade básica de rede, sinalização multimídia.

Na mesma instância houve configuração básica do sistema operacional no que diz respeito a iniciar automaticamente o processo de conectividade com o ambiente de acesso. Outra característica da instância matriz é a criação e configuração para início automático do algoritmo *Statistics.py* feito em *Python*, que é um dos componentes do *Orchestrator* acoplado em cada instância. O algoritmo coleta e envia estatísticas de consumo de *CPU* e memória para o *Orchestrator*. Não utilizar o serviço de telemetria oferecido pela nuvem caracteriza a solução como flexível no sentido de estar desacoplada de tecnologia; também torna a solução independente do *pooling* e instalação de agentes *Simple Network Management Protocol (SNMP)*[2] necessários em ferramentas como *Zabbix* (ZABBIX, 2017) ou *Nagios* (NAGIOS, 2017). Tecnicamente foi constatado elevado *overhead* no serviço de virtualização *Nova* do *OpenStack* quando habilitado esses serviços (BELL et al., 2015).

[2] Protocolo para gerenciamento de elementos de rede (STALLINGS, 1998).

Para gerenciamento dos recursos de *compute* sob ótica de entrega e continuidade do serviço sob demanda, o *Orchestrator* possui o Módulo de Elasticidade. Esse é uma linha de execução em *thread* separado da linha de execução principal do *Orchestrator*. Na Figura 14 é possível identificar a associação da *thread ComputeMonitor* com o programa principal. Na execução da solução a *thread ComputeMonitor* (ou Módulo de Computação) estabelece um *socket* que recebe mensagens transportadas pelo protocolo *UDP* através da porta 7070. Todo conteúdo recebido nessa porta se refere a mensagens em formado *JSON* de consumo de *CPU*, memória e o *IP* das instâncias que operam no *OpenStack* e oferecem serviços de rede, nesse caso roteamento de mensagens *SIP*.

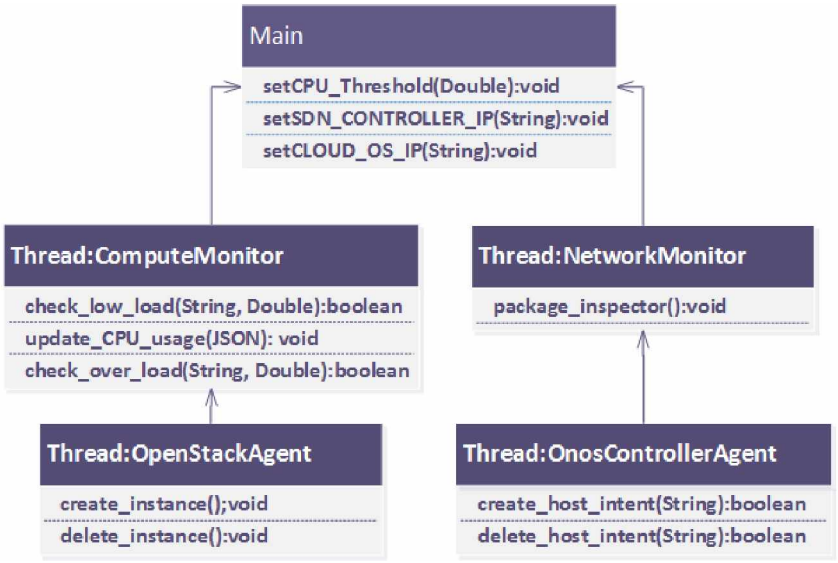

Figura 14 – Rotinas do *Orchestrator*.

Mensagens são recebidas em intervalos de tempo parametrizáveis. O algorítimo *Statistics.py* recebe parâmetros como entrada que especificam a frequência do envio das estatísticas para o *Orchestrator*. Para toda mensagem que o *Orchestrator* recebe ele extrai duas informações da mensagem *JSON*; o *IP* da instância e seu consumo de *CPU*. Em tempo de execução o *Orchestrator* cria uma cadeia de estatísticas numéricas de consumo de *CPU* para cada instância, que configura um histórico numérico. O conceito histórico é utilizado para avaliação de utilização da instância. Assim, toda mensagem recebida é armazenada no histórico e posteriormente é verificado o nível de saúde da instância no que diz respeito a processamento.

Além do armazenamento das mensagens de estatísticas de *hardware*, o *Orchestrator* submete a *VNF* remetente à verificação de nível de carga de trabalho. Segundo limiares definidos previamente na execução do *Orchestrator*, poderá eventualmente inicializar o Módulo de Elasticidade e utilizar seus métodos. Esses métodos conforme ilustrados na Figura 14 responsáveis por tal tarefa são o *check_low_load()* e *check_over_load()*. Ocorre que o consumo de *CPU* em um dado momento por uma máquina pode variar bruscamente de um instante para outro. Por motivos de processos em segundo plano em execução ou

rotinas de *kernel*. Para não esbarrar nesse cenário, a verificação de nível de carga de trabalho considera as três últimas estatísticas do histórico na forma de média aritmética para comparação com o *threshold* definido conforme na Figura 15.

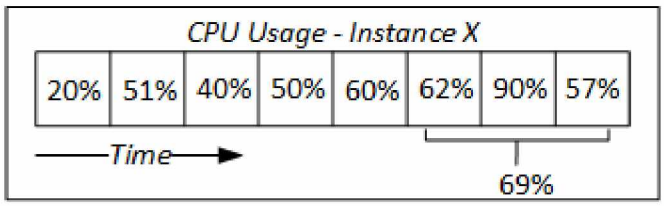

Figura 15 – *Orchestrator* – Histórico de consumo de *CPU*.

Os métodos descritos acima validam o mérito da alocação ou não de mais recursos. Quando uma *thread* para orquestração de mais recursos de *compute* é iniciada, uma trava (*lock*) é estabelecida para que o módulo de monitoramento não inicie outro serviço para criação de instância, uma vez que já existe uma iniciativa com esse fim em execução. O fato da *thread ComputeMonitor* executar somente com uma instância, não há possibilidade de ocorrer nenhum caso de condição de disputa pela trava. Quando a *thread OpenStackAgent* terminar de executar as rotinas para criação de uma instância e inicialização a trava poderá ser liberada, pois novo recurso passa a compor o rol de recursos de *compute* e é apto para receber carga de trabalho. Cabe ressaltar que esses parâmetros, frequência das mensagens e *threshold* são definidos na chamada da execução inicial do *Orchestrator*, assim são personalizáveis.

3.2.3 Módulo de Rede

Toda mensagem que transpassa o *Orchestrator* é analisada com objetivo de saber sua natureza. Para tanto foi definido a *thread NetworkMonitor* cuja tarefa principal é executar rotinas de inspeção de pacotes, ou seja essa funcionalidade materializa o *Módulo de DPI*. Isto é feito por meio da execução contínua do método *package_inspector()*. As rotinas do *Orchestrator* foram desenvolvidas inicialmente para serem capazes de inspecionar mensagens do protocolo *SIP*; essas mensagens são divididas em cabeçalho e corpo da mensagem. Os cabeçalhos são inspecionados em detalhes como o tipo de método, origem e destino da mensagem; já no corpo da mensagem tipicamente encontram-se informações para descrever a seção. A descrição de seção contém informações do *codec* e padrão de amostragem de áudio a ser utilizado na seção.

A rotina de inspeção de pacotes do *Orchestrator* consiste em:

1. se a mensagem recebida contém o cabeçalho *SIP* ela deverá ser considerada para uma análise mais detalhada – pode haver mensagens com outros cabeçalhos como *HTTP* ou *H.323* (THOM, 1996);

2. extrai-se da mensagem *SIP* o tipo de método, *BYE* e *INVITE* são considerados na execução do algoritmo;

3. também é extraído da mensagem *SIP* o endereço *IP* do chamador e do chamado;

4. se constatado que o método é *INVITE* conforme ilustrado na Figura 16 é iniciado uma rotina para criação de uma *Intent* por meio do controlador *SDN* – uma *Intent* provê conectividade *L2* entre *hosts* ou *devices* consumidores de serviços multimídia;

5. por outro lado, se constatado que o método é *BYE* é iniciado uma rotina para destruição da *Intent* entre o chamador e o chamado.

```
INVITE sip:user1@192.0.2.4 SIP/2.0
Via: SIP/2.0/UDP 192.0.2.10:5060;branch=z9hG4bKg48sh128
Via: SIP/2.0/UDP 192.0.2.20:5060;branch=z9hG4bK03djaoe1
To: sip:other-user@othernetwork.com
From: sip:another-user@anothernetwork.com;tag=938s0
Call-ID: 843817637684230998sdasdh09
CSeq: 101 INVITE
```

Figura 16 – Mensagem *SIP* contendo o método *INVITE*.

Fonte: (SPARKS, 2003).

Uma vez que o *Ochestrator* possui endereços *IP*, uma nova linha de execução é iniciada com objetivo de construir uma *Intent* entre o chamador e o chamado. Para não onerar a rotina contínua de inspeção de pacotes, uma *thread OnosControllerAgent* realiza o trabalho de interação com o controlador *SDN*, nesse caso o *ONOS*. Com a mesma dinâmica, quando a rotina de inspeção de pacotes percebe uma mensagem *SIP* com o método *BYE*, uma nova *thread* é criada porém com o objetivo de destruir a *Intent* criada. Pois conceitualmente métodos *BYE* sinalizam término de seções.

O *Orchestrator* classifica os endereços *IP* das entidades consumidoras de serviço multimídia em nível de controle e dados. O nível de controle são endereços criados para entidades se comunicarem entre si e para o controlador *SDN* atuar. Conforme a especificidade da arquitetura construída, a rede de controle é identificada por uma *VLAN*, ela é necessária para que o controlador *SDN* identifique e trabalhe adequadamente com os *flows* entre as entidades. Foi determinado para rede de controle o endereço: 192.168.254.x/24. Por outro lado, a rede de dados possibilita a comunicação das entidades com agentes externos ao ambiente de acesso, isso ocorre no cenário de consumo de serviço multimídia, uma vez que é necessário sinalização para estabelecer a seção. A rede de dados é determinada pelo endereço: 10.136.12.x/16. Portanto, o *Orchestrator* em tempo de execução faz mapeamento dos endereços *IP* de dados e controle das entidades do ambiente de acesso.

Capítulo 4

Avaliação Experimental

Esse capítulo descreve as ferramentas e os experimentos realizados para validação da hipótese. A Seção 4.1 descreve as tecnologias utilizadas para validar a solução. Na Seção 4.2 são discutidas as métricas utilizadas para validar o comportamento da solução. A Seção 4.3 descreve os experimentos e a Seção 4.4 discute e analisa os resultados dos experimentos realizados sobre os cenários propostos.

4.1 Ferramentas e Plataformas

Para realização do trabalho foram utilizadas as ferramentas: (1) controlador *SDN ONOS*; (2) *Kamailio SIP Server*; (3) *OpenStack Cloud Computing Software*, (4) *testbed Future Internet Brazilian Environment for Experimentation (FIBRE)* para experimentação. As rotinas do *Orchestrator* foram escritas em *Java 1.8 Programming Language* (disponibilizadas em repositório público[1]) e *Python*. Os experimentos ocorreram em ambiente real voltado para pesquisa experimental.

4.1.1 *Open Network Operating System (ONOS)*

A separação do plano de dados e controle é parte fundamental da arquitetura *SDN*. Portanto, orquestrar essa separação pesa sobre o controlador a responsabilidade por requisitos de desempenho, segurança, robustez e resiliência (SCOTT-HAYWARD, 2015). Nesse contexto o *Open Network Operating System (ONOS)* foi proposto em 2014 com destaque de controlador distribuído, escalável e tolerante a falha (BERDE et al., 2014). O presente trabalho foi construído e experimentado com o *ONOS* por ter boa documentação, *southbound interface* compatível com protocolo *OpenFlow 1.3* e pela possibilidade de ser implantado em diversos domínios de aplicação, tanto dentro de nuvens ou em redes *WAN* (SALMAN et al., 2016). O *ONOS* possui boas taxas de *throughput* comparado com seus pares (STANCU et al., 2015), por isso foi considerado como parte da solução

[1] https://github.com/romoreira/NetworkCompute-aware-Orchestrator-VoIP

uma vez que o objeto de pesquisa versa sobre redes multimídia, que carece de garantias da rede.

A verão Falcon[2] do *ONOS* traz consigo melhorias de desempenho no *core*; foram adicionadas camadas de segurança; a interface gráfica de operação foi aprimorada; a *northbound Interface* foi ampliada para ofertar métricas de tabela de *flows*, grupos de tabelas de *flows*; a *southbound Interface* foi aprimorada e recebeu suporte para gerenciamento de elementos via *SNMP*, e ajustes de compatibilidade com *IPv6*. Finalmente nessa versão o *framework* de *Intents* foi completamente construído, em versões anteriores ele era disponível em partes, e o consumo de tal funcionalidade como serviço só era possível via linha de comando no controlador. Agora o *framework* de *Intents* possibilita especificar um *bandwitdth* objetivo no caminho entre as entidades, isso é possível através de *Rest API*; essas capacidades pesou na escolha do *ONOS* como controlador da solução proposta.

O *ONOS* foi adotado como controlador da solução pelo fato de ter em sua implementação a possibilidade de inserir regras de comportamento mais abstratas, não somente baseado em *flows*. Como descrito em Kim e Feamster (2013) essa capacidade é descrita como intenção de alto nível ou *Intent*. Nesse sentido o *ONOS* se caracteriza como solução que oferece a implementação da funcionalidade de *Intents*. Segundo Bondkovskii et al. (2016) *Intents* são fáceis de serem gerenciadas pela interface gráfica ou por linha de comando, entretanto a edição de *Intents* não é prevista, nesse caso é necessário remover e instalar novamente com as adequações desejadas. O fato do *ONOS* implementar esse serviço o destaca entre seus pares. Também por ter trabalhos que compararam controladores pela ótica de desempenho onde o *ONOS* sobressai (ZHAO; IANNONE; RIGUIDEL, 2015; ANDRADE et al., 2016; YAMEI; QING; QI, 2016).

A conectividade fim-a-fim que o *framework* de *Intents* provê traz ganhos no contexto de redes multimídia sobretudo em cenários de *VoIP*. Parâmetros de rede influenciam diretamente na qualidade das chamadas de voz; atraso, *jitter*, perda de pacotes e ecos estão categorizados como grandes problemas da telefonia *IP* (KOVAC; HALAS, 2010). Esses aspectos são ofensores das métricas aferidoras de qualidade e experiência de chamadas de voz, em específico a métrica subjetiva *MOS*. Dessa forma, falhas ou indisponibilidade em *switches* que encaminha *flows* de uma sessão *RTP* causa interrupção e incompreensão momentânea da mensagem trocada. Até que o *ONOS* ateste falhas e determine um caminho alternativo para finalmente escrever novas regras nos *switches* da topologia. Por consequência ocorre degradação no aferidor de qualidade *MOS*, mas a sessão *RTP* permanece ativa; isso causa menos custo para o cliente (KUSHMAN; KANDULA; KATABI, 2007). A resiliência da sessão *RTP* evita que ocorra quedas de chamadas; como em Bu, Liu e Towsley (2006) quedas são ofensores da *MOS*.

[2] Versão 1.5.0 do controlador *SDN ONOS* disponível para comunidade (ONOS, 2016).

4.1.2 *Future Internet Brazilian Environment for Experimentation (FIBRE)*

A necessidade de ter uma plataforma para experimentos livre das redes em produção e com características similares as redes reais culminou a construção do *testbed OFELIA* (SUñé et al., 2014). O *testbed* foi desenvolvido no contexto do projeto *Seventh Framework Programme for Research and Technological Development – FP7*, um projeto financiado pela União Europeia executado na janela dos anos 2007-2013 (FP7, 2007). O *FIBRE* é uma expansão do modelo anteriormente construído e desde 2011 é operado pela Rede Nacional de Ensino e Pesquisa (RNP) no Brasil. O *testbed* é composto por treze ilhas geograficamente distribuídas no Brasil, as ilhas são controladas pelo *OFELIA Control Framework (OCF)* (SUñé et al., 2014) e *OFELIA Managment (OMF)* (RAKOTOARIVELO et al., 2010). Esses *frameworks* de controle propõem mais flexibilidade na condução de experimentos por prover orquestração de diversos recursos simultaneamente. Com o *testbed* é possível experimentar soluções *SDN*, soluções de acesso *WiFi*, *WiMax* e redes *3G/4G* (SALMITO et al., 2014).

O *testbed* possui tecnologia de *slice* de rede onde na mesma infraestrutura de hardware é possível conduzir diversos experimentos paralelos em redes *OpenFlow* por usuários distintos (SHERWOOD et al., 2010). Cada experimentador reserva recursos de computação e rede em alguma ilha do *testbed*, e a conectividade das ilhas e divisão das redes ocorre por meio do protocolo *IEEE 802.1Q* (que atualmente realiza o conceito de *VLAN*). Recursos do *FIBRE* são denominados *Aggregate Managers (AMs)* e podem ser incorporados via *browser* pelo usuário na interface de gerenciamento do *Network Operation Center (NOC)*.

Uma vez agregados pela interface do *NOC* é possível que o usuário determine o *datapath* do seu *slice*. O *datapath* são as ligações entre os *swtiches OpenFlow*, bem como a ligação dos *hosts* que irão compor a topologia dos experimentos. Uma vez que os recursos de rede foram reservados no *FlowVisor* (SHERWOOD et al., 2009) é possível iniciar os experimentos; no ambiente experimental os recursos são limitados, por isso há uma política para início, pausa e término de experimentos (desalocação dos recursos e disponibilização para outros experimentadores); todos os experimentos são criados com especificação de tempo que utilizará os recursos e detalhes descritivos. Finalmente, por meio de *VPN* é possível operar os *hosts* que estão no *testbed* por meio de acesso *SSH*. Por serem entidades reais, é possível instalar quaisquer serviços para serem experimentados.

4.2 Métricas de Avaliação

Foi considerado para avaliação da solução a métrica Chamadas Por Segundo (CPS) (*Calls Per Second (CPS)*) e Tempo de Resposta. Tempo de Resposta, no contexto sinalização *VoIP*, é o tempo que o serviço de roteamento gasta para encaminhar os pacotes de

controle entre os envolvidos nas chamadas até o estabelecimento da sessão *RTP*. Também há outros recursos relacionados ao tempo, como tempo de registro, autenticação e rede. A métrica tempo de resposta afeta diretamente a qualidade de experiência do usuário (HOßFELD; FIEDLER; ZINNER, 2011; COLLANGE et al., 2012; FIEDLER et al., 2011), uma vez que se espera um comportamento específico quando se tenta estabelecer uma chamada *VoIP*.

De forma semelhante, espera-se que um sistema *VoIP* sobre computação em nuvem seja resistente a variações na carga de trabalho. Mesmo com essas cargas elevadas o sistema deverá ser capaz de oferecer e completar o serviço sem prejuízos aos usuários. O principal prejuízo de um sistema *VoIP* congestionado é chamadas que não se completam, e isso causa má percepção para o usuário (JIA et al., 2015); também degradação da avaliação de *QoS* pelas métricas subjetivas e objetivas (KUSHMAN; KANDULA; KATABI, 2007; ZHANG et al., 2013). Numerosos *benchmarks* avaliam a robustez de sistema *VoIP* com a métrica *CPS*. Nesse sentido a métrica é explorada com o *hardware* no limite de forma a não degradar a qualidade do serviço sobretudo no contexto de *IaaS*.

4.3 Experimentos

Essa seção descreve os experimentos realizados com o intuito de avaliar o comportamento da solução proposta frente a critérios relevantes para as aplicações multimídia, especificamente *VoIP*. No mesmo sentido é avaliada a capacidade da solução em gerenciar recursos de computação de forma sensível ao contexto de carga de trabalho. Por fim é avaliada a capacidade da solução em exercer influência sobre elementos da rede em função da aplicação em execução com objetivo de aprimorar a qualidade de experiência. A condução desses experimentos permite cobrir completamente os dois universos de influência do *Orchestrator*.

4.3.1 Orientação a recursos de Computação

É experimentado a capacidade do *Orchestrator* em reagir sob demanda por meio da *Rest API* nos recursos computacionais baseado na carga de trabalho. Conforme a Figura 17, a aplicação de sinalização *VoIP* é executada sobre o modelo de serviço *IaaS* gerenciado pelo *OpenStack*. Dessa forma, é possível conceber a infraestrutura como *pool* de servidores de sinalização, *Kamailio SIP Router (KSR)*. Cada *KSR* possui capacidade limitada de sinalização de chamadas simultâneas por segundo. Por isso, é importante que o *Orchestrator* seja capaz de lidar com a sazonalidade da carga de trabalho, para sempre prover recursos suficientes para estabelecer as sessões nas aplicações multimídia. O *Orchestrator* é executado em *hardware Intel Core 2 Quad CPU Q9550 @ 2.83GHz* de processamento e 8 GB RAM.

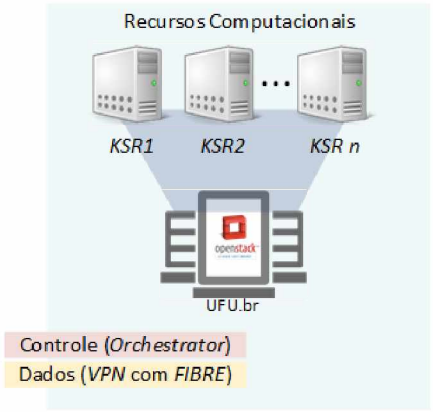

Figura 17 – Infraestrutura de Computação.

Além de sinalização de sessões multimídia, opera sobre a infraestrutura do *OpenStack* uma função de rede virtualizada. Tal função é útil dado a necessidade do transporte de carga útil em cenários de multi-domínios, similar ao que ocorre no modelo de serviço *roaming* prestado pelas operadoras de telefonia móvel. Em ambos é necessário que uma entidade faça interface entre os domínios, conceitualmente tem-se um *proxy RTP*. Toda instância possui característica e capacidade limitada de *hardware*, conforme padronização descrita na Tabela 3. No entanto, é possível personalizar e criar novos modelos de *flavor* alternativo ao padrão do *OpenStack*. Para o cenário de experimentos foi personalizado um *flavor m1.small* com 1 (um) *vCPU*, 20 (vinte) *GB* de armazenamento e 1024 *MB* de *RAM*.

Tabela 3 – Características padrão dos *Flavors*.

Flavor	*vCPUs*	Disco (em GB)	*RAM* em (MB)
m1.tiny	1	1	512
m1.small	1	20	2048
m1.medium	2	40	4096
m1.large	4	80	8192
m1.xlarge	8	160	16384

Fonte: (OPENSTACK, 2017b).

Instâncias com os sistemas operacionais *Debian 8 Jessie*, *CentOS* 7 e *Ubuntu Server* 16.04 foram definidas como matriz do serviço de sinalização de sessões multimídia, ou seja foi criada no *OpenStack* a *VNF* sobre esses três sistemas operacionais hospedeiros. Nessa instância foi automatizado o estabelecimento automático da *VPN*[3] com o *testbed FIBRE* para eventual transporte de carga útil. Também foi instalado e configurado serviço de roteamento de chamadas *Kamailio*. A configuração ocorre através da edição do arquivo de configuração *"kamailio.cfg"* contido no diretório */etc/kamailio/*. Por ser uma solução

[3] Através da ferramenta *OpenVPN* – solução de *VPN open source* (OPENVPN, 2017).

multimídia genérica o *Kamailio* exige carregamento de módulos específicos e ajustes de parâmetros para cada modo de operação.

Além disso, foi acoplado um *script Python* detalhado no Anexo A.2 na inicialização do sistema operacional. Assim, quando a instância é inicializada, imediatamente o *script* envia estatísticas de *hardware* coletadas pelo comando *"top"* (BRESNAHAN; BLUM, 2015). O *datagrama* que transporta a mensagem com estatísticas de *hardware* possui tamanho típico de $4,86$ Kilobytes transportados no canal de 1.000 Mbps. A mensagem representa menos de 0,004% da capacidade do canal portanto não ocorre prejuízos no tráfego. Ao fim desses ajustes, foi criado no *OpenStack* um *snapshot* da instância base. Com o *snapshot*, o *OpenStack* armazena uma cópia exata da instância e a armazena como uma imagem elegível para ser usada no lançamento de futuras instâncias. Foi implementado no *Orchestrator* a capacidade de buscar dentre a lista de imagens disponíveis no *OpenStack* a imagem base para o lançamento da futura instância.

Para verificar a necessidade de inicializar uma nova instância e não incorrer a ocasião de decisões errôneas do *Orchestrator* devido a rajadas de consumo de *CPU*, foi definido uma média de consumo. A consumo médio consiste nas três últimas amostras enviadas pelas instâncias. Além disso, toda mensagem recebida é submetida ao cálculo da média. Uma vez calculado é verificado a condição do consumo de *CPU* estar maior ou igual a 90%, se estiver uma *thread* para criação da instância será iniciada. Por outro lado, se o consumo estiver menor ou igual a 5%, a instância é destruída por procedimentos de uma *thread*; a exclusão ocorre sem prejuízos as sinalizações correntes.

Para validar o cenário descrito foi considerado uma aplicação multimídia *VoIP* sinalizada pelo protocolo *SIP*, para o transporte de carga útil o protocolo *RTP*. Para gerar cargas contínuas de mensagens *SIP* foi utilizado a ferramenta *SIPp benchmark* (HEWLETT-PACKARD, 2003); nessa ferramenta foi configurado cenários que são armazenados em arquivos *xml*. O cenário experimentado e transcrito em *xml* é ilustrado na Figura 18. Os experimentos de carga de trabalho são: (1) parte-se de uma carga baixa para uma carga elevada após 10 segundos; (2) parte-se de uma carga elevada para uma carga baixa; (3) tempo de resposta a um pedido de chamada *INVITE* em cenário de elevada carga de trabalho.

A Figura 18 ilustra o fluxo de troca de mensagens entre entidades para estabelecer uma sessão *RTP* entre o *UAC* e *UAS*. É importante destacar que as entidades estão geograficamente distintas. É possível notar na imagem que a primeira mensagem *INVITE* a transpassar o *Orchestrator* é por ele balanceada para algum *KSR* disponível na lista de despacho. Uma vez balanceada, o *Orchestrator* mantém o estado da sinalização – *stateful*, assim futuras trocas de mensagens não se perderão entre os *KSRs*. Após seguir o fluxo ilustrado, o protocolo *RTP* assume o controle; um trecho de áudio (8 segundos) exemplo armazenado em formato *"pcap"* é executado do *UAC* para o *UAS*. Na Seção 4.4 é discutido o comportamento do experimento de carga de trabalho.

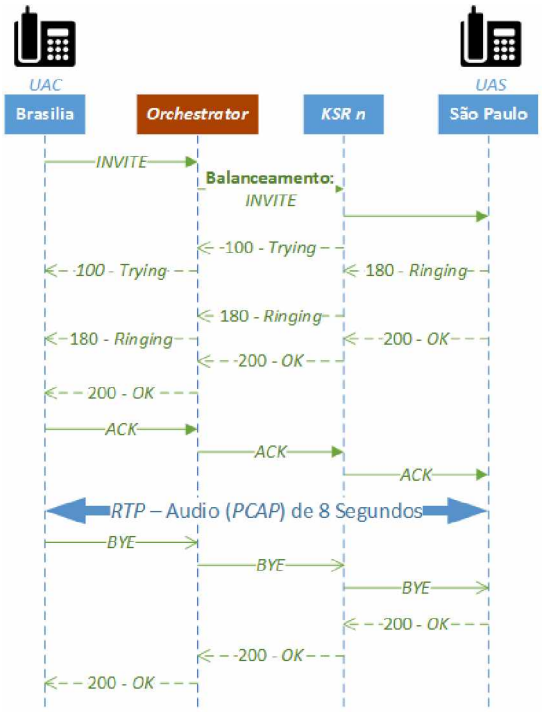

Figura 18 – Cenário Experimentado.

4.3.2 Orientação a recursos de Rede

É experimentado a capacidade do *Orchestrator* em reagir sob demanda através da *API Rest* do controlador *SDN ONOS*. A Figura 19 ilustra o cenário de experimentos multi-federado oferecido pelo *testbed FIBRE* que foi utilizado. Cada ilha possui recursos de virtualização que estão sobre uma rede *SDN*, isso permite experimentar inúmeros serviços de rede sobre instâncias reais. Os consumidores de serviços *VoIP* são experimentados em instâncias *Debian 8 Jessie* sobre um *hardware* virtualizado de $1,5GB$ de *RAM* e com $20GB$ de armazenamento. Foi compilado e instalado na instância *UAC* o *SIPp Benchmark*, após foi configurado módulo cliente no *SIPp* através do arquivo *xml* de configuração. O procedimento é análogo para a instância *UAS* que também é conectada na rede *SDN*, porém foi configurado o módulo servidor através do arquivo *xml*.

Cada ilha do cenário é geograficamente distribuída e são conectadas por um *backbone* de *switches SDN*. As ilhas que possuem recursos de virtualização são conectadas no *backbone FIBRE* por meio dos *switches* topo de *rack*, que também são *switches SDN*. É possível notar dois tipos de *links* entre os *switches*: *links* convencionais e com conectividade. A conectividade estabelecida por meio de *Intents* ocorre dependendo da aplicação multimídia e especificamente o conteúdo da mensagem que é trocada. Os *links* convencionais são os que conectam elementos porém sem garantia de *bandwidth* e conectividade em caso de falhas. Os recursos computacionais UFU.br onde são executados os serviços de *proxy RTP (VNF)* e sinalização *VoIP* com escalabilidade, estão conectados ao *backbone FIBRE* através de uma canal seguro *VPN* para operação dos serviços.

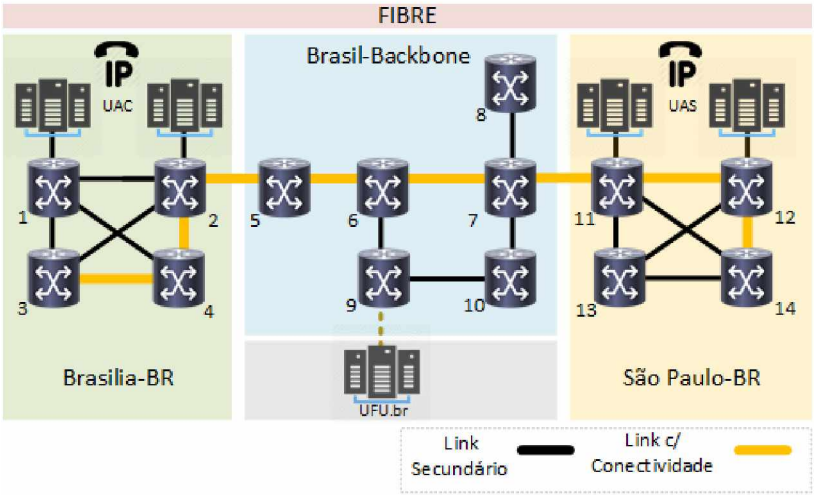

Figura 19 – Infraestrutura de Rede.

O ambiente de experimentos foi dividido em duas redes, rede de controle e dados. A rede de controle carimba todo pacote encaminhado pelas instâncias *UAC* ou *UAS* com a *VLAN* 100. Assim o *FlowVisor*[4] percebe a *VLAN* e encaminha os pacotes para que o controlador *ONOS* faça a devida tratativa. O controlador *ONOS* está no mesmo domínio das instâncias *UAS* e *UAC*. A rede de dados é a rede que conecta as instâncias reais no *testbed FIBRE* e provê acesso com a internet e também conectividade entre instâncias. O endereçamento das redes é: 192.168.254.0/24 *VLAN* 100 para a rede de controle e 10.12.0.0/16 para a rede de dados.

O *Orchestrator* está fora do domínio do *testbed FIBRE* e também dos servidores *SIP* em UFU.br, isso garante a ampliação da solução para cenários com diversos domínios de rede. Mesmo fora do domínio, o *gateway* do *testbed FIBRE* encaminha mensagens de sinalização para o *Orchestrator* através do canal seguro *VPN*. Por esse mesmo canal o *Orchestrator* interage com o controlador *ONOS* para manutenção da execução da solução. O *Orchestrator* busca do *ONOS* estatísticas dos *links*, *flows*, *devices*, novos *hosts* (entidades) que são conectadas ao *switches* e a lista de *Intents* instaladas. Quando uma mensagem de sinalização esbarra no *Orchestrator* uma *thread* faz a inspeção do conteúdo da mensagem e eventualmente em consequência do conteúdo interage com o *ONOS*. A Figura 20 ilustra a tratativa às mensagens assíncronas endereçadas ao *Orchestrator*.

Quando uma mensagem *INVITE* esbarra no *Orchestrator*, de forma assíncrona ele faz o balanceamento de carga para os *KSRs* disponíveis e encaminha ao *ONOS* parâmetros para criação da *Intent* e define na *Intent* o *bandwidth* máximo do canal. O *framework* de *Intents* provê conectividade *L2*, por isso, o *Orchestrator* faz um *hash* em sua lista de *hosts* com os *IPs* dos envolvidos na chamada e busca o *MAC* associado. O *Orchestrator* monta a mensagem *JSON* através da especificação do tipo da *Intent*, *MAC* dos envolvidos na chamada e o *bandwidth* do *path* virtual. Por outro lado, se o *Orchestrator* inspeciona

[4] Mecanismo de separação de redes *SDN* no *testbed FIBRE* (SHERWOOD et al., 2009))

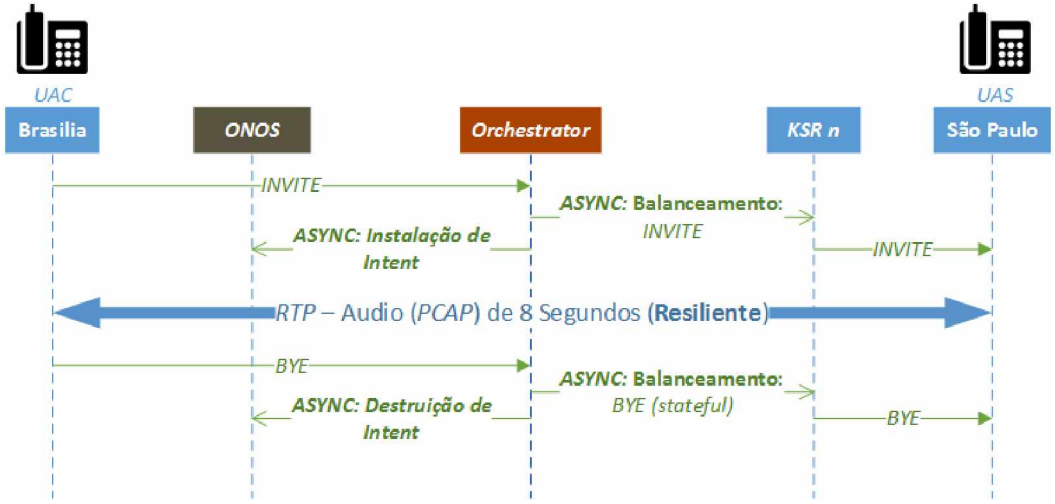

Figura 20 – Fluxo de mensagens do *Orchestrator*.

uma mensagem *BYE*, ele de forma assíncrona encaminha a mensagem ao *KSR* responsável (nesse caso não há balanceamento – comportamento *stateful*) e no mesmo instante encaminha ao *ONOS* parâmetros para destruição da *Intent*. A Seção 4.4 discute e avalia os resultados alcançados.

4.4 Avaliação dos Resultados

Os resultados alcançados nos experimentos descritos na Seção 4.3 demonstram que é possível construir uma solução *MANO compliant* capaz de gerenciar recursos de rede e *compute* simultaneamente. Também que mecanismos de inspeção de pacotes elevam o nível de assertividade de soluções destinadas a encurtar a distância de aplicações multimídia e a rede.

A subseção 4.4.1 discute a característica da solução em orquestrar recursos computacionais em cenários de oferta de serviços multimídia sobre infraestrutura *cloud computing*, também é discutido o comportamento da solução sob a ótica de três cenários. A subseção 4.4.2 discute as capacidades da solução em orquestrar recursos de rede para os mesmos serviços de multimídia. Por fim, a subseção 4.4.3 destaca as capacidades da solução contrastando-a com o estado da arte; também são discutidos aspectos teóricos da contribuição.

4.4.1 Orientação à Recursos de *compute*

É necessário que o *Orchestrator* tenha parâmetros para determinar quando uma nova instância esta apta para receber carga de trabalho. Por isso, foram conduzidos experimentos que mensuram o tempo que o *OpenStack* gasta para lançar uma nova instância, adicionado ao tempo que a instância está realmente disponível para receber carga de si-

nalização. A conclusão do lançamento de uma nova instância não assegura que ela está ociosa, é necessário instantes adicionais para que as rotinas do sistema operacional e aplicações básicas terminem de entrar em operação. Equanto o *Orchestrator* não percebe que a nova instância esta 100% ociosa ele não adiciona seu destino à lista de *KSR* elegíveis para receber carga antes de perfazer o tempo necessário. Esse tempo é calculado conforme: $launch\_time + so\_routines\_time = time\_to\_receice\_workload$. O tempo médio que a infraestrutura do *testbed* gasta para lançar uma nova instância é $66,73$ segundos. Os dados do experimentos estão descritos na Tabela 4.

Tabela 4 – Tempo de Lançar uma Instância.

Variável	N	Média	Desv. Padrão	95% Inter. Conf.
Tempo de Inicialização (segundos)	15	66,73s	$5,26s$	(63,82s; 69,64s)

Para validar o cenário 1, discutido na subseção 4.3.1 submeteu-se o *Orchestrator* à carga de trabalho que parte do instante $t = 0s$ com 80 *CPS* e após o instante $t = 10s$ altera-se taxa para 160 *CPS*. O gráfico da Figura 21 ilustra o comportamento do consumo de *CPU* dos recursos computacionais e a decisão do *Orchestrator* por lançar um novo *KSR* para lidar com o transbordo de carga de trabalho. O instante $t = 23s$ marca o momento que o *Orchestrator* inicia uma *thread* para executar procedimentos de criação de um novo *KSR*. A segunda linha do gráfico representa as estatísticas de *CPU* que o *KSR2* recente criado reporta ao *Orchestrator*, é notável que o consumo percentual está elevado, isso ocorre pois o rotinas e serviços básicos do S.O. ainda estão a entrar em operação.

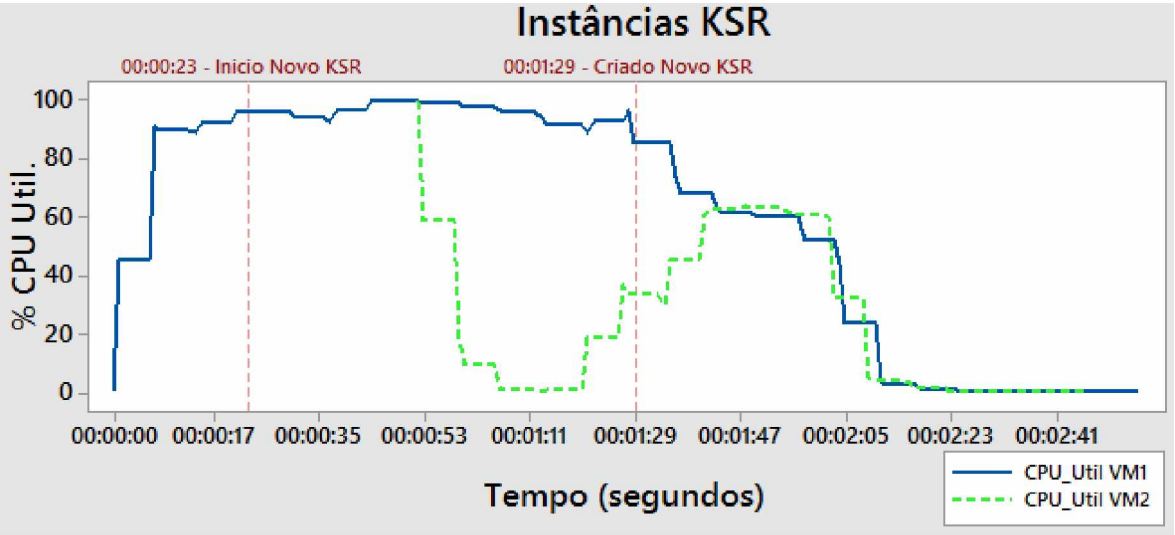

Figura 21 – Cenário 1 – De Carga baixa para Carga elevada.

No instante $t = 1{:}29m$ a instância efetivamente está apta a receber carga de sinalização, para esse fim o *Orchestrator* carrega o destino (*IP* de controle) do banco de dados para a lista de despacho em tempo de execução. Posteriormente o consumo de *CPU* da instância

KSR1 é degradado, e como se nota a instância *KSR2* divide a carga de sinalização. Note que o consumo de *CPU* (primeiro eixo y) da *KSR2* é levemente maior que da *KSR1*, a política de balanceamento de carga adotada pelo módulo *"dispacher"* do *Kamailio* é circular, *round-robin*. Além disso foi incorporado peso aos destinos elegíveis no âmbito da implementação. O *Orchestrator* implementa a regra que o servidor mais jovem recebe mais carga de trabalho. Esse comportamento se justifica pelo seguinte fato: se um *KSR* primário está com elevado consumo de *CPU*, mesmo com a entrada de um novo *KSR* secundário por meio do *Orchestrator*, a política circular continua a despachar carga de trabalho de forma igualitária, assim um ligeiro aumento de carga extrapolaria a capacidade do *KSR* primário e haveria má percepção do serviço pelo cliente.

No instante $t = 2{:}05m$, a carga de sinalização é abruptamente parada, no entanto as instâncias *KSRs* trabalham nas sinalizações remanescentes. Concluído, os *KSRs* registram percentuais mínimos de consumo de *CPU*, abaixo de 5%, assim o *Orchestrator* inicia uma *thread* com procedimentos para exclusão da instância. O *KSR* que registrar primeiro níveis mínimos de consumo de *CPU* avaliados pelo *Orchestrator*, é o primeiro a sair. Foi adotada a política de recursos computacionais mínimos para sinalização de chamadas, o *Orchestrator* sempre mantém 1 *KSR* em operação.

Para validar o cenário 2, é investigada a distribuição elevada de carga logo no início da execução do experimento. Tem-se uma alvo de 250 *CPS* que abruptamente é imposto sobre o *Orchestrator* conforme ilustrado no segundo eixo y da Figura 22. Como é possível notar, logo no início da execução o *CPS* alvo alcança valores superiores depois converge para aproximadamente 220 *CPS*. Isso ocorre porque na medida que as mensagens de sinalização (*INVITE*) alcançam o *KSR* ele deve manter o estado da sinalização (*stateful*) nas tabelas de roteamento da memória, para que mensagens de sinalização subsequente sejam encaminhadas corretamente. Em instantes iniciais da execução do experimento a memória está ociosa, após algumas mensagens de sinalização e memória é saturada. Isso compromete a capacidade de sinalização das chamadas e consequentemente o nível estabiliza ao máximo que o *hardware* consegue tratar.

Note que o consumo de *CPU* da instância *KSR1* após o experimento estabilizar é 100% por isso o *CPS* alvo não é alcançado. O *Orchestrator* identifica utilização elevada de *CPU* do *KSR1*, no instante $t = 36s$ e inicia uma *thread* para instanciar um novo *KSR*. Respeitado o tempo de conclusão do processo de *scaling up* do *KSR2* no instante $t = 1{:}38m$ ele efetivamente compõe a lista de despacho do *Orchestrator* e entra em operação. Posteriormente a aglutinação de servidores *KSRs* e o balanceamento proposto pelo *Orchestrator* garante alcançar o objetivo de 250 *CPS*. Com confiança de 95% sobre a média, foi alcançado 245 *CPS* sobretudo chamadas finalizadas sem retransmissão. Retransmissão de mensagens de sinalização degrada a utilização eficiente do canal de comunicação, causa estouros de *buffer* e aumento do tempo de resposta da requisição.

No instante $t = 2{:}27m$ o *benchmark* finaliza o objetivo de 250 CPS. É possível notar

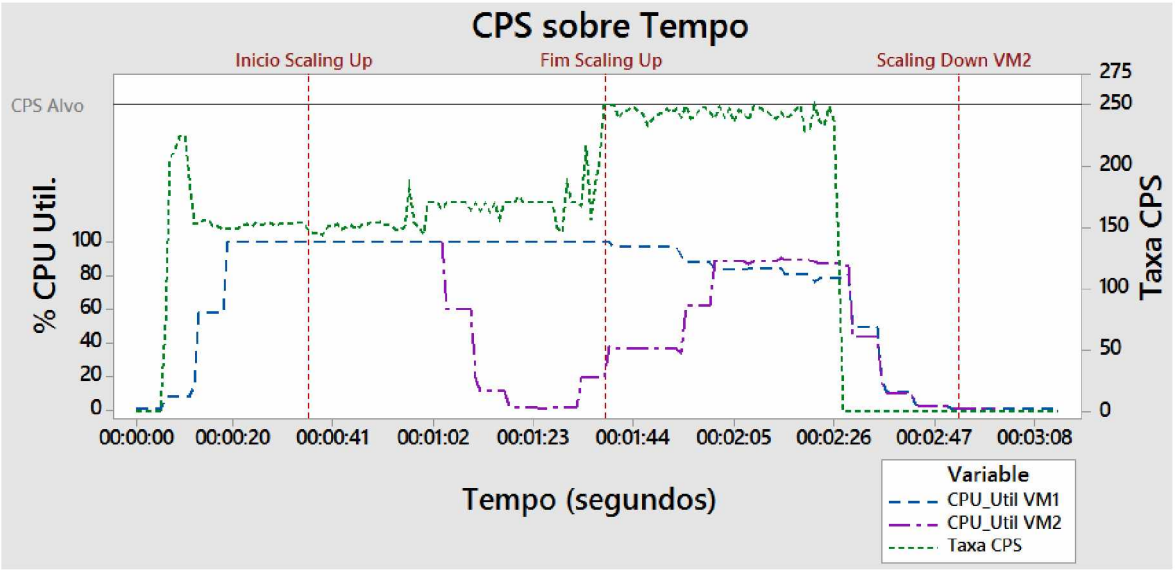

Figura 22 – Cenário 2 – De Carga elevada para Carga baixa.

que os níveis de consumo de *CPU* decrescem gradativamente na medida que terminam sinalizações remanescentes. Por fim no instante $t = 2{:}50m$ o *Orchestrator* identifica o *KSR2* com ociosidade e imediatamente inicia uma *thread* com procedimentos para excluir a instância do *pool*, bem como da lista de destinos elegíveis do seu banco de dados. Além de excluir do banco de dados é necessário atualizar da memória *RAM* a lista de destinos elegíveis. O *Orchestrator* implementa uma chamada remota de procedimento (*Remote Procedure Call (RPC)*) ao processo responsável pelo despacho de sinalização.

O cenário 3 experimenta tempos de resposta a requisições *INVITE*. Ambientes que exigem muitas retransmissões logram de tempos de resposta inapropriados sob a ótica de *QoS*. O gráfico da Figura 23 ilustra o comportamento dos serviços *VoIP* com o mesmo *timestamp* experimentado no cenário 1. É possível notar um comportamento instável do tempo de resposta do instante $t = 0$ até $t = 1{:}29m$; com 95% de confiança sobre a média é alcançado $188,7ms$ para toda requisição *INVITE*. Posteriormente o *Orchestrator* lida com o transbordo de carga de trabalho de sinalização.

O *Orchestrator* aumenta o *pool* de recursos computacionais, assim o cenário aprecia tempos de resposta expressivamente menores. Conforme Tabela 5 o tempo de resposta para toda requisição *INVITE* é $64,73ms$. Nesse caso o tempo de resposta se traduz como o intervalo do momento que o *UAC* tenta chamar o *UAS* até o estabelecimento da sessão *RTP*. Após a intervenção do *Orchestrator* não ocorre retransmissões e há redução de $65,6\%$ do tempo de resposta. O usuário perceberia uma serviço menos congestionado nesse cenário e os operadores perceberiam aumento percentual de chamadas completadas.

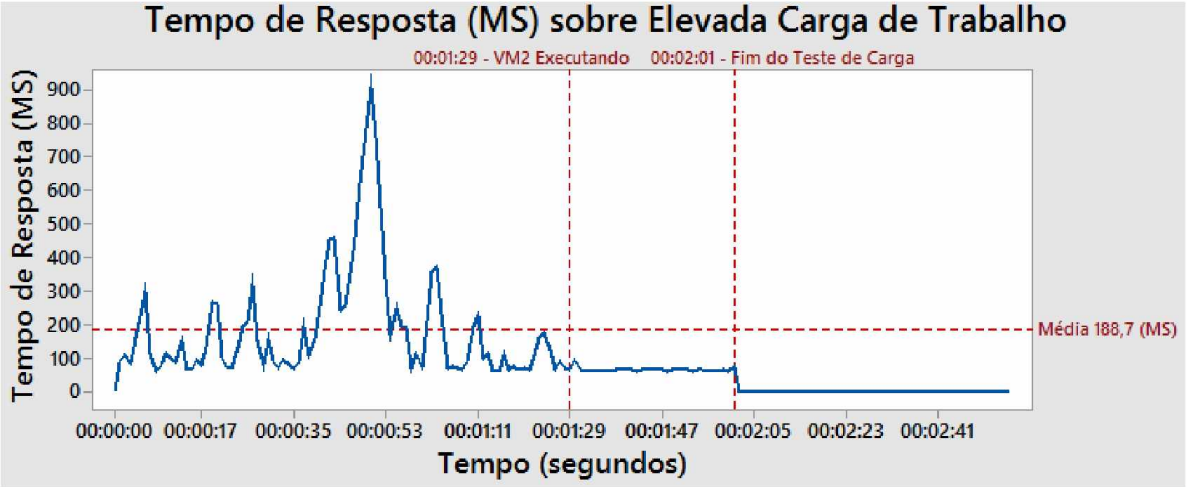

Figura 23 – Cenário 3 – Tempo de Resposta.

Tabela 5 – Tempo de Resposta (ms) com o *Orchestrator*.

Variável	N	Média	Desv. Padrão	95% Inter. Conf.
Tempo de Resposta (milissegundos)	33	$64,73ms$	$6,27ms$	$(62,50; 66,95)$

4.4.2 Orientação a recursos de rede

Para validar o experimento foi executado o cenário conforme descrito anteriormente e o comportamento do *Orchestrator* é ilustrado na Figura 24. Quando o *Orchestrator* inspeciona uma mensagem recebida e encontra o método *INVITE*, uma *Intent* do tipo *host-to-host* é instalada entre o *UAC* e *UAS*. O resultado de uma *Intent* é ilustrado na Figura 19 que rotula o *path* da *Intent* como link com conectividade. *Intents host-to-host* proveem conectividade entre os *hosts*, mesmo que um *switch* ou *link* intermediário falhe, o *ONOS* recalcula um novo *path* e o fluxo *RTP* é restabelecido.

Time	Source	Destination	Protocol	Length	Info
0.000000000	10.136.12.43	10.200.0.22	SIP/SDP	590	Request: INVITE sip:3030@10.136.12.45:5060 |
0.000185000	10.200.0.22	10.136.12.43	SIP	333	Status: 100 trying -- your call is important to us |
3.112090000	10.200.0.22	10.136.12.38	HTTP	261	GET /onos/v1/hosts HTTP/1.1
3.177269000	10.136.12.38	10.200.0.22	HTTP	492	HTTP/1.1 200 OK (application/json)
3.193811000	10.200.0.22	10.136.12.38	HTTP	263	GET /onos/v1/devices HTTP/1.1
3.276624000	10.136.12.38	10.200.0.22	HTTP	1192	HTTP/1.1 200 OK (application/json)
3.277400000	10.200.0.22	10.136.12.38	HTTP	263	GET /onos/v1/intents HTTP/1.1
3.338586000	10.136.12.38	10.200.0.22	HTTP	170	HTTP/1.1 200 OK (application/json)
3.339334000	10.200.0.22	10.136.12.38	HTTP	194	POST /onos/v1/intents HTTP/1.1 (application/json)
3.423476000	10.136.12.38	10.200.0.22	HTTP	234	HTTP/1.1 201 Created
12.602679000	10.200.0.22	10.136.12.45	SIP	449	Request: BYE sip:3030@10.136.12.45:5060 |
13.268155000	10.200.0.22	10.136.12.38	HTTP	261	GET /onos/v1/hosts HTTP/1.1
13.344993000	10.136.12.38	10.200.0.22	HTTP	492	HTTP/1.1 200 OK (application/json)
13.345512000	10.200.0.22	10.136.12.38	HTTP	263	GET /onos/v1/devices HTTP/1.1
13.417821000	10.136.12.38	10.200.0.22	HTTP	1192	HTTP/1.1 200 OK (application/json)
13.418335000	10.200.0.22	10.136.12.38	HTTP	263	GET /onos/v1/intents HTTP/1.1
13.492017000	10.136.12.38	10.200.0.22	HTTP	476	HTTP/1.1 200 OK (application/json)
13.492651000	10.200.0.22	10.136.12.38	HTTP	322	DELETE /onos/v1/intents/org.onosproject.cli/0x1 HTTP/1.1
13.580022000	10.136.12.38	10.200.0.22	HTTP	112	HTTP/1.1 204 No Content

Figura 24 – *Orchestrator* frente às mensagens *SIP*.

Quando o *Orchestrator* inspeciona uma mensagem com método *BYE*, imediatamente uma *thread* é iniciada para exclusão da *Intent*. Isso é necessário para que o *ONOS* não

seja sobrecarregado com as regras de *flows* desnecessárias. Um fato importante é que as chamadas podem ser encerradas sem serem devidamente sinalizadas pelo método *BYE*. Para isso o *Orchestrator* implementa a funcionalidade de expirar a *Intent* dado um período parametrizável de tempo. Uma *thread* decrementa o tempo da *Intent* ou da coleção (se houver), quando esse aproximar de zero, o *Orchestrator* sinaliza ao *ONOS* para exclusão da *Intent*.

Como se nota na Figura 24 os eventos do *Orchestrator* ocorrem conforme:

1. O *Orchestrator* recebe e inspeciona uma mensagem de sinalização *SIP* – constatado *INVITE*;
2. inspecionada a mensagem ele sinaliza o *UAC* que tentará estabelecer a chamada;
3. de forma assíncrona, ele busca do *ONOS* a lista de *hosts, devices* e *Intents*;
4. caso não haja nenhuma *Intent* entre as entidades, o *Orchestrator* solicita ao *ONOS* que crie;
5. chamada em curso;
6. o *Orchestrator* recebe e inspeciona uma mensagem de sinalização *SIP* – constatado *BYE*;
7. de forma assíncrona, ele busca do *ONOS* a lista de *hosts, devices* e *Intents*;
8. o *Orchestrator* solicita ao *ONOS* a destruição da *Intent* entre *hosts*.

4.4.3 Análise Geral

A Tabela 6 contrasta a solução construída com seus pares. Nela encontram-se esforços percebidos na literatura para aproximar aplicações multimídia da rede e dos recursos de computação.

Essa aproximação com a rede se traduz na capacidade da solução exercer influência (instalação de *Intents*) através das interfaces de controle aos elementos de rede, controlador *SDN ONOS*. A solução exerce influência mediante inspeção de pacotes (conteúdo das mensagens de sinalização) com objetivo de prover um canal de transporte de carga útil resistente a eventuais falhas dos elementos intermediários. Diferente das soluções discutidas no estado da arte, o *Orchestrator* faz isso com grande assertividade e de forma disruptiva, pois pauta suas decisões mediante a aplicação multimídia em execução, através da inspeção de pacotes. Também, a solução é versátil por ser capaz de operar em ambiente multi-domínios; isso a torna elegível de ser incorporada pelos operadores para redução de custos com quedas de chamadas e indisponibilidade, também a melhora da percepção do serviço prestado ao usuário. Em contraponto, as soluções do estado da arte, são restritas ao domínio, ou seja, tratam questões internas de *cloud*, fato ainda mais grave, operam

Tabela 6 – Comparativo das características das Soluções.

Trabalho	Provê Escalabilidade	Métricas de QoS	Visão Global	Provê Flexibilidade	Critério para tomada de decisão
(PODDAR; VISHNOI; MANN, 2015)	Não, a solução não provê escalabilidade.	Latência do controlador *SDN*.	Sim.	Não, trabalha apenas com controlador *SDN* específico e com *OpenStack Neutron*. Não realiza quaisquer influência em recursos baseados no contexto da aplicação.	*Threshold* do uso da memória, *CPU* e *bandwidth*.
(MUELLER; MAGEDANZ, 2012)	Não, a solução não provê escalabilidade.	*Bitrate* em *streaming* de media.	Não, a solução não conhece consumo de *CPU* ou memória das entidades ofertantes do serviço.	Parcial, baseado no contexto da aplicação a solução exerce influência nos recursos de rede, somente.	Condições do tráfego de rede, políticas previamente definidas, e coleta de métricas através do equipamento do usuário enviadas a entidade *GARC*.
(REGO et al., 2015)	Sim, a solução provê escalabilidade.	*Jitter* e taxa de perda de pacotes.	Sim, todas entidades da solução são monitoradas por uma entidade central.	Não, a solução é fortemente acoplada ao *OpenNebula* e não exerce nenhuma influência em recursos baseada no contexto da aplicação.	*Threshold* do uso de *CPU*.
(CARELLA et al., 2015)	Sim, a solução provê escalabilidade.	Taxa de Registro e Chamadas por segundo	Não, os componentes *IMS* executam em diferentes *VMs* e a solução não consegue ver isso.	Parcial, baseado na aplicação a solução consegue exercer influência nos recursos de *compute*, somente.	*Threshold* do uso de *CPU*.
(HAHN; GAJIC, 2015)	Sim, a solução provê escalabilidade.	Transações por segundo, latência e *bandwidth*.	Não, a solução apenas conhece a rede interna do *data center*.	Não, a solução não exerce nenhuma influência em quaisquer recursos baseado no contexto da aplicação.	Predição de tráfego com padrões definidos *a priori* e *threshold* do uso de *CPU*.
Orchestrator	Sim, a solução provê escalabilidade.	Chamadas por Segundo (CPS), tempo de resposta e % *CPU*.	Sim.	Sim, baseado no contexto da aplicação a solução exerce influência simultânea em recursos de rede e *compute* e é fracamente acoplada à tecnologia.	Melhor caminho para garantir conectividade entre entidades que consomem aplicações multimídia, contexto das aplicações multimídia e utilização de *CPU*.

dentro da nuvem. Pelo lado do acesso, tais soluções dificilmente são elegíveis de se tornar uma entidade de rede, para trabalhar em cooperação com por exemplo elementos de *core* numa rede *LTE*. A incorporação da solução é possível através de um acordo de serviço entre operadores, similar ao que ocorre com troca de tráfego.

A solução aproxima a aplicação dos recursos de *compute* ao prover escalabilidade sob demanda para lidar com cenários de elevada carga de trabalho. Aplicações multimídia abstraem as características do *hardware* sobre o qual operam, e conforme observado nos experimentos, o elevado consumo de *CPU* aumenta o tempo de resposta de sinalização; elevados tempos de resposta degradam a qualidade de experiência do usuário nas sessões multimídia. Conceitualmente há limites superiores para o tempo de resposta, seja de sinalização ou carga útil, uma vez extrapolado em caso de aplicações *VoIP*, ocorre in-

compreensão da fala, quedas e para cenários mais sofisticados aumento de processamento dos *proxies RTP*. Diferente das soluções encontradas no estado da arte, o conceito do *Orchestrator* lida adequadamente com essas questões de computação em paralelo, políticas de balanceamento, gerenciamento de recursos e características da rede sob restrições da aplicação.

Cenários multi-domínios evidenciam que a solução é compatível com o gerenciamento e orquestração previsto no *framework NFV*. Os *KSRs* implementam a função de rede *proxy RTP*, essa função é necessária para permitir a comunicação de voz entre dois equipamentos do usuário em domínios de rede diferentes ou dentro de redes *NAT*. A Tabela 7 descreve a compatibilidade da solução com o *framework NFV MANO*, o *framework* sumariza a não exaustiva lista de características desejáveis para a entidade *NFV MANO*. A Tabela é estruturada pelos blocos funcionais descritos no *framework* (ETSI, 2014).

Tabela 7 – Características da solução frente ao *framework NFV MANO*.

Solução	*NFV Orchestrator*	*VNF Manager*	*Virtualized Infraestrucure Manager*
Orchestrator	é capaz de gerenciar *VNF* (*proxy RTP*) quando aplicável, mantém uma lista de servidores em seu repositório e aplica políticas de carga a cada um deles;	lida com o gerenciamento do ciclo de vida das instâncias *VNF*, escalabilidade, terminação de instância, gerenciamento de integridade (despacho de carga de trabalho pautado na disponibilidade efetiva);	controla os recursos de computação (*storage, compute* e *network*) através da interface *Rest API* do *OpenStack*, o *Orchestrator* escolhe característica do *flavor* e o sistema operacional hospedeiro do *VNF*, define grupo de segurança e o *IP* público para oferta de serviço multi-domínio.

Capítulo 5

Conclusão

O objetivo do trabalho de prover uma solução *NFV compliant* para gerenciamento de recursos de computação e rede para aprimoramento de *QoS* em aplicações multimídia baseadas em *NFV* foi alcançado através da materialização do *Orchestrator*; uma abordagem que permite gerenciamento simultâneo de recursos sobretudo baseado no contexto de aplicação multimídia.

Inicialmente a literatura correlata foi revisada com objetivo de pontuar a contribuição da hipótese frente as lacunas do estado da arte. A revisão permitiu adquirir *know-how* em sinalização de aplicações multimídia, *cloud computing*, redes definidas por *software* e virtualização de funções de rede. Foram encontrados inúmeras propostas que visam aprimorar *QoS* em aplicações multimídia em diversos cenários, porém todas negligenciam o aspecto de gerenciamento conjunto e simultâneo, pautado de características de segurança, flexibilidade e extensível para interoperabilidade com outras tecnologias de acesso.

A construção da solução discutida em detalhes no Capítulo 3 foi precisamente pautada nos objetivos específicos previstos na Seção 1.2. Todos eles foram tratados e cada aspecto foi demonstrado no Capítulo 4. Lidar com esses micro objetivos exigiu grande esforço na curva de aprendizado dado a variedade tecnologias utilizadas; sobretudo a integração de inúmeros módulos, domínios de rede (ambiente de acesso e serviços de computação) e *scripts*. A conclusão deles permitiu atingir o objetivo maior por meio da implementação da prova de conceito em cenários de aplicações *VoIP*.

O objetivo dessa dissertação foi a construção de um orquestrador de recursos de rede e computação para aprimoramento de *QoS* em aplicações multimídia baseadas em *NFV*; também um esforço para diminuir a distância das aplicações com os recursos sobre os quais elas operam. Como demonstrado, o *Orchestrator* atua de forma assíncrona nos recursos de rede e computação baseado no contexto da aplicação e carga de trabalho. Desta forma, a prova de conceito se apresenta como candidata adequada para resolver os problemas postulados em prol do aprimoramento de *QoS* em aplicações multimídia.

Por isso é possível concluir que objetivos gerais e específicos foram alcançados conforme demonstrado e discutido anteriormente, conclui-se que a hipótese de construir uma

solução *NFV compliant* para gerenciamento eficiente e simultâneo de recursos para aprimoramento de *QoS* em aplicações multimídia é realizável. As próximas seções destacam as contribuições do trabalho desenvolvido e trabalhos futuros.

5.1 Principais Contribuições

O desenvolvimento do trabalho contribui com uma solução de design arquitetural baseado no *MANO* do conceito *NFV*. Isso posiciona a solução como uma entidade extensível para arquiteturas de internet do futuro; e, por meio dela, cumprir requisitos mais específicos de orquestração e aprimoramento de *QoS* para domínios diversos. O trabalho contribui com uma abordagem pouco intrusiva para coleta de estatísticas de consumo de *hardware* de *VNFs*, para a prova de conceito foi utilizado a função de sinalização *VoIP*. Entende-se por pouco intrusiva a característica de independência de protocolo como *SNMP*, *pooling* das ferramentas de monitoramento (*Zabbix* por exemplo) e telemetria de nuvem (*Ceilometer* para o caso do *OpenStack*).

O *Orchestrator* se posiciona como uma solução flexível em termos de tecnologia e domínio de atuação. Ou seja, no que diz respeito a domínio de atuação é possível conceber sua operação na rede de um operador somente, ou prover gerenciamento de recursos entre operadores; como garantia conectividade, priorização de tráfego, determinação de parâmetros de rede fim-a-fim e gerenciamento de serviços de computação. Além disso, o *Orchestrator* contribui com o conceito de independência de tecnologias de acesso, ou seja, é possível incorporar um *plugin* para que ele atue em elementos de redes móveis ou redes *Wi-Fi*. A prova de conceito, por exemplo, se baseia em redes *LAN*.

A condução do trabalho permitiu esbarrar no problema de balanceamento de carga para aplicações de tempo real, que além de possuir requisitos de desempenho diferenciados, exige comportamento *stateful*. Foi necessário customizar uma política de balanceamento típico *Round-robin*, para tanto adicionou-se uma camada de prioridades para esquivar de cenários de balanceamento de carga para uma *VNF* com alto grau de saturação. Com isso, o trabalho contribui com uma perspectiva de balanceamento que garante integridade da *VNF* mesmo em cenários de elevada carga de trabalho. Assim, houve redução e prevenção de colapso dos serviços mesmo em tais cenários, além de elevar a percepção e *QoS*.

Houve também a necessidade de construir uma *VNF* matriz para os serviços de sinalização *VoIP*. Após adequações necessárias a *VNF* passou a compor o repositório de serviços disponíveis gerenciado pelo *Orchestrator*. Além disso, o trabalho contribui com módulos assíncronos para gerenciamento de recursos o que torna a solução orientada a *compute*, rede e aplicação. Destaque para o módulo de inspeção de pacotes que promove o conceito como um bom candidato em termos de assertividade para diminuir a abstração das aplicações em execução, a rede utilizada para transporte, bem como o hardware hospedeiro.

5.2 Trabalhos Futuros

A construção do trabalho permitiu vislumbrar inúmeros aspectos passíveis de mais contribuições no que diz respeito a tópicos emergentes das redes de computadores. Podem ser considerados como segue:

1. Ampliar o escopo de aprimoramento de *QoS* para redes de internet do futuro;
2. Implementar a interface de políticas com a entidade *Policy and Charging Rules Function (PCRF) (ETSI, 2012b)* para abranger redes móveis;
3. Implementar uma interface com servidores de distribuição de conteúdo para aprimoramento de *QoS* em distribuição de vídeo;
4. Investigar impactos de performance inerentes a execução da solução;
5. Ampliar a funcionalidade da solução a fim de ofertar módulo de monitoramento gráfico;
6. Investigar impactos de performance inerentes ao tipo *codec* de mídia;
7. Investigar políticas de balanceamento de carga alternativas.

5.3 Submissões em Produção Bibliográfica

Submissão no *Global Communications Conference (GLOBECOM)* 2017 – *A Flexible Network and Compute-Aware Orchestrator to Enhance QoS in NFV-based Multimedia Services.*

Referências

3GPP. *3GPP Messaging using the IP Multimedia (IM) Core Network (CN) subsystem, Release 5*. 1999. Disponível em: <http://www.3gpp.org/specifications/releases/75-release-5>.

ABU-LEBDEH, M. et al. Cloudifying the 3gpp IP multimedia subsystem for 4g and beyond: A survey. **IEEE Communications Magazine**, v. 54, n. 1, p. 91–97, jan. 2016. ISSN 0163-6804.

AGRAWAL, D. et al. Database scalability, elasticity, and autonomy in the cloud. In: _____. **Database Systems for Advanced Applications: 16th International Conference, DASFAA 2011, Hong Kong, China, April 22-25, 2011, Proceedings, Part I**. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011. p. 2–15. ISBN 978-3-642-20149-3. Disponível em: <http://dx.doi.org/10.1007/978-3-642-20149-3_2>.

ALSHAKHSI, S. A. A.; HASBULLAH, H. Improving qos of vowlan via cross-layer-based adaptive approach. In: **2011 International Conference on Information Science and Applications**. [S.l.: s.n.], 2011. p. 1–8. ISSN 2162-9048.

AMAZON. **Elastic Compute Cloud (EC2) – Servidor e hospedagem na nuvem – AWS**. 2017. Disponível em: <https://aws.amazon.com/pt/ec2/>.

_____. **Elastic Compute Cloud (EC2) – Servidor e hospedagem na nuvem – AWS**. 2017. Disponível em: <https://aws.amazon.com/pt/ec2/>.

ANDRADE, L. et al. On the benchmarking mainstream open software-defined networking controllers. In: **Proceedings of the 9th Latin America Networking Conference**. New York, NY, USA: ACM, 2016. (LANC '16), p. 9–12. ISBN 978-1-4503-4591-0. Disponível em: <http://doi.acm.org/10.1145/2998373.2998447>.

ARMBRUST, M. et al. A view of cloud computing. **Commun. ACM**, ACM, New York, NY, USA, v. 53, n. 4, p. 50–58, abr. 2010. ISSN 0001-0782. Disponível em: <http://doi.acm.org/10.1145/1721654.1721672>.

BAKSHI, K. Considerations for software defined networking (sdn): Approaches and use cases. In: **2013 IEEE Aerospace Conference**. [S.l.: s.n.], 2013. p. 1–9. ISSN 1095-323X.

BASTA, A. et al. Applying nfv and sdn to lte mobile core gateways, the functions placement problem. In: **Proceedings of the 4th Workshop on All Things Cellular: Operations, Applications, & Challenges**. New York, NY, USA: ACM, 2014. (AllThingsCellular '14), p. 33–38. ISBN 978-1-4503-2990-3. Disponível em: <http://doi.acm.org/10.1145/2627585.2627592>.

BEACON. ***Beacon SDN Controller***. 2015. <https://openflow.stanford.edu/display/Beacon/Home>.

BELL, C. G.; NEWELL, A. **Computer structures: Readings and examples**. [S.l.]: McGraw-Hill New York, 1971. v. 2.

BELL, T. et al. Scaling the cern openstack cloud. In: IOP PUBLISHING. **Journal of Physics: Conference Series**. [S.l.], 2015. v. 664, n. 2, p. 022003.

BERDE, P. et al. Onos: Towards an open, distributed sdn os. In: **Proceedings of the Third Workshop on Hot Topics in Software Defined Networking**. New York, NY, USA: ACM, 2014. (HotSDN '14), p. 1–6. ISBN 978-1-4503-2989-7. Disponível em: <http://doi.acm.org/10.1145/2620728.2620744>.

BERGSTRA, J. A.; MIDDELBURG, C. A. **ITU-T Recommendation G.107 : The E-Model, a computational model for use in transmission planning**. [S.l.], 2003.

BERNAL, P. S. M. Voz sobre protocolo ip: A nova realidade da telefonia. **São Paulo: Érica**, p. 271–350, 2007.

BIST, M.; WARIYA, M.; AGARWAL, A. Comparing delta, open stack and xen cloud platforms: A survey on open source iaas. In: **2013 3rd IEEE International Advance Computing Conference (IACC)**. [S.l.: s.n.], 2013. p. 96–100.

BOGINENI, K. et al. Lte part i: core network. **IEEE Communications Magazine**, IEEE, v. 47, n. 2, p. 40–43, 2009.

BOLLA, R. et al. Energy efficiency in the future internet: A survey of existing approaches and trends in energy-aware fixed network infrastructures. **IEEE Communications Surveys Tutorials**, v. 13, n. 2, p. 223–244, Second 2011. ISSN 1553-877X.

BONDKOVSKII, A. et al. Qualitative comparison of open-source sdn controllers. In: **NOMS 2016 - 2016 IEEE/IFIP Network Operations and Management Symposium**. [S.l.: s.n.], 2016. p. 889–894.

BONETTO, E. et al. Optical technologies can improve the energy efficiency of networks. In: **2009 35th European Conference on Optical Communication**. [S.l.: s.n.], 2009. p. 1–4. ISSN 1550-381X.

BOURAS, C.; KOLLIA, A.; PAPAZOIS, A. Sdn nfv in 5g: Advancements and challenges. In: **2017 20th Conference on Innovations in Clouds, Internet and Networks (ICIN)**. [S.l.: s.n.], 2017. p. 107–111.

BRESNAHAN, C.; BLUM, R. **LPIC-1 Linux Professional Institute Certification Study Guide: Exam 101-400 and Exam 102-400**. [S.l.]: John Wiley & Sons, 2015.

BU, T.; LIU, Y.; TOWSLEY, D. F. On the tcp-friendliness of voip traffic. In: CITESEER. **INFOCOM**. [S.l.], 2006.

CAMARILLO, G.; GARCIA-MARTIN, M.-A. **The 3G IP multimedia subsystem (IMS): merging the Internet and the cellular worlds**. [S.l.]: John Wiley & Sons, 2007.

CARELLA, G. et al. Cloudified IP Multimedia Subsystem (IMS) for Network Function Virtualization (NFV)-based architectures. In: **Computers and Communication (ISCC), 2014 IEEE Symposium on**. [S.l.: s.n.], 2014. Workshops, p. 1–6.

_____. Quality audit and resource brokering for network functions virtualization (nfv) orchestration in hybrid clouds. In: **2015 IEEE Global Communications Conference (GLOBECOM)**. [S.l.: s.n.], 2015. p. 1–6.

CARVALHO, L. S. G. a. D.; MOTA, E. D. S. Survey on application-layer mechanisms for speech quality adaptation in voip. **ACM Comput. Surv.**, ACM, New York, NY, USA, v. 45, n. 3, p. 36:1–36:31, jul. 2013. ISSN 0360-0300. Disponível em: <http://doi.acm.org/10.1145/2480741.2480753>.

CASSANDRA. ***The Apache Cassandra Project***. 2015. <http://cassandra.apache.org/>.

CHAMORRO, V. G.; CASTILLO, C. N.; LOPEZ-PIRES, F. An Elastic VoIP Solution Based on OpenStack. In: **2016 International Conference on Information Systems Engineering (ICISE)**. [S.l.: s.n.], 2016. p. 43–47.

CHUNG, K.-Y. Recent trends on convergence and ubiquitous computing. **Personal Ubiquitous Comput.**, Springer-Verlag, London, UK, UK, v. 18, n. 6, p. 1291–1293, ago. 2014. ISSN 1617-4909. Disponível em: <http://dx.doi.org/10.1007/s00779-013-0743-2>.

CISCO. **Cisco VNI Mobile Forecast (2015 – 2020) – Cisco - Cisco**. 2016. Disponível em: <http://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/mobile-white-paper-c11-520862.html>.

_____. **Voice Over IP - Per Call Bandwidth Consumption - Cisco**. 2016. Disponível em: <http://www.cisco.com/c/en/us/support/docs/voice/voice-quality/7934-bwidth-consume.html>.

CLEARWATER. ***The Project Clearwater***. 2015. <http://www.projectclearwater.org/>.

COLLANGE, D. et al. User impatience and network performance. In: **Proceedings of the 8th Euro-NF Conference on Next Generation Internet NGI 2012**. [S.l.: s.n.], 2012. p. 141–148.

COSTA, P. Bridging the gap between applications and networks in data centers. **SIGOPS Oper. Syst. Rev.**, ACM, New York, NY, USA, v. 47, n. 1, p. 3–8, jan. 2013. ISSN 0163-5980.

COSTA-REQUENA, J. et al. Sdn and nfv integration in generalized mobile network architecture. In: **2015 European Conference on Networks and Communications (EuCNC)**. [S.l.: s.n.], 2015. p. 154–158.

COUTINHO, E. F. et al. Elasticity in cloud computing: a survey. **annals of telecommunications - annales des télécommunications**, v. 70, n. 7, p. 289–309, 2015. ISSN 1958-9395. Disponível em: <http://dx.doi.org/10.1007/s12243-014-0450-7>.

DAHLMAN, E. et al. **3G evolution: HSPA and LTE for mobile broadband**. [S.l.]: Academic press, 2010.

DEERING, S. Icmp router discovery messages. 1991.

DERAKHSHAN, F. et al. Enabling cloud connectivity using sdn and nfv technologies. In: _____. **Mobile Networks and Management: 5th International Conference, MONAMI 2013, Cork, Ireland, September 23-25, 2013, Revised Selected Papers**. Cham: Springer International Publishing, 2013. p. 245–258. ISBN 978-3-319-04277-0.

DIXIT, A. et al. Towards an elastic distributed sdn controller. In: ACM. **ACM SIGCOMM Computer Communication Review**. [S.l.], 2013. v. 43, n. 4, p. 7–12.

DODIG-CRNKOVIC, G. Scientific methods in computer science. In: **Proceedings of the Conference for the Promotion of Research in IT at New Universities and at University Colleges in Sweden, Skövde, Suecia**. [S.l.: s.n.], 2002. p. 126–130.

DORIA, A. et al. **Forwarding and control element separation (ForCES) protocol specification**. [S.l.], 2010.

DUDOUET, F.; EDMONDS, A.; ERNE, M. Reliable cloud-applications: An implementation through service orchestration. In: **Proceedings of the 1st International Workshop on Automated Incident Management in Cloud**. New York, NY, USA: ACM, 2015. (AIMC '15), p. 1–6. ISBN 978-1-4503-3476-1. Disponível em: <http://doi.acm.org/10.1145/2747470.2747471>.

ETSI. **3GPP TS 23.228 V2.0.0 (2001-03)**. 2001. Disponível em: <http://www.3gpp.org/ftp/tsg_sa/tsg_sa/TSGS_11/Docs/PDF/SP-010121.pdf>.

_____. **3GPP TS 32 260 V6.5.0 (2006-03)**. 2006. Disponível em: <http://www.qtc.jp/3GPP/Specs/32260-650.pdf>.

_____. **3GPP TS 129.212 V7.4.0 (2008-04)**. 2008. Disponível em: <http://www.etsi.org/deliver/etsi_ts/129200_129299/129212/07.04.00_60/ts_129212v070400p.pdf>.

_____. **3GPP TS 32.260 V9.2.0 (2009-12)**. 2009. Disponível em: <www.3gpp.org/ftp/tsg_sa/WG5_TM/TSGS5_68/_specs_for.../32260-920.doc>.

_____. *Network functions virtualization – an introduction, benefits, enablers, challenges e call for action*. Whitepaper, 2012.

_____. **TS 123 203 - V10.8.0 - Digital cellular telecommunications system (Phase 2+); Universal Mobile Telecommunications System (UMTS); LTE; Policy and charging control architecture (3GPP TS 23.203 version 10.8.0 Release 10) - ts_123203v100800p.pdf**. 2012. Disponível em: <http://www.etsi.org/deliver/etsi_ts/123200_123299/123203/10.08.00_60/ts_123203v100800p.pdf>.

_____. **3GPP TS 129 214 V11.7.0 (2013-01)**. 2013. Disponível em: <http://www.etsi.org/deliver/etsi_ts/129200_129299/129214/11.07.00_60/ts_129214v110700p.pdf>.

_____. **3GPP TS 129 229 V11.7.0 (2013-01)**. 2013. Disponível em: <http://www.etsi.org/deliver/etsi_ts/129200_129299/129229/10.05.00_60/ts_129229v100500p.pdf>.

_____. **GS NFV-MAN 001 - V1.1.1 - Network Functions Virtualisation (NFV); Management and Orchestration - gs_NFV-MAN001v010101p.pdf**. 2014. Disponível em: <http://www.etsi.org/deliver/etsi_gs/NFV-MAN/001_099/001/01.01.01_60/gs_NFV-MAN001v010101p.pdf>.

_____. **3GPP TS 123 002 V12.7.0 (2015-07)**. 2015. Disponível em: <http://www.etsi.org/deliver/etsi_ts/123000_123099/123002/12.07.00_60/ts_123002v120700p.pdf>.

_____. **NFV_White_Paper 3**. 2015.

ETSIGS. ***Network functions virtualization (NFV); architectural framework v1.1.1***. Tech. Rep., 2013. Disponível em: <http://www.etsi.org/deliver/etsigs/NFV/001099/002/01.01.0160/gsNFV002v010101p.pdf>.

EVENTSTUDIO. **IMS Registration for an Unauthenticated User**. 2007. Disponível em: <http://www.eventhelix.com/ims/registration/ims_registration.pdf>.

FARHADY, H.; LEE, H.; NAKAO, A. Software-defined networking: A survey. **Computer Networks**, v. 81, p. 79 – 95, 2015. ISSN 1389-1286. Disponível em: <http://www.sciencedirect.com/science/article/pii/S1389128615000614>.

FAROOQI, A. H.; MUNIR, A. Intrusion Detection System for IP Multimedia Subsystem using K-Nearest Neighbor classifier. In: **Multitopic Conference, 2008. INMIC 2008. IEEE International**. [S.l.: s.n.], 2008. p. 423–428.

FIEDLER, M. et al. Time is perception is money – web response times in mobile networks with application to quality of experience. In: _____. **Performance Evaluation of Computer and Communication Systems. Milestones and Future Challenges: IFIP WG 6.3/7.3 International Workshop, PERFORM 2010, in Honor of Günter Haring on the Occasion of His Emeritus Celebration, Vienna, Austria, October 14-16, 2010, Revised Selected Papers**. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011. p. 179–190. ISBN 978-3-642-25575-5. Disponível em: <http://dx.doi.org/10.1007/978-3-642-25575-5_15>.

FIEDLER, M.; HOSSFELD, T.; TRAN-GIA, P. A generic quantitative relationship between quality of experience and quality of service. **Network, IEEE**, v. 24, n. 2, p. 36–41, mar. 2010. ISSN 0890-8044.

FLOODLIGHT. ***Floodlight Open SDN Controller***. 2015. <http://www.projectfloodlight.org/floodlight/>.

FOSTER, I. et al. Cloud computing and grid computing 360-degree compared. In: **2008 Grid Computing Environments Workshop**. [S.l.: s.n.], 2008. p. 1–10. ISSN 2152-1085.

FOUNDATION, O. N. Software-defined networking: The new norm for networks. **ONF White Paper**, v. 2, p. 2–6, 2012.

FP7. **Understanding - FP7 - Research - Europa**. 2007. Disponível em: <https://ec.europa.eu/research/fp7/index_en.cfm?pg=understanding>.

GALANTE, G.; BONA, L. C. E. d. A survey on cloud computing elasticity. In: **2012 IEEE Fifth International Conference on Utility and Cloud Computing**. [S.l.: s.n.], 2012. p. 263–270.

GKOUNIS, D. et al. Cases for including a reference monitor to sdn. In: **Proceedings of the 2016 Conference on ACM SIGCOMM 2016 Conference**. New York, NY, USA: ACM, 2016. (SIGCOMM '16), p. 599–600. ISBN 978-1-4503-4193-6. Disponível em: <http://doi.acm.org/10.1145/2934872.2959066>.

GLITHO, R. Cloudifying the 3gpp IP Multimedia Subsystem: Why and How? In: **New Technologies, Mobility and Security (NTMS), 2014 6th International Conference on**. [S.l.: s.n.], 2014. p. 1–5.

GOOGLE. **Google Cloud Computing, Serviços de Hospedagem e APIs | Google Cloud Platform**. 2017. Disponível em: <https://cloud.google.com/>.

HAHN, W.; GAJIC, B. Gw elasticity in data centers: Options to adapt to changing traffic profiles in control and user plane. In: **2015 18th International Conference on Intelligence in Next Generation Networks**. [S.l.: s.n.], 2015. p. 16–22.

HALLINGBY, H. K. et al. Convergence in action: A case study of the norwegian internet. **Telematics and Informatics**, v. 33, n. 2, p. 641 – 649, 2016. ISSN 0736-5853. Disponível em: <http://www.sciencedirect.com/science/article/pii/S0736585315001100>.

HAPROXY. **HAProxy - The Reliable, High Performance TCP/HTTP Load Balancer**. 2017. Disponível em: <http://www.haproxy.org/>.

HASAN, M. Z.; AL-RIZZO, H.; AL-TURJMAN, F. A survey on multipath routing protocols for qos assurances in real-time wireless multimedia sensor networks. **IEEE Communications Surveys Tutorials**, PP, n. 99, p. 1–1, 2017. ISSN 1553-877X.

HASHEM, I. A. T. et al. The rise of "big data" on cloud computing: Review and open research issues. **Information Systems**, v. 47, p. 98 – 115, 2015. ISSN 0306-4379. Disponível em: <http://www.sciencedirect.com/science/article/pii/S0306437914001288>.

HAWILO, H. et al. Nfv: state of the art, challenges, and implementation in next generation mobile networks (vepc). **IEEE Network**, v. 28, n. 6, p. 18–26, Nov 2014. ISSN 0890-8044.

HERNANDEZ-VALENCIA, E.; IZZO, S.; POLONSKY, B. How will nfv/sdn transform service provider opex? **IEEE Network**, v. 29, n. 3, p. 60–67, May 2015. ISSN 0890-8044.

HEWLETT-PACKARD, H. **Sipp tool by hp invent. SIPp Open Source SIP UA Simulator tool**. 2003.

HOßFELD, T.; FIEDLER, M.; ZINNER, T. The qoe provisioning-delivery-hysteresis and its importance for service provisioning in the future internet. In: **2011 7th EURO-NGI Conference on Next Generation Internet Networks**. [S.l.: s.n.], 2011. p. 1–8.

HOLMA, H. et al. Voip over hspa with 3gpp release 7. In: **2006 IEEE 17th International Symposium on Personal, Indoor and Mobile Radio Communications**. [S.l.: s.n.], 2006. p. 1–5. ISSN 2166-9570.

HOUSLEY, R. et al. **Internet X. 509 public key infrastructure certificate and CRL profile**. [S.l.], 1998.

HU, F.; HAO, Q.; BAO, K. A survey on software-defined network and openflow: From concept to implementation. **IEEE Communications Surveys Tutorials**, v. 16, n. 4, p. 2181–2206, Fourthquarter 2014. ISSN 1553-877X.

HU, Z. et al. A comprehensive security architecture for SDN. In: **Intelligence in Next Generation Networks (ICIN), 2015 18th International Conference on**. [S.l.: s.n.], 2015. p. 30–37.

HUEBSCHER, M. C.; MCCANN, J. A. A survey of autonomic computing—degrees, models, and applications. **ACM Comput. Surv.**, ACM, New York, NY, USA, v. 40, n. 3, p. 7:1–7:28, ago. 2008. ISSN 0360-0300. Disponível em: <http://doi.acm.org/10.1145/1380584.1380585>.

HUONG-TRUONG, T. et al. QoE-aware resource provisioning and adaptation in IMS-based IPTV using OpenFlow. In: **Local Metropolitan Area Networks (LANMAN), 2013 19th IEEE Workshop on**. [S.l.: s.n.], 2013. p. 1–3.

IETF. **Session Initiation Protocol (SIP) Parameters**. 2002. Disponível em: <https://www.ietf.org/assignments/sip-parameters/sip-parameters.xml>.

_____. **Diameter Session Initiation Protocol (SIP) Application**. 2006. Disponível em: <https://tools.ietf.org/html/rfc4740>.

ITU-T. **P.861 : Objective quality measurement of telephone-band (300-3400 Hz) speech codecs**. 1995. Disponível em: <https://www.itu.int/rec/T-REC-P.861/en>.

_____. **G.114 : One-way transmission time**. 2003. Disponível em: <https://www.itu.int/rec/T-REC-G.114-200305-I/en>.

_____. **P.800.1 : Mean Opinion Score (MOS) terminology**. 2003. Disponível em: <https://www.itu.int/rec/T-REC-P.800.1-200303-S/en>.

JADEJA, Y.; MODI, K. Cloud computing - concepts, architecture and challenges. In: **Computing, Electronics and Electrical Technologies (ICCEET), 2012 International Conference on**. [S.l.: s.n.], 2012. p. 877–880.

JAIN, R.; PAUL, S. Network virtualization and software defined networking for cloud computing: a survey. **Communications Magazine, IEEE**, v. 51, n. 11, p. 24–31, November 2013. ISSN 0163-6804.

JAMMEH, E. et al. Quality of experience (qoe) driven adaptation scheme for voice/video over ip. **Telecommunication Systems**, v. 49, n. 1, p. 99–111, 2012. ISSN 1572-9451. Disponível em: <http://dx.doi.org/10.1007/s11235-010-9356-5>.

JIA, Y. J. et al. Performance characterization and call reliability diagnosis support for voice over lte. In: **Proceedings of the 21st Annual International Conference on Mobile Computing and Networking**. New York, NY, USA: ACM, 2015. (MobiCom '15), p. 452–463. ISBN 978-1-4503-3619-2. Disponível em: <http://doi.acm.org/10.1145/2789168.2790095>.

JOSEPH, J. P. Pstn services migration to ims are sps finally reaching the tipping point for large scale migrations? In: **2010 14th International Telecommunications Network Strategy and Planning Symposium (NETWORKS)**. [S.l.: s.n.], 2010. p. 1–6.

JUNG, Y.; IBANEZ, A. A. Improving wireless voip quality by using adaptive packet coding. **Electronics Letters**, v. 46, n. 6, p. 459–460, March 2010. ISSN 0013-5194.

KAMAILIO. ***Kamailio IMS***. 2016. <http://www.openimscore.org/>.

KIM, H.; FEAMSTER, N. Improving network management with software defined networking. **IEEE Communications Magazine**, v. 51, n. 2, p. 114–119, February 2013. ISSN 0163-6804.

KIM, H. J. et al. The qoe evaluation method through the qos-qoe correlation model. In: **2008 Fourth International Conference on Networked Computing and Advanced Information Management**. [S.l.: s.n.], 2008. v. 2, p. 719–725.

KIM, W.; PARK, G. S.; SONG, H. An effective cross-layer designed packet scheduling, call admission control, and handover system for video streaming services over {LTE} network. **Journal of Visual Communication and Image Representation**, v. 31, p. 335 – 346, 2015. ISSN 1047-3203. Disponível em: <http://www.sciencedirect.com/science/article/pii/S1047320315001285>.

KOUKAL, M.; BESTAK, R. Architecture of ip multimedia subsystem. In: **Proceedings ELMAR 2006**. [S.l.: s.n.], 2006. p. 323–326. ISSN 1334-2630.

KOVAC, A.; HALAS, M. Analysis of influence of network performance parameters on voip call quality. **Knowledge in Telecommunication Technologies and Optics**, p. 26–30, 2010.

KUSHMAN, N.; KANDULA, S.; KATABI, D. Can you hear me now?!: It must be bgp. **SIGCOMM Comput. Commun. Rev.**, ACM, New York, NY, USA, v. 37, n. 2, p. 75–84, mar. 2007. ISSN 0146-4833. Disponível em: <http://doi.acm.org/10.1145/1232919.1232927>.

LASZEWSKI, G. von et al. Comparison of multiple cloud frameworks. In: **2012 IEEE Fifth International Conference on Cloud Computing**. [S.l.: s.n.], 2012. p. 734–741. ISSN 2159-6182.

LEE, Y.; LEE, J.; SONG, J. Design and implementation of wireless {PKI} technology suitable for mobile phone in mobile-commerce. **Computer Communications**, v. 30, n. 4, p. 893 – 903, 2007. ISSN 0140-3664. Nature-Ispired Distributed Computing. Disponível em: <http://www.sciencedirect.com/science/article/pii/S0140366406004038>.

LI, L. et al. Adaptive voice stream multicast over low-power wireless networks. In: **2010 31st IEEE Real-Time Systems Symposium**. [S.l.: s.n.], 2010. p. 292–301. ISSN 1052-8725.

LIU, J. et al. Scalable application-aware resource management in software defined networking. In: **2015 17th International Conference on Transparent Optical Networks (ICTON)**. [S.l.: s.n.], 2015. p. 1–5. ISSN 2162-7339.

LIZHONG, W. et al. An adaptive forward error control method for voice communication. In: **2010 International Conference on Networking and Digital Society**. [S.l.: s.n.], 2010. v. 2, p. 186–189.

LU, J.; DOUSSON, C.; KRIEF, F. Distributed self-diagnosis algorithm for voip service over ims network. In: **2015 International Conference on Cyber-Enabled Distributed Computing and Knowledge Discovery**. [S.l.: s.n.], 2015. p. 186–191.

MAGEDANZ, T.; SCHREINER, F. QoS-aware multi-cloud brokering for NGN services: Tangible benefits of elastic resource allocation mechanisms. In: **Communications and Electronics (ICCE), 2014 IEEE Fifth International Conference on**. [S.l.: s.n.], 2014. p. 168–173.

MARTINS, J. et al. Clickos and the art of network function virtualization. In: **Proceedings of the 11th USENIX Conference on Networked Systems Design and Implementation**. Berkeley, CA, USA: USENIX Association, 2014. (NSDI'14), p. 459–473. ISBN 978-1-931971-09-6. Disponível em: <http://dl.acm.org/citation.cfm?id=2616448.2616491>.

MASTELIC, T. et al. Cloud computing: Survey on energy efficiency. **ACM Comput. Surv.**, ACM, New York, NY, USA, v. 47, n. 2, p. 33:1–33:36, dez. 2014. ISSN 0360-0300. Disponível em: <http://doi.acm.org/10.1145/2656204>.

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

_____. OpenFlow: Enabling Innovation in Campus Networks. **SIGCOMM Comput. Commun. Rev.**, v. 38, n. 2, p. 69–74, mar. 2008. ISSN 0146-4833. Disponível em: <http://doi.acm.org/10.1145/1355734.1355746>.

MICROSOFT. **Microsoft Azure Documentation | Microsoft Docs**. 2017. Disponível em: <https://docs.microsoft.com/en-us/azure/>.

MIJUMBI, R. et al. Network function virtualization: State-of-the-art and research challenges. **IEEE Communications Surveys Tutorials**, v. 18, n. 1, p. 236–262, Firstquarter 2016. ISSN 1553-877X.

MILOJIčIć, D.; LLORENTE, I. M.; MONTERO, R. S. Opennebula: A cloud management tool. **IEEE Internet Computing**, v. 15, n. 2, p. 11–14, March 2011. ISSN 1089-7801.

MKWAWA, I. H. et al. Feedback-free early voip quality adaptation scheme in next generation networks. In: **2010 IEEE Global Telecommunications Conference GLOBECOM 2010**. [S.l.: s.n.], 2010. p. 1–5. ISSN 1930-529X.

MUELLER, J.; MAGEDANZ, T. Towards a generic application aware network resource control function for next-generation-networks and beyond. In: **2012 International Symposium on Communications and Information Technologies (ISCIT)**. [S.l.: s.n.], 2012. p. 877–882.

MYAKOTNYKH, E. S.; THOMPSON, R. A. Adaptive rate voice over ip quality management algorithm. **International Journal on Advances in Telecommunications**, Citeseer, v. 2, n. 2, p. 98–110, 2009.

MYERS, M. et al. **X. 509 Internet public key infrastructure online certificate status protocol-OCSP**. [S.l.], 1999.

NAGIOS. **Nagios - Network, Server and Log Monitoring Software**. 2017. Disponível em: <https://www.nagios.com/>.

NAKAMURA, T. Feasibility study for further advancement for e-utra (lte-advanced). **3rd Gen. Partner. Proj., Sophia Antipolis, France, TS36**, v. 912, p. V9, 2011.

NURMI, D. et al. The eucalyptus open-source cloud-computing system. In: **Proceedings of the 2009 9th IEEE/ACM International Symposium on Cluster Computing and the Grid**. Washington, DC, USA: IEEE Computer Society, 2009. (CCGRID '09), p. 124–131. ISBN 978-0-7695-3622-4. Disponível em: <http://dx.doi.org/10.1109/CCGRID.2009.93>.

ONOS. ***Open Network Operating System***. 2015. <http://onosproject.org/>.

_____. **Falcon - ONOS - Wiki**. 2016. Disponível em: <https://wiki.onosproject.org/display/ONOS/Downloads>.

_____. **ONOS - Docker Hub**. 2017. Disponível em: <https://hub.docker.com/r/onosproject/onos/>.

OPENDAYLIGHT. ***OpenDayLight SDN Controller***. 2015. <https://www.opendaylight.org/>.

OPENIMS. ***Open Source IMSCore***. 2016. <http://www.openimscore.org/>.

OPENNEBULA. **OpenNebula – Flexible Enterprise Cloud Made Simple**. 2017. Disponível em: <https://opennebula.org/>.

OPENSTACK. **Fuel - OpenStack**. 2017.

_____. **OpenStack Docs: Manage flavors**. 2017. Disponível em: <https://docs.openstack.org/admin-guide/cli-manage-flavors.html>.

_____. ***OpenStack Open Source Cloud Computing Software***. 2017. Disponível em: <https://www.openstack.org/>.

OPENVPN. **OpenVPN - Open Source VPN**. 2017. Disponível em: <https://openvpn.net/>.

PANDEY, A. K.; KUMAR, S.; KUMAR, K. Dynamic call admission control for qos provision in mobile multimedia networks using artificial neural networks. In: **2016 2nd International Conference on Next Generation Computing Technologies (NGCT)**. [S.l.: s.n.], 2016. p. 355–361.

PARADOWSKI, A.; LIU, L.; YUAN, B. Benchmarking the performance of openstack and cloudstack. In: **2014 IEEE 17th International Symposium on Object/Component/Service-Oriented Real-Time Distributed Computing**. [S.l.: s.n.], 2014. p. 405–412. ISSN 1555-0885.

PODDAR, R.; VISHNOI, A.; MANN, V. Haven: Holistic load balancing and auto scaling in the cloud. In: **2015 7th International Conference on Communication Systems and Networks (COMSNETS)**. [S.l.: s.n.], 2015. p. 1–8. ISSN 2155-2487.

PROCHNIK, V.; UNE, M. Y. 3. the computer science academic community and the diffusion of the internet in brazil. **Infrastructures, Information Systems and the Economics of Innovation, ed. by C. Avgerou and RL La Rovere, Cheltenham: Edward Elgar Publishing**, 2003.

PUTHAL, D. et al. Cloud computing features, issues, and challenges: A big picture. In: **Computational Intelligence and Networks (CINE), 2015 International Conference on**. [S.l.: s.n.], 2015. p. 116–123. ISSN 2375-5822.

RAKOTOARIVELO, T. et al. Omf: A control and management framework for networking testbeds. **SIGOPS Oper. Syst. Rev.**, ACM, New York, NY, USA, v. 43, n. 4, p. 54–59, jan. 2010. ISSN 0163-5980. Disponível em: <http://doi.acm.org/10.1145/1713254.1713267>.

REGO, P. A. L. et al. An openflow-based elastic solution for cloud-cdn video streaming service. In: **2015 IEEE Global Communications Conference (GLOBECOM)**. [S.l.: s.n.], 2015. p. 1–7.

RICHARDSON, L.; RUBY, S. **RESTful web services**. [S.l.]: "O'Reilly Media, Inc.", 2008.

RIX, A. et al. Perceptual evaluation of speech quality (pesq), an objective method for end-to-end speech quality assessment of narrowband telephone networks and speech codecs. **ITU-T Recommendation**, v. 862, 2001.

ROLOFF, E. et al. Evaluating high performance computing on the windows azure platform. In: **2012 IEEE Fifth International Conference on Cloud Computing**. [S.l.: s.n.], 2012. p. 803–810. ISSN 2159-6182.

ROSENBERG, J. et al. **SIP: session initiation protocol**. [S.l.], 2002.

RYU. ***Ryu SDN Controller***. 2015. <http://osrg.github.io/ryu/>.

SALMAN, O. et al. Sdn controllers: A comparative study. In: **2016 18th Mediterranean Electrotechnical Conference (MELECON)**. [S.l.: s.n.], 2016. p. 1–6.

SALMITO, T. et al. Fibre-an international testbed for future internet experimentation. In: **Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos-SBRC 2014**. [S.l.: s.n.], 2014. p. p–969.

SCHULZRINNE, H. et al. **RTP: A transport protocol for real-time applications**. [S.l.], 2003.

SCHULZRINNE, H.; WEDLUND, E. Application-layer mobility using sip. In: IEEE. **Service Portability and Virtual Customer Environments, 2000 IEEE**. [S.l.], 2000. p. 29–36.

SCOTT-HAYWARD, S. Design and deployment of secure, robust, and resilient sdn controllers. In: **Proceedings of the 2015 1st IEEE Conference on Network Softwarization (NetSoft)**. [S.l.: s.n.], 2015. p. 1–5.

SEFRAOUI, O.; AISSAOUI, M.; ELEULDJ, M. Openstack: toward an open-source solution for cloud computing. **International Journal of Computer Applications**, Foundation of Computer Science, v. 55, n. 3, 2012.

SHANNON, J. et al. Enhancing multimedia qoe via more effective time synchronisation over 802.11 networks. In: **Proceedings of the 7th International Conference on Multimedia Systems**. New York, NY, USA: ACM, 2016. (MMSys '16), p. 25:1–25:9. ISBN 978-1-4503-4297-1. Disponível em: <http://doi.acm.org/10.1145/2910017.2910615>.

SHARKH, M. A. et al. Resource allocation in a network-based cloud computing environment: design challenges. **IEEE Communications Magazine**, v. 51, n. 11, p. 46–52, November 2013. ISSN 0163-6804.

SHERWOOD, R. et al. Carving research slices out of your production networks with openflow. **SIGCOMM Comput. Commun. Rev.**, ACM, New York, NY, USA, v. 40, n. 1, p. 129–130, jan. 2010. ISSN 0146-4833. Disponível em: <http://doi.acm.org/10.1145/1672308.1672333>.

_____. Flowvisor: A network virtualization layer. **OpenFlow Switch Consortium, Tech. Rep**, p. 1–13, 2009.

SIMSEK, M. et al. 5g-enabled tactile internet. **IEEE Journal on Selected Areas in Communications**, v. 34, n. 3, p. 460–473, March 2016. ISSN 0733-8716.

SPARKS, R. J. The session initiation protocol (sip) refer method. 2003.

STALLINGS, W. **SNMP,SNMPV2,Snmpv3,and RMON 1 and 2**. 3rd. ed. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1998. ISBN 0201485346.

STANCU, A. L. et al. A comparison between several software defined networking controllers. In: **2015 12th International Conference on Telecommunication in Modern Satellite, Cable and Broadcasting Services (TELSIKS)**. [S.l.: s.n.], 2015. p. 223–226.

STREIJL, R. C.; WINKLER, S.; HANDS, D. S. Mean opinion score (mos) revisited: methods and applications, limitations and alternatives. **Multimedia Systems**, v. 22, n. 2, p. 213–227, 2016. ISSN 1432-1882. Disponível em: <http://dx.doi.org/10.1007/s00530-014-0446-1>.

SUñé, M. et al. Design and implementation of the {OFELIA} {FP7} facility: The european openflow testbed. **Computer Networks**, v. 61, p. 132 – 150, 2014. ISSN 1389-1286. Special issue on Future Internet Testbeds – Part I. Disponível em: <http://www.sciencedirect.com/science/article/pii/S1389128613004301>.

TANENBAUM, A. **Computer Networks**. 4th. ed. [S.l.]: Prentice Hall Professional Technical Reference, 2002. ISBN 0130661023.

TARREAU, W. **HAProxy-the reliable, high-performance TCP/HTTP load balancer**. 2012.

TARUTE, A.; GATAUTIS, R. ICT Impact on SMEs Performance. **Procedia - Social and Behavioral Sciences**, v. 110, p. 1218 – 1225, 2014. ISSN 1877-0428. Disponível em: <http://www.sciencedirect.com/science/article/pii/S1877042813056085>.

TEBBANI, B.; HADDADOU, K. Codec-based adaptive qos control for vowlan with differentiated services. In: **2008 1st IFIP Wireless Days**. [S.l.: s.n.], 2008. p. 1–5. ISSN 2156-9711.

THOM, G. A. H. 323: the multimedia communications standard for local area networks. **IEEE Communications Magazine**, IEEE, v. 34, n. 12, p. 52–56, 1996.

THORPE, C. et al. imos: Enabling voip qos monitoring at intermediate nodes in an openflow sdn. In: **2016 IEEE International Conference on Cloud Engineering Workshop (IC2EW)**. [S.l.: s.n.], 2016. p. 76–81.

TRANORIS, C. et al. Integrating openflow in ims networks and enabling for future internet researchand experimentation. In: _____. **The Future Internet: Future Internet Assembly 2013: Validated Results and New Horizons**. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013. p. 77–88. ISBN 978-3-642-38082-2.

TREMA. ***Full-Stack OpenFlow Framework in Ruby and C***. 2015. <https://trema.github.io/trema/>.

VOORSLUYS, W.; BROBERG, J.; BUYYA, R. Introduction to cloud computing. **Cloud computing: Principles and paradigms**, John Wiley & Sons, Inc., p. 1–41, 2011.

VORAS, I. et al. Evaluating open-source cloud computing solutions. In: **2011 Proceedings of the 34th International Convention MIPRO**. [S.l.: s.n.], 2011. p. 209–214.

WANG, S.; LI, D.; XIA, S. The problems and solutions of network update in sdn: A survey. In: **Computer Communications Workshops (INFOCOM WKSHPS), 2015 IEEE Conference on**. [S.l.: s.n.], 2015. p. 474–479.

WEI, L.; WU, M.; WU, D. Wireless voip adaptive source rate control algorithm. In: **2009 5th International Conference on Wireless Communications, Networking and Mobile Computing**. [S.l.: s.n.], 2009. p. 1–4. ISSN 2161-9646.

WEN, X. et al. Comparison of open-source cloud management platforms: Openstack and opennebula. In: **2012 9th International Conference on Fuzzy Systems and Knowledge Discovery**. [S.l.: s.n.], 2012. p. 2457–2461.

XEN. **Why The Xen Project?** 2016.

YAMEI, F.; QING, L.; QI, H. Research and comparative analysis of performance test on sdn controller. In: **2016 First IEEE International Conference on Computer Communication and the Internet (ICCCI)**. [S.l.: s.n.], 2016. p. 207–210.

YUHE, S.; JIE, X. New solutions of voip on multi-hop wireless network. In: **2009 IITA International Conference on Control, Automation and Systems Engineering (case 2009)**. [S.l.: s.n.], 2009. p. 199–202.

ZABBIX. **Zabbix :: The Enterprise-Class Open Source Network Monitoring Solution**. 2017. Disponível em: <http://www.zabbix.com/>.

ZHANG, Q.; CHENG, L.; BOUTABA, R. Cloud computing: state-of-the-art and research challenges. **Journal of internet services and applications**, Springer, v. 1, n. 1, p. 7–18, 2010.

ZHANG, X. et al. Modeling and analysis of skype video calls: Rate control and video quality. **IEEE Transactions on Multimedia**, v. 15, n. 6, p. 1446–1457, Oct 2013. ISSN 1520-9210.

ZHAO, Y.; IANNONE, L.; RIGUIDEL, M. On the performance of sdn controllers: A reality check. In: **2015 IEEE Conference on Network Function Virtualization and Software Defined Network (NFV-SDN)**. [S.l.: s.n.], 2015. p. 79–85.

ZHOU, J.; SHE, X.; CHEN, L. Source and channel coding adaptation for optimizing voip quality of experience in cellular systems. In: **2010 IEEE Wireless Communication and Networking Conference**. [S.l.: s.n.], 2010. p. 1–6. ISSN 1525-3511.

Apêndices

APÊNDICE **A**

Codificações complementares à solução

Foram desenvolvidos algoritmos complementares a solução proposta. Para tal foi utilizada a linguagem Python e Shell script, devido a compatibilidade com sistemas operacionais Linux.

A.1 Rotina *cpuhealth*

As instâncias do *OpenStack* possuem o arquivo de inicialização automático dentro do diretório "/etc/init.d/" que faz chamada ao algoritimo *Statistics.py*

```
#!/bin/sh
    python /home/debian/Statistics.py -H <ORCHESTRATOR_IP>
    -p 7070 > /dev/null 2>&1 < /dev/null &
```

A.2 Algoritimo *Statistics.py*

Algoritimo incorporado as instâncias do *OpenStack* que é inicializada instantaneamente junto com o sistema operacional. A rotina envia estatísticas de *hardware* em intervalos ajustáveis de tempo. As estatísticas são consumidas e trabalhadas pelo *Orchestrator*.

```
__author__ = "rodrigo"
__date__ = "$Nov 16, 2016 5:32:06 PM$"

import json
import optparse
import random
import requests
import socket
import sys
import time
```

```
def collectCpuInfo(metricDict, cpuHistory, metricPath):

    procStatFile = open("/proc/stat")
    statLines = procStatFile.readlines()
    procStatFile.close()

    cpuLine = statLines[0].split()
    if not "cpu" in cpuLine[0]:
        print "collection of CPU Info failed"
        return

    user = int(cpuLine[1])
    nice = int(cpuLine[2])
    system = int(cpuLine[3])
    idle = int(cpuLine[4])
    iowait = int(cpuLine[5])
    irq = int(cpuLine[6])
    softirq = int(cpuLine[7])


    if cpuHistory.has_key('user'):
        curUser = user - cpuHistory['user']
        curNice = nice - cpuHistory['nice']
        curSystem = system - cpuHistory['system']
        curIdle = idle - cpuHistory['idle']
        curIowait = iowait - cpuHistory['iowait']
        curIrq = irq - cpuHistory['irq']
        curSoftirq = softirq - cpuHistory['softirq']
        totalCPU = curUser + curNice + curSystem + curIdle +
            curIowait + curIrq + curSoftirq


    m = {}
    m['type'] = 'IntCounter'
    m['name'] = metricPath + '|CpuUsage:user'
    m['value']= "{0}".format(int(float(curUser)/totalCPU *
        100 + .5))
```

```
metricDict['metrics'].append(m)

m = {}
m['type'] = 'IntCounter'
m['name'] = metricPath + '|CpuUsage:nice'
m['value']= "{0}".format(int(float(curNice)/totalCPU *
    100 + .5))
metricDict['metrics'].append(m)

m = {}
m['type'] = 'IntCounter'
m['name'] = metricPath + '|CpuUsage:system'
m['value']= "{0}".format(int(float(curSystem)/totalCPU *
    100 + .5))
metricDict['metrics'].append(m)

m = {}
m['type'] = 'IntCounter'
m['name'] = metricPath + '|CpuUsage:idle'
m['value']= "{0}".format(int(float(curIdle)/totalCPU *
    100 + .5))
metricDict['metrics'].append(m)

m = {}
m['type'] = 'IntCounter'
m['name'] = metricPath + '|CpuUsage:iowait'
m['value']= "{0}".format(int(float(curIowait)/totalCPU *
    100 + .5))
metricDict['metrics'].append(m)

m = {}
m['type'] = 'IntCounter'
m['name'] = metricPath + '|CpuUsage:irq'
m['value']= "{0}".format(int(float(curIrq)/totalCPU *
    100 + .5))
metricDict['metrics'].append(m)

m = {}
m['type'] = 'IntCounter'
```

```
    m['name'] = metricPath + '|CpuUsage:softirq'
    m['value']= "{0}".format(int(float(curSoftirq)/totalCPU *
        100 + .5))
    metricDict['metrics'].append(m)


    cpuHistory['user'] = user
    cpuHistory['nice'] = nice
    cpuHistory['system'] = system
    cpuHistory['idle'] = idle
    cpuHistory['iowait'] = iowait
    cpuHistory['irq'] = irq
    cpuHistory['softirq'] = softirq

def collectLoadAvg(metricDict, metricPath):

    procLoadavgFile = open("/proc/loadavg")
    loadavgLines = procLoadavgFile.readlines()
    procLoadavgFile.close()

    if len(loadavgLines) < 1:
        print ("Falha ao coletar estatisticas" +
        "gerais de consumo de Hardware")
        return

    loadLine = loadavgLines[0].split()
    load1 = int(float(loadLine[0]) + 0.5)
    load5 = int(float(loadLine[1]) + 0.5)
    load15 = int(float(loadLine[2]) + 0.5)

    m = {}
    m['type'] = 'IntCounter'
    m['name'] = metricPath + '|Load:load1'
    m['value']= "{0}".format(load1)
    metricDict['metrics'].append(m)

    m = {}
    m['type'] = 'IntCounter'
    m['name'] = metricPath + '|Load:load5'
```

```
    m['value']= "{0}".format(load5)
    metricDict['metrics'].append(m)

    m = {}
    m['type'] = 'IntCounter'
    m['name'] = metricPath + '|Load:load15'
    m['value']= "{0}".format(load15)
    metricDict['metrics'].append(m)


def collectMemInfo(metricDict, metricPath):

    procMemFile = open("/proc/meminfo")
    memfileLines = procMemFile.readlines()
    procMemFile.close()

    if len(memfileLines) < 1:
        print "collection of loadAverage statistics failed"
        return


    for l in memfileLines:
        elems = l.split()
        m = {}
        m['type'] = 'IntCounter'
        m['name'] = metricPath +
        '|MemInfoKB:{0}'.format(elems[0][:-1])
        m['value']= "{0}".format(elems[1])
        metricDict['metrics'].append(m)

def main(argv):

    parser = optparse.OptionParser()

    parser.add_option("-v", "--verbose", help = "verbose output",
    dest = "verbose", default = False, action = "store_true")

    parser.add_option("-H", "--hostname", default = "localhost",
    help = "Statistics.py is running and sending messages" +
```

```
    "to Orchestrator with IP", dest = "hostname")

parser.add_option("-p", "--port", help
    = "Statistics.py is connected with",
type = "int", default = 8080, dest = "port")

parser.add_option("-m", "--metric_path", help =
    "metric path header for all metrics",
dest = "metricPath", default =
    "linuxStats|{0}".format(socket.gethostname()))

(options, args) = parser.parse_args();

if options.verbose == True:
    print "Verbose enabled"


url = "http://{0}:{1}/apm/metricFeed".format(options.hostname,
options.port)
headers = {'content-type': 'application/json'}

if options.verbose:
    print "Submitting to: {0}".format(url)

submissionCount = 0
cpuHistory = {}


while True:
    metricDict = {'metrics' : []}


collectCpuInfo(metricDict, cpuHistory, options.metricPath)

collectLoadAvg(metricDict, options.metricPath)

collectMemInfo(metricDict, options.metricPath)

try:
```

```
            r = requests.post(url, data = json.dumps(metricDict),
            headers = headers)
        except:
            print "Erro ao Fazer Post com destino ao Orchestrator
            na porta UDP"

        if options.verbose:
            print "jsonDump:"
            print json.dumps(metricDict, indent = 4)


        print "Response:"
        print r.text


        print "StatusCode: {0}".format(r.status_code)

        submissionCount += 1
        print "Submitted metric: {0}".format(submissionCount)
        time.sleep(5)


if __name__ == "__main__":
    main(sys.argv)
```