UNIVERSIDADE FEDERAL DO CEARÁ

CAMPUS QUIXADÁ

TECNÓLOGO EM REDES DE COMPUTADORES

ROSELI DA ROCHA BARBOSA

ANÁLISE DE DESEMPENHO DAS TECNOLOGIAS DE VIRTUALIZAÇÃO DE REDE DA PLATAFORMA OPENSTACK

QUIXADÁ

2017

ROSELI DA ROCHA BARBOSA

ANÁLISE DE DESEMPENHO DAS TECNOLOGIAS DE VIRTUALIZAÇÃO DE REDE DA PLATAFORMA OPENSTACK

Monografia apresentada ao Curso de Redes de Computadores da Universidade Federal do Ceará, como requisito parcial à obtenção do título de tecnólogo em Redes de Computadores. Área de concentração: Computação.

Orientador: Prof. Dr. Paulo Antonio Leal Rego

QUIXADÁ

2017

Dados Internacionais de Catalogação na Publicação
Universidade Federal do Ceará
Biblioteca Universitária
Gerada automaticamente pelo módulo Catalog, mediante os dados fornecidos pelo(a) autor(a)

B211a Barbosa, Roseli da Rocha.
Análise de desempenho das tecnologias de virtualização de rede da plataforma OpenStack / Roseli da Rocha Barbosa. – 2017.
61 f. : il. color.

Trabalho de Conclusão de Curso (graduação) – Universidade Federal do Ceará, Campus de Quixadá, Curso de Redes de Computadores, Quixadá, 2017.
Orientação: Prof. Dr. Paulo Antonio Leal Rego.

1. Generic Routing Encapsulation. 2. Lan extensível virtual. 3. OpenStack. 4. Virtualização. 5. Análise - Medição de desempenho. I. Título.

CDD 004.6

ROSELI DA ROCHA BARBOSA

ANÁLISE DE DESEMPENHO DAS TECNOLOGIAS DE VIRTUALIZAÇÃO DE REDE DA PLATAFORMA OPENSTACK

Monografia apresentada ao Curso de Redes de Computadores da Universidade Federal do Ceará, como requisito parcial à obtenção do título de tecnólogo em Redes de Computadores. Área de concentração: Computação.

Aprovada em: ____/____/____

BANCA EXAMINADORA

Prof. Dr. Paulo Antonio Leal Rego (Orientador)
Universidade Federal do Ceará – UFC

Prof. Dr. Arthur de Castro Callado
Universidade Federal do Ceará - UFC

Prof. Me. Michel Sales Bonfim
Universidade Federal do Ceará - UFC

Dedico esse trabalho em especial a minha mãe Eliene por toda a fonte de carinho e incentivo que me ajudou em todos os momentos de minha vida.

AGRADECIMENTOS

Agradeço a Deus por ter me dado saúde e força para superar as dificuldades.

À minha família que sempre me apoiou em minhas decisões.

Ao meu orientador Paulo Antônio Leal Rego, que se dispôs a me orientar e me auxiliar em toda a construção deste estudo e que tenho uma eterna gratidão.

A Jean Dias, Rodrigo Barbieri e Rodrigo Freitas por todo apoio e incentivo nos estudos com o *OpenStack*. Obrigada pela confiança que em mim depositaram para a realização deste trabalho.

Aos professores Michel Sales, Ticiana Linhares, Arthur Callado, Rafael Braga, Jefferson Carvalho e Marcos Dantas por serem verdadeiros arquitetos de sonhos e engenheiros do futuro. Obrigada por cada contribuição para a minha formação pessoal e profissional durante a graduação.

Aos meus amigos de classe, Alisson, Juliana, Paulo Rennê, Isac, Tiago, Micaele, Mateus Sousa, Mateus Nobre, Cleilson, Pedro, Assis, Aleff Brunno, Andreza, Sara, Wanderson Luan, Leilton, Samuel e Wilkinson que com o passar dos anos se tornaram verdadeiros irmãos. Obrigada por todos os momentos de carinho, companheirismo e dedicação nessa longa caminhada.

Obrigada a todos que, mesmo não estando citados aqui, tanto contribuíram para a conclusão desta etapa na qual me fez me tornar a pessoa que sou hoje.

“Cada sonho que você deixa pra trás, é um pedaço do seu futuro que deixa de existir.”

(Steve Jobs)

RESUMO

As VLANs (*Virtual Local Area Network*), ou redes locais virtuais, visam fornecer o isolamento do tráfego de rede entre os usuários pertencentes à mesma arquitetura física. Nos últimos anos, a computação em nuvem despontou como um dos maiores avanços da tecnologia da informação e revolucionou a gestão de recursos computacionais em ambientes empresariais e acadêmicos. Com a popularidade da computação em nuvem, surgiu a necessidade de ampliar o suporte a múltiplos inquilinos pertencentes a redes logicamente independentes, fomentando o advento da virtualização de redes. Diversos estudos com plataformas de nuvem vêm sendo realizados, buscando aperfeiçoar as técnicas de virtualização de redes existentes, apontando a melhor solução para cada ambiente proposto. Nesse cenário, o *OpenStack* se tornou propulsor, ganhando cada vez mais novos adeptos por sua natureza aberta e atualmente ser considerado um dos maiores orquestradores de nuvem. O *OpenStack* possibilita a implementação de modelos flexíveis de redes, atendendo às necessidades de grandes centros de dados que utilizam técnicas de sobreposição de redes para fornecer infraestruturas escaláveis. Com o intuito de avaliar o desempenho das tecnologias de virtualização de redes suportadas pela plataforma *OpenStack*, este trabalho visa comparar o desempenho de redes virtuais utilizando os protocolos de tunelamento VXLAN e GRE, com *Linux Bridge* e *Open vSwitch*. Para avaliar o desempenho e escalabilidade da infraestrutura de nuvem, foram analisadas métricas como vazão, taxa de perda de pacotes, latência e jitter para fluxos de conexão TCP e UDP usando a ferramenta de geração e medição de tráfego Uperf. Com os experimentos, foi possível constatar que cenários utilizando *Open vSwitch* com VXLAN obtiveram maiores taxas de vazão entre *compute nodes* diferentes, mostrando dessa forma maiores taxas de perda de pacotes. No entanto, *Linux Bridge* com VXLAN apresentou menores taxas de latência e jitter na maioria dos resultados.

Palavras-chave: Encapsulamento de roteamento genérico. Rede local virtual. *OpenStack*. Virtualização. Análise - Medição de desempenho

ABSTRACT

VLANs (Virtual Local Area Network) aim to provide the network traffic isolation between users pertain to the same physical architecture. In recent years, computing has emerged as one of the greatest information technology improvement. Which has revolutionized the computing resource management in business and academic environments. The popularity of cloud computing has made to appeared the need to increase support for multiple hosts belonging a logically independent networks, promoting the advent of networks virtualization. Several studies have been carried out with cloud computing platforms, aiming to improve network virtualization existing techniques, pointing out the best solution to each proposed environment. In this scenario, the OpenStack became propulsor, increasing new adepts by its open essence and currently be considered one of largest cloud orchestrators. The OpenStack enables the implantation of flexible models of network, answering the needs of big data centers that use superposition techniques to provide scalable structures. With the purpose of to measure the networks virtualization technologies performance supported by OpenStack platform, this paper aims to compare the networks virtualization performance utilizing the VXLAN and GRE tunneling protocols, with Linux Bridge and Open vSwitch. To evaluate performance and scalability of cloud infrastructure, were analyzed metrics as throughput, packets loss, latency and jitter for connection fluxes TCP and UDP by using the traffic generation and measuring tool Uperf. With the experiments was possible to verify that scenarios using Open vSwitch with VXLAN obtained larger throughput taxes between diffents compute nodes, exposing this way bigger packets loss taxes. However, Linux Bridge with VXLAN demonstrates smaller latency and jitter taxes in the most of results.

Keywords: Generic Routing Encapsulation. Lan extensível virtual. OpenStack. Virtualization. Analysis - Performance measurement

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

API	*Application Programming Interface*
DPI	*Deep Packet Inspection*
GRE	*Generic Routing Encapsulation*
HTTP	*Hypertext Transfer Protocol*
IaaS	*Infrastructure as a Service*
ML2	*Plugin Modular Layer 2*
NAT	*Network Address Translation*
NFS	*Network File System*
NFV	*Network Functions Virtualization*
PaaS	*Platform as a Service*
PCIe	PCI *Express*
SaaS	*Software as a Service*
SDN	*Software Defined Network*
STP	*Spanning Tree Protocol*
STT	*Stateless Transport Tunneling*
VLAN	*Virtual Local Area Network*
VNI	VXLAN *Network Identifier*
VPN	*Virtual Private Network*
VTEP	VXLAN *Tunnel Endpoint*
VXLAN	*Virtual Extensible Local Area Network*

SUMÁRIO

1 INTRODUÇÃO

Com o advento da Internet, diversos serviços passaram a ser acessados e consumidos com uma maior facilidade e flexibilidade de uso. Em contrapartida, a crescente necessidade de ampliação de recursos computacionais em ambientes empresariais e acadêmicos fomentou o surgimento da computação em nuvem, que também se utiliza da Internet para propiciar um amplo acesso aos seus usuários (BUYYA; VECCHIOLA; SELVI, 2013).

Na computação em nuvem, recursos são entregues como serviços, possibilitando total garantia da extensão da capacidade computacional conforme a necessidade. Devido a esse modelo de negócio, nos últimos anos, os usuários têm movido seus dados e aplicações para a nuvem, caracterizando uma ampla adesão a esse tipo de serviço (SOUSA; MOREIRA; MACHADO, 2009).

As principais tecnologias sobre a quais a computação em nuvem se baseia não são novas, mas sim soluções previamente adotadas e estabelecidas. Dessa forma, a computação em nuvem não é algo novo, porém torna-se inovadora por reunir diversos componentes em uma mesma infraestrutura (CAROLAN et al., 2009).

Um dos componentes chave da computação em nuvem é a virtualização. Técnicas de virtualização têm sido utilizadas para facilitar a gestão de recursos computacionais em uma infraestrutura de nuvem. Com a virtualização, há a garantia do compartilhamento, isolamento e escalabilidade dos recursos computacionais, abstraindo e dinamizando a utilização das características físicas de uma plataforma computacional (JAIN; PAUL, 2013).

Com a popularização da virtualização, a comunidade de pesquisa em Redes de Computadores passou a investir constantemente em iniciativas que utilizam conceitos de virtualização, visando atender a uma demanda crescente de usuários e ao surgimento de aplicações de rede mais avançadas. Tais iniciativas motivaram diversos estudos, permitindo a criação e implantação de diferentes arquiteturas e protocolos em uma infraestrutura (CARAPINHA; JIMÉNEZ, 2009).

A virtualização de redes vem sendo considerada uma excelente alternativa para os grandes centros de dados. Embora não seja um conceito totalmente novo, ainda se tem buscado aprimorar a sua aplicabilidade. Com a virtualização de redes, tornou-se possível fornecer uma plataforma onde múltiplas redes coexistem em uma mesma infraestrutura com características distintas (CARAPINHA; JIMÉNEZ, 2009).

Em ambientes de computação em nuvem, a virtualização de redes é essencial para

suportar múltiplos inquilinos pertencentes a redes logicamente independentes. Nesse cenário, surgiu a necessidade de adequar o uso das LANs virtuais para lidar com as limitações no número de quadros de dados Ethernet disponíveis, quando necessária a sua adoção nos grandes centros de dados. Além disso, em redes de camada 2 (*Layer 2*) ou camada de enlace, é comum o uso do protocolo STP (do inglês, *Spanning Tree Protocol*) para resolver problemas de *loops* na rede quando houverem caminhos redundantes. No entanto, a sua funcionalidade é encarada por muitos como um problema, uma vez que o STP bloqueia o uso de enlaces para impossibilitar a replicação e o *loop* de quadros (MAHALINGAM et al., 2014).

Visando contornar as limitações das arquiteturas de redes de camada 2, surgiu o conceito de redes de virtualização por sobreposição. Na prática, isto consiste na sobreposição da camada 2 em uma rede de camada 3. Consequentemente, tornou-se possível a conectividade entre diferentes máquinas virtuais que podem estar localizadas em diferentes centros de dados ou em diferentes servidores de um mesmo centro de dados.

Tendo como objetivo melhorar o desempenho das infraestruturas de nuvem, diversos estudos abordam como utilizar técnicas de virtualização de redes em diferentes contextos. Deste modo, foi considerado diferentes fatores para a escolha da solução de nuvem para este trabalho. Entre as soluções estudadas, foi adotada a plataforma *OpenStack* que vem ganhando destaque nos últimos anos devido a sua alta escalabilidade e flexibilidade. A organização dos seus serviços se tornou um diferencial entre as demais plataformas, tornando-a a principal escolha entre os usuários, além do fato de ser base de desenvolvimento de sistemas comerciais (BARROS et al., 2015). A cada dia, a comunidade de colaboradores do projeto *OpenStack* se dissemina, e por causa desse crescimento, ele vem sendo utilizado por grandes empresas, garantindo um ciclo de lançamentos frequentes.

A plataforma *OpenStack* fornece suporte a várias tecnologias para implantação de infraestruturas de redes. Dentre as tecnologias estão VXLAN e GRE. Nesse contexto, **o objetivo geral deste trabalho é avaliar o desempenho das tecnologias de virtualização de rede disponíveis na plataforma *OpenStack***. O presente estudo utilizou a plataforma de nuvem *OpenStack* para a implantação de uma nuvem privada, por se tratar de uma solução de código aberto que realiza a entrega de rede como serviço através da virtualização.

Para cada experimento realizado buscou-se avaliar métricas como vazão TCP e UDP, taxa de perda de pacotes, latência e jitter. Além disso, considerou-se o tipo de alocação das máquinas virtuais, se pertencem ao mesmo *compute node* ou em diferentes *compute nodes*, os

tipos de tráfego (TCP ou UDP), mecanismos de comunicação (*Open vSwitch* e *Linux Bridge*) e conexões unidirecionais e bidirecionais.

1.1 Estrutura da monografia

Esta monografia está organizada da seguinte forma: o Capítulo 1 contextualiza o trabalho e apresenta os objetivos propostos. No Capítulo 2, são apresentados os fundamentos e definições referentes à computação em nuvem e virtualização de redes, com foco na plataforma *OpenStack*. O Capítulo 3 apresenta diferentes técnicas de virtualização de redes em plataformas de computação em nuvem, discutindo os estudos relacionados a este trabalho. A avaliação da proposta é definida no Capítulo 4, esclarecendo o planejamento para a criação do cenário, as métricas utilizadas e os resultados obtidos. Por fim, o Capítulo 5 aponta as considerações finais encontradas com a realização desta pesquisa.

2 FUNDAMENTAÇÃO TEÓRICA

Este Capítulo destina-se à introdução dos fundamentos conceituais sobre os temas computação em nuvem e virtualização de redes com a plataforma de nuvem *OpenStack*, que estão diretamente relacionados a este trabalho.

2.1 Virtualização

O conceito de virtualização surgiu por volta da década de 1950, embora tenha ganhado utilidade através do uso da plataforma x86 em 1980. Nesse período era comum que cada fabricante de *mainframe* tivesse seus próprios sistemas operacionais. Essa foi a principal motivação para o surgimento de máquinas virtuais, devido a necessidade da utilização de outros *softwares* disponíveis no mercado. Pensando nisso, a IBM introduziu no mercado as primeiras máquinas virtuais (MATTOS, 2008).

Em 1972, o cientista Robert P. Goldberg da universidade de Harvard em uma dissertação abordou o conceito da utilização de sistemas computacionais virtuais. No mesmo ano, a IBM lançou um *mainframe* que era capaz de executar diferentes sistemas operacionais através de um programa hipervisor. Conceitualmente, os hipervisores tem a função de garantir a alocação de recursos de *hardware* entre VMs (Máquinas Virtuais). Com o sucesso dessa tecnologia, a IBM passou a adotar este recurso em seus equipamentos, possibilitando a interação e execução de aplicações entre diferentes plataformas no mesmo *hardware* (BUENO, 2009).

Com o avanço da virtualização, a sua utilização passou a ser adotada para diferentes finalidades, como a virtualização de *desktops*, de aplicações, de armazenamento, de servidores e também a virtualização de redes, que é considerada parte fundamental para o escopo deste estudo. Em VMware (2017), os autores apresentam a seguinte definição para virtualização de rede:

> A virtualização de rede é a reprodução completa de uma rede física no *software*. Os aplicativos são executados na rede virtual exatamente da mesma maneira como se estivessem em uma rede física. A virtualização de rede apresenta serviços e dispositivos lógicos do sistema de rede (portas lógicas, *switches*, roteadores, *firewalls*, balanceadores de carga, VPNs e outros) para cargas de trabalho conectadas. As redes virtuais oferecem os mesmos recursos e garantias de uma rede física e ainda fornecem os benefícios operacionais e a independência de *hardware* da virtualização (VMWARE, 2017).

Com base nas características citadas em VMware (2017), fica ainda mais evidente a importância da virtualização para a computação em nuvem, visto que existem alguns mecanismos

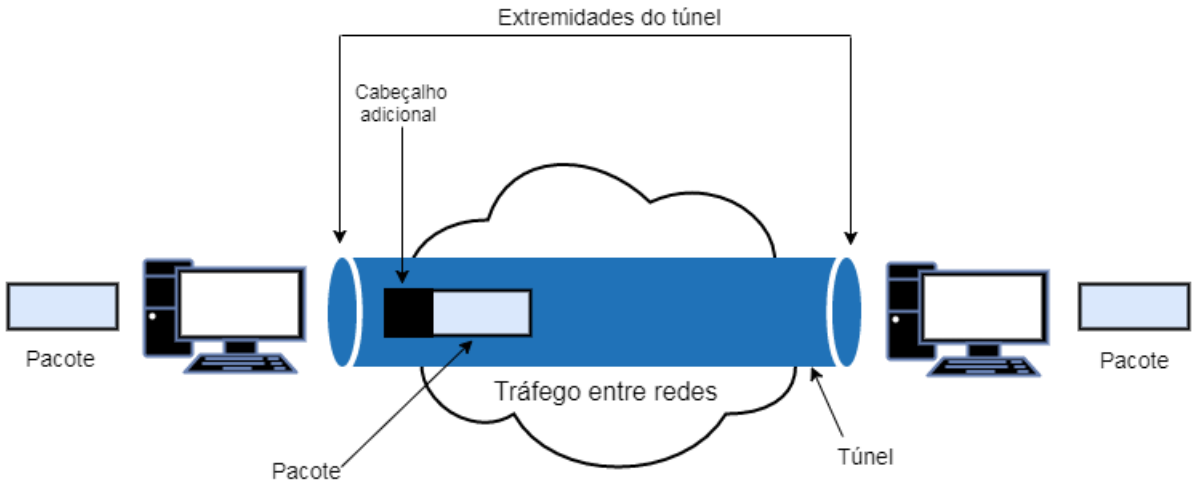
lógicos de virtualização que são fundamentais e frequentemente aplicados em infraestruturas de nuvem, como o uso de *switches* e roteadores virtuais, redes L2 e redes L3.

Através dos *switches* virtuais, é possível implementar conectividade entre VMs pertencentes ao mesmo servidor utilizando *bridges* internas, ou até mesmo em diferentes servidores através de *bridges* externas. Outro fator importante é possibilitar a implantação de redes locais virtuais para garantir o isolamento do tráfego. Semelhante à utilização dos *switches* virtuais, estão os roteadores virtuais que estão sendo cada vez mais difundidos quando se trata de implementações que aplicam os conceitos de NFV (BARROS et al., 2015).

A virtualização de redes tem sido comumente empregada em grandes centros de dados para prover conectividade em camada 2 (VLAN) ou em camada 3, quando redes de multiusuários são consideradas. Estas são vulgarmente conhecidas pela utilização de protocolos de sobreposição, como o VXLAN, NVGRE e STT (BARROS et al., 2015).

Para entender o funcionamento dos protocolos de sobreposição, é importante entender o conceito de tunelamento, que consiste na criação de um túnel virtual que realiza o encapsulamento de um protocolo, adicionando o cabeçalho de um protocolo de tunelamento (ilustrado na Figura 1). Desse modo, a técnica de tunelamento possibilita o tráfego dos pacotes em túneis virtuais em uma rede pública dando a ideia de conexão ponto a ponto.

Figura 1 – Tunelamento



Fonte: Elaborada pela autora

2.2 Computação em Nuvem

O contexto histórico da computação em nuvem foi marcado por uma série de tecnologias antecessoras, que inicialmente visavam conectar diferentes máquinas para fornecer uma maior taxa de processamento (RABELO, 2016). Desde então, já havia a necessidade da utilização de recursos computacionais para a execução de tarefas, pois era possível obter uma redução do tempo para a conclusão de trabalhos além de garantir um melhor desempenho.

Com o passar dos anos, o poder computacional aumentou ainda mais devido à disseminação dos sistemas distribuídos (SANTOS, 2008). Em consequência disso, surgiu a ideia da utilização de grades computacionais *(grid computing)*, tendo como objetivo inovador a conexão de diferentes máquinas para melhorar a taxa de processamento e capacidade de armazenamento, a fim de otimizar a execução das tarefas computacionais. Dessa forma, havia uma abstração para quem utilizava essa tecnologia, pois tinha-se a visão de um ambiente centralizado.

O termo *grid computing* originou-se devido aos sistemas de energia elétrica conhecidos como *power grids*. Essa associação deu-se, pois a eletricidade é utilizada sem ter o conhecimento de onde ela foi gerada (RABELO, 2016). Da mesma forma funcionam as grades computacionais, através da utilização de sistemas distribuídos é possível fornecer recursos heterogêneos, muitas vezes até separados geograficamente, mas que dão suporte à criação de aplicações paralelas.

Apesar de todos os benefícios fornecidos pelas *grids*, existiam algumas desvantagens, entre elas o alto custo para a sua utilização. Organizações ou empresas que tivessem a necessidade de utilizar tal serviço deveriam investir em tecnologias de alto custo e em profissionais para realizar o gerenciamento. Outro fator importante é que com o tempo os equipamentos de *hardware* são desgastados e há a necessidade de sua substituição. Em resumo, para a criação de um cenário como esse seria necessário grandes investimentos a curto e a longo prazo.

Com o advento da Internet, surgiu no mercado um novo padrão da utilização de recursos sob demanda, que foi conhecido como *utility computing*. O termo originou-se das chamadas *utilities*, isto é, são empresas que tem como plano de negócio a cobrança pelos serviços de acordo com o que é consumido (ARMBRUST et al., 2009). Esse modelo de serviço foi inicialmente criado pela *Amazon*, que passou a ofertar a aquisição de recursos de processamento, armazenamento e transmissão de dados de forma temporária.

De acordo com Sousa, Moreira e Machado (2009), o principal motivo para a utilização de serviços baseados em *utility computing* deve-se ao fato de que os gastos serão de maneira linear, ou seja, de acordo com o que for consumido, não necessitando de investimentos iniciais em equipamentos de TI (Tecnologia da Informação, do inglês *Information Technology*). Diante disso, o uso da *utility computing* se apresentava como um excelente investimento.

A estratégia de negócio da *utility computing* é propor o fornecimento total de disponibilidade, ou seja, os usuários podem ler e gravar dados a qualquer momento, sem nunca serem bloqueados; os tempos de respostas são quase constantes e não dependem do número de usuários simultâneos, do tamanho do banco de dados ou de qualquer parâmetro do sistema. Os usuários não precisam se preocupar com *backups*, pois se os componentes falharem, o provedor é responsável por substituí-los e tornar os dados disponíveis em tempo hábil por meio de réplicas (BRANTNER et al., 2008).

A computação em nuvem é uma evolução dos serviços e produtos de tecnologia da informação, também chamada de *utility computing* (BRANTNER et al., 2008). O objetivo da computação em nuvem assim como a *utility computing* é fornecer recursos sob demanda, tendo como principal inovação a qualidade de serviço e escalabilidade. Os usuários que se utilizam desse serviço não precisam se preocupar com possíveis falhas, pois são utilizados mecanismos de prevenção e detecção, sem afetar o desempenho da utilização das aplicações.

Para garantir que os componentes sejam utilizados de maneira escalável, Aboulnaga et al. (2009) afirmam que a virtualização auxilia a elasticidade da computação em nuvem por criar instâncias (Máquinas virtuais) através de um único meio físico. Essa ideia consiste na emulação de ambientes isolados sendo capazes de executar diferentes sistemas operacionais. Portanto, torna-se possível aproveitar melhor os recursos de *hardware*.

Tendo em vista a importância da virtualização, a computação em nuvem se utiliza desta tecnologia para garantir um melhor gerenciamento dos recursos de *hardware* e *software*. Atualmente diversos estudos são realizados no meio acadêmico, com o objetivo de difundir esta tecnologia e buscar melhores soluções de uso.

2.2.1 Principais características

A principal característica da computação em nuvem é fornecer acesso a recursos computacionais (servidores, aplicações, serviços e redes) tornando ágil o gerenciamento dos recursos (JANSEN; GRANCE, 2011).

Mell e Grance (2011) abordam a ideia de que a computação em nuvem, além de ter uma ótima capacidade de acesso aos recursos computacionais, apresenta ainda outros papeis fundamentais, que são:

Elasticidade: É caracterizada pela crescente escalabilidade de utilização dos recursos, devendo ser fornecido dinamicamente pelos provedores de serviços.

Agrupamento de recursos: Pode ser definido como a utilização de vários recursos físicos e virtuais para fornecer serviços, possibilitando uma utilização de maneira abstrata para o consumidor.

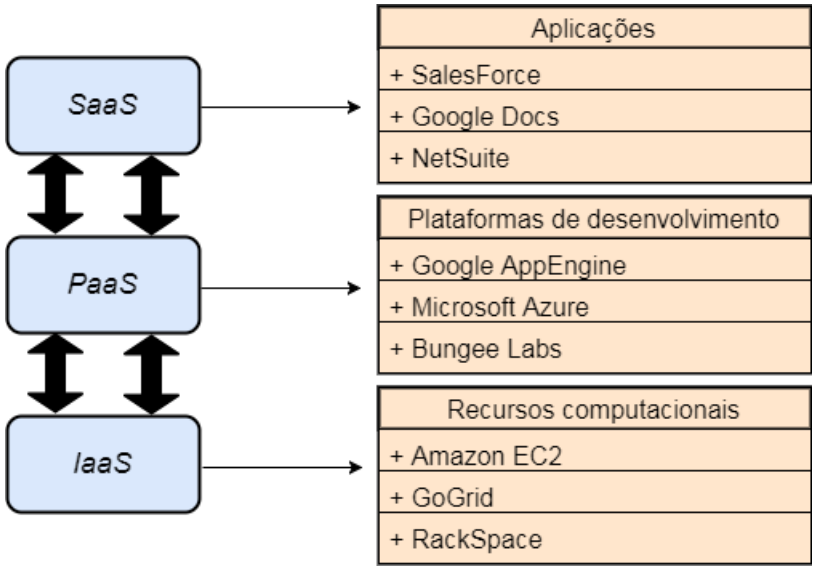
Serviços sob demanda: Conforme a necessidade for surgindo, o usuário terá a necessidade de alocar novos recursos sem que haja quaisquer tipos de intervenções e interrupções do uso do serviço.

Gerenciamento dos recursos: Característica essencial quando há a necessidade de controlar e medir o uso dos recursos que estão sendo disponibilizados para cada usuário.

Amplo Acesso: Os recursos computacionais são disponibilizados e gerenciados por meio da Internet, através de uma interface de acesso fornecida pelo provedor.

Visando o amplo crescimento da computação em nuvem, a disponibilização dos recursos foi dividida em três modelos: *SaaS*, *PaaS* e *IaaS*, como ilustrado na Figura 2.

Figura 2 – Modelos de serviço da computação em nuvem



Fonte: Elaborada pela autora

- ***Software* como um Serviço *(Software as a Service - SaaS):*** Representa o fornecimento

de aplicações sendo executadas em uma infraestrutura na nuvem e que são acessíveis através da Internet. Os usuários que se utilizam desse modelo não controlam nem gerenciam os recursos fornecidos (MELL; GRANCE, 2011). Atualmente esse serviço vem amplamente sendo utilizado e podemos destacar como exemplo de *SaaS* o Google Docs, o Gmail e ferramentas de Gestão de Relacionamento com o Cliente *(Customer Relationship Management - CRM).*

- **Plataforma como um Serviço *(Platform as a Service - PaaS):*** Nesse tipo de serviço é ofertado ao consumidor um ambiente para a criação e execução de aplicações, através do uso de linguagens de programação, bibliotecas e serviços (MELL; GRANCE, 2011). Assim como o modelo *SaaS*, no *PaaS* o consumidor não controla a infraestrutura da nuvem, mas tem o total controle sobre as aplicações que serão criadas e gerenciadas. As plataformas *Google App Engine* e *Microsoft Azure* são exemplos de *PaaS*.
- **Infraestrutura como um Serviço *(Infrastructure as a Service - IaaS):*** Este modelo apresenta como característica o fornecimento de diversos recursos, tais como o uso de servidores para armazenamento, processamento e execução de *software* em geral, incluindo sistemas operacionais e aplicativos. Como os demais modelos, no *IaaS* os usuários não administram e nem controlam a infraestrutura da nuvem subjacente, mas tem o controle dos *softwares* e aplicações que são utilizados neste ambiente (MELL; GRANCE, 2011). A *Amazon Elastic Cloud Computing (EC2)* é um exemplo de uma *IaaS*.

O uso da computação em nuvem vem sendo amplamente difundido. Na prática, uma infraestrutura de nuvem pode ser considerada o uso de *softwares* e *hadwares* em um ambiente que tenha um grande poder computacional. Os recursos de TI são fornecidos como serviços e, em geral, os usuários que consomem esses recursos não tem a necessidade de conhecer a tecnologia que será utilizada.

Devido aos benefícios proporcionados pelos modelos de serviços, a computação em nuvem passou a ser largamente adotada por diversas empresas e organizações, sendo subdividida em modelos de implantação diferenciando-se de acordo com o tipo de acesso e disponibilidade. Mell e Grance (2011) definem a seguinte divisão:

- **Nuvem Privada *(Private Cloud):*** A infraestrutura é destinada a uma única empresa ou organização e seus usuários terão acesso restrito aos recursos disponibilizados.
- **Nuvem Pública *(Public Cloud):*** Esse tipo de infraestrutura é disponibilizado para o público em geral, consistindo na ideia de pagamento do serviço de acordo com os recursos

que serão consumidos.

- **Nuvem Comunidade *(Community Cloud):*** Consiste no uso compartilhado entre empresas ou organizações que possuem interesses em comum.
- **Nuvem Híbrida *(Hybrid Cloud):*** Compreende a utilização de duas ou mais nuvens sendo usadas de acordo com as necessidades vigentes.

2.2.2 Ferramentas de gerenciamento de IaaS

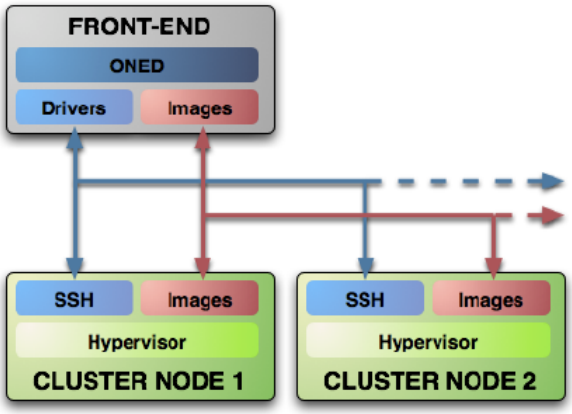
Para a criação de diferentes tipos de nuvens evidenciaram-se no mercado diversas plataformas privadas e de código aberto, todas possuindo características próprias e se destacando em diferentes aspectos. A seguir são apresentadas algumas das principais tecnologias de nuvem e sua aplicabilidade nos meios empresarial e acadêmico.

OpenNebula: *O OpenNebula* foi criado como um projeto de pesquisa em 2005 por Ignacio M. Llorente e Rubén S. Montero, mas teve sua primeira versão disponível em março de 2008 como uma solução para gerenciamento de *IaaS*. Inicialmente, o seu principal foco era a criação de nuvens privadas pois possui recursos simplificados se comparado com outras tecnologias. Por outro lado, também suporta a criação de nuvens públicas e híbridas quando se deseja combinar diferentes estruturas ou apenas realizar o gerenciamento dos recursos através da interface disponibilizada. Para a sua criação, foram utilizadas as linguagens de programação C++, Ruby e Shell (NURMI et al., 2009).

A arquitetura do *OpenNebula* é composta de vários *hosts*. O *host* chamado de *front-end* administra os recursos da nuvem, enquanto os demais *hosts* são responsáveis pela virtualização das máquinas virtuais do ambiente. O nó *front-end* executa o *OpenNebula Daemon*, cuja função é administrar toda a infraestrutura do ambiente, além de executar drivers com funcionalidades diversas. A sua comunicação com os outros nós é possível através de canais SSH (do inglês, *Secure Shell*) (SOTO, 2011).

Além do Daemon no nó *front-end*, os demais componentes de sua arquitetura são os drivers para acessar os recursos dos *clusters* e o *scheduler* para realizar o escalonamento e alocação das VMs. Na Figura 3 é demonstrada a arquitetura do *OpenNebula* e seus respectivos componentes.

Figura 3 – Arquitetura padrão de uma nuvem *OpenNebula*



Fonte: OpenNebula (2016)

O ciclo de lançamentos de *releases* do *OpenNebula* segue um padrão de a cada ano uma versão principal, fornecendo algumas atualizações menores durante esse período. Atualmente encontra-se na versão 5.0.0 lançada em junho de 2016 (OPENNEBULA, 2016).

CloudStack: O *CloudStack* foi criado em 2008 com o nome VMops fundada por Sheng Liang e apenas em 2010 foi lançado com 98% do seu código aberto, após conquistar o domínio cloud.com. Em 2011 passou a ser liderado pela *Citrix Systems* liberando o restante do seu código. Já em 2012 foi doado para a *Apache Software Foundation*[1] desenvolvendo a sua comunidade de *software* livre (HIGGINBOTTOM, 2013).

Inicialmente o *CloudStack* foi desenvolvido para gerenciar grandes redes, por possuir alta capacidade de escalabilidade e disponibilidade em sua infraestrutura. Outra grande característica do *CloudStack* é possibilitar a implantação de nuvens privadas, híbridas e públicas, sendo flexível. Além disso, disponibiliza uma interface bastante intuitiva e de fácil gerenciamento (CLOUDSTACK, 2016).

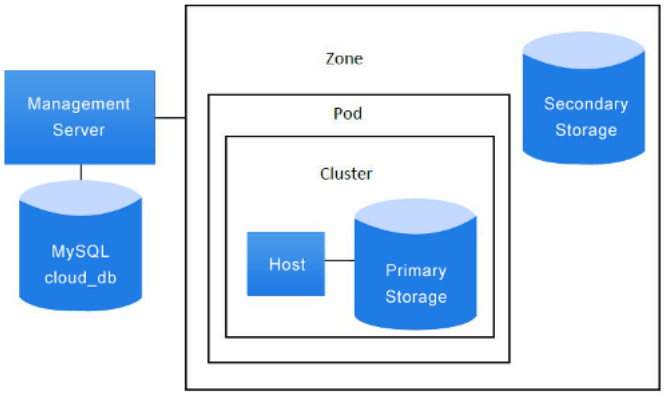
Conforme mostrado em Higginbottom (2013) e em CloudStack (2016), a arquitetura do *CloudStack* é formada pelos seguintes componentes (como ilustrado na Figura 4):

- **Região:** Consiste em um grupo de zonas de disponibilidade dentro de uma mesma área geográfica.
- **Zona de disponibilidade:** Corresponde a uma ou mais zonas por centro de dados. Contém pelo menos um *pod*, um *cluster* e uma unidade de *Storage* secundário.

[1] <https://http://www.apache.org/>.

- ***Pod*:** Entidade lógica, normalmente, um rack contendo um ou mais *clusters* e uma infraestrutura de rede.
- ***Cluster*:** Grupo de servidores *(Hosts)* idênticos executando o mesmo *software* de virtualização (Hipervisor).
- ***Storage* Primário:** É onde os discos das máquinas virtuais residem e pode ser utilizado o disco local de um *host* ou um *storage* compartilhado.
- ***Storage* Secundário:** É onde são armazenados os *templates* de máquinas virtuais, arquivos ISO e *Snapshots* e é utilizado o protocolo NFS (do inglês, *Network File System*).

Figura 4 – Arquitetura do *CloudStack*



Fonte: CloudStack (2016)

Eucalyptus: A princípio o *Eucalyptus* se iniciou como um projeto de pesquisa do departamento de Ciências da Computação, na Universidade da Califórnia, Santa Bárbara. Atualmente o *Eucalyptus* é indicado para ser aplicado em ambientes corporativos por facilitar o acesso de aplicações. O funcionamento do *Eucalyptus* é semelhante ao das nuvens disponibilizadas pela *Amazon*, diferenciando-se por se tratar de nuvens privadas (EUCALYPTUS, 2016). Foi escrito nas linguagens de programação Python, C e Java (SHARATH; BASU, 2013).

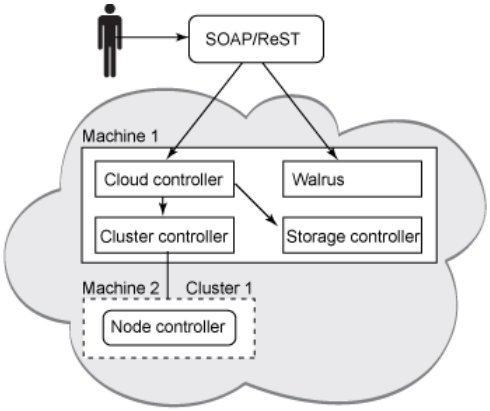
Em Nurmi et al. (2009) e em Sharath e Basu (2013) foi definida a arquitetura do *Eucalyptus* com as seguintes características:

- ***Cloud Controller*:** É o componente controlador principal que é responsável por organizar os recursos virtualizados e interagir com os demais componentes de sua infraestrutura, além de manter todas as instâncias virtuais da rede.

- ***Walrus*:** É o controlador que envia as mensagens de acesso aos dispositivos de armazenamento e armazena dados persistentes.
- ***Storage Controller*:** É o serviço de armazenamento que implementa a interface da *Amazon S3*.
- ***Node Controller:*** Controla todas as atividades das máquinas virtuais, desde a sua execução até a inspeção e encerramento das instâncias.

Na Figura 5 é ilustrada a arquitetura da plataforma *Eucalyptus* e os relacionamentos entre seus componentes.

Figura 5 – Topologia de uma instalação *Eucalyptus* em um único *cluster*



Fonte: Eucalyptus (2016)

A subseção seguinte apresenta mais detalhes sobre a plataforma de nuvem *OpenStack*, por se tratar da solução de nuvem escolhida para este trabalho. Os principais requisitos utilizados para a escolha, foram apresentados em Barros et al. (2015):

- Considerado o orquestrador de nuvem de código aberto mais utilizado no âmbito acadêmico e industrial;
- Ser base de desenvolvimento de sistemas comerciais;
- Possui ciclo de lançamentos frequentes;
- Organiza os seus serviços em módulos funcionais;
- Possui um serviço que provê modelos flexíveis de redes atendendo às necessidades de aplicações e usuários.

2.3 *OpenStack*

O *OpenStack* é uma plataforma de nuvem de código aberto que surgiu em 2010 através de uma iniciativa da *Rackspace Hosting*[2] e da NASA (LEDYAYEV; RICHTER, 2014). Sua estrutura era baseada na plataforma Nebula, da NASA, e no sistema de arquivos em nuvem da *Rackspace*. Ele foi escrito na linguagem de programação Python e atualmente implementa duas APIs de controle, a API EC2 e *Rackspace*. Fornece ainda suporte a diferentes hipervisores (Xen, KVM, HyperV, Qemu) (OPENSTACK, 2017).

Atualmente, o *OpenStack* tem uma das maiores comunidades de código aberto, que conta com o apoio de grandes organizações, como VMware, IBM, Cisco, Dell, Red Hat e Canonical, contribuindo para o seu desenvolvimento (SINGH; SINGH; PACHAURI, 2014). O *OpenStack* foi projetado para fornecer serviços em um ambiente de *IaaS* que tem por finalidade a implantação de nuvens públicas e privadas. Desde que se iniciou, seguiu um padrão de lançamentos a cada seis meses, fornecendo ótimo desempenho em suas funcionalidades.

Devido a sua natureza aberta, qualquer usuário pode implementar e criar novos componentes conforme sua necessidade. Essa integração permite a expansão de forma horizontal da infraestrutura de nuvem (AMORIM; BORGES; NACIF, 2009).

Dentre os serviços fornecidos pelo *OpenStack* estão os do tipo Core, compostos pelo *Nova, Keystone, Neutron, Glance, Swift e Cinder*, além de outros serviços essenciais como o *Horizon, Ceilometer* e o *Heat*. Em uma infraestrutura organizacional, esses são os principais elementos que devem existir para a utilização do ambiente.

Em Amorim, Borges e Nacif (2009) são definidas as seguintes características de cada serviço:

- **Nova (*Cloud Management*):** Gerencia o ciclo de vida das instâncias de máquinas virtuais. Seus principais componentes são: uma base de dados, que é usada por todos os componentes; Painel *Web*, que consiste em um componente externo que tem a função de se comunicar com a API; API, que recebe solicitações HTTP, converte comandos e se comunica com outros componentes através da fila ou HTTP; *Auth Manager*, que tem a função de gerenciar os usuários e projetos; *Object Store*, que armazena e recupera imagens; por fim, há o *Scheduler* que define as VMs que devem ser alocadas em cada *host*, gerencia os IPs *Forwarding*, pontes e VLANs, e gerencia a comunicação entre máquinas virtuais e seus hipervisores.

[2] <https://www.rackspace.com>.

- ***Keystone (Identity service)*:** Fornece seis funções básicas, como o serviço que valida as credenciais e informações dos usuários e grupos de usuários; *Resource*, que provê dados sobre projetos e domínios; *Assigntment*, que fornece dados sobre *roles* às entidades geridas pelos serviços de *Identity* e de *Resource*; *Token*, que valida e administra solicitações de autenticação; *Catalog*, que fornece o catálogo de todos os serviços, e por fim o *Policy* que tem como função fornecer um mecanismo de autorização baseado em regras.
- ***Glance (Image Service):*** Armazena e recupera imagens de disco das máquinas virtuais, através de uma API RESTful.
- ***Swift (Object Storage):*** Armazena e recupera objetos de dados não estruturados por meio de uma API RESTful. Sua arquitetura é formada por um servidor proxy com a função de conectar o resto da arquitetura *Swift*; um *ring* cuja função é mapear o nome de entidades armazenadas em disco e sua localização física; um servidor de objetos que armazena os objetos em dispositivos locais; um servidor de contas que lista todos os contêineres de cada conta, e um replicador que tem a função de identificar possíveis falhas.
- ***Cinder (Block Storage):*** Prover um *block storage* persistente para as instâncias em execução, garantindo que continue existindo mesmo após uma máquina virtual seja apagada. Em sua arquitetura possui uma base de dados, um *cinder-ap* que recebe as requisições HTTP, um *cinder-scheduler* na qual decide quais máquinas receberão um determinado bloco, um *cinder-volume* cuja função é gerenciar de forma dinâmica os dispositivos físicos, e por último um *cinder-backup* que realiza *backup* de todos os blocos.
- ***Horizon (Control Panel/Dashboard):*** Interface gráfica para todos os serviços, dinamizando o gerenciamento dos recursos da nuvem.
- ***Ceilometer (Accounting):*** Fornece o monitoramento e medição dos recursos da nuvem, facilitando tomadas de decisões de prevenção e correção quando ocorrer possíveis falhas. Entre as soluções impostas estão os agentes de recolhimento que medem o quanto cada recurso está sendo utilizado, os agentes de notificação com a função de identificar possíveis eventos, os coletores que realiza o recolhimento dos dados de cada evento e os armazena, e uma API para visualizar os dados dos eventos coletados.
- ***Heat (Orchestration):*** Mecanismo que permite uma automatização da implantação da infraestrutura, visto que gerencia o ciclo de vida do ambiente, pois quando houver a necessidade de realizar mudanças basta efetuar tais modificações em seu *template* que serão atualizadas.

A próxima seção irá descrever detalhadamente o funcionamento do serviço *Neutron*, também considerado o serviço de rede do *OpenStack*. O *Neutron* é responsável por todos os aspectos para a criação de infraestruturas de redes virtuais, por esse motivo é composto de diversos componentes que são descritos a seguir.

2.3.1 OpenStack Neutron

O serviço de rede do *OpenStack* passou por grandes mudanças desde sua idealização, visando se adequar ao aumento dos protocolos de roteamento e às novas regras de segurança. Com o passar dos anos, a fundação *OpenStack* lançou o serviço de rede *Quantum* baseado em redes definidas por *software* (SDN, do inglês *Software-Defined Network*) e em seguida foi nomeado para *Neutron* (GEBREYOHANNES, 2014).

A proposta do serviço *Neutron* é prover rede como serviço, permitindo a criação de infraestruturas de redes dinâmicas. Por meio de uma API, os demais serviços do *OpenStack* utilizam o *Neutron* para fornecer conectividade entre as VMs.

Em OpenStack (2017), são abordados os principais componentes do *Neutron*, os tipos de redes e *bridges* utilizadas para fornecer a comunicação entre os serviços e acesso à rede externa, detalhando ainda como é estabelecida a comunicação entre as VMs, conforme descrito abaixo:

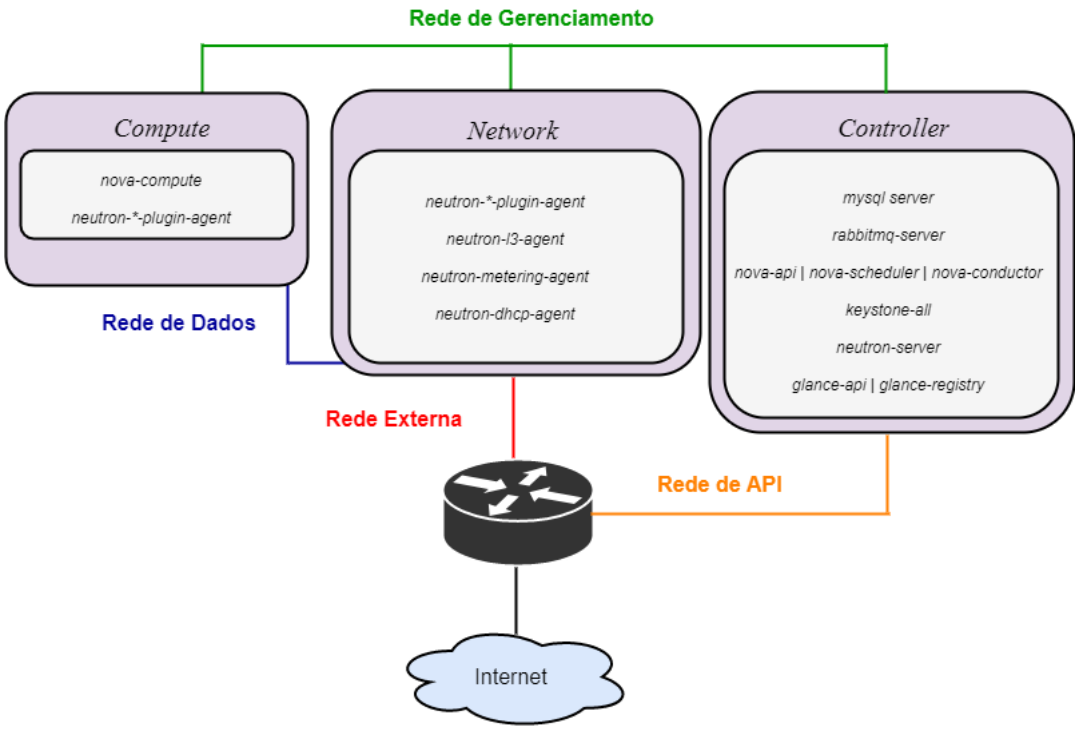
- **Servidor *Neutron* (*neutron-server* e *neutron-*-plugin*):** Serviço executado no *network node* fornecendo APIs para conexão. Ele faz uso de um servidor AMQP (do inglês, *Advanced Message Queuing Protocol*) para intermediar as mensagens entre os serviços. Atualmente o *OpenStack* suporta três implementações de AMQP: RabbitMQ, Qpid e ZeroMQ. Geralmente o servidor de mensagens utilizado é o RabbitMQ[3].
- **Agente plugin (*neutron-*-agent*):** É executado nos *compute nodes* e sua principal função é implementar serviços de redes (e.g., *neutron-linuxbridge-agent*, *neutron-openvswitch-agent*).
- **Agente L3 (*neutron-l3-agent*):** Tem a função de prover acesso à rede externa às VMs fornecendo serviço NAT.
- **Agente DHCP (*neutron-dhcp-agent*):** Implementa serviço DHCP para as VMs.

A Figura 6 apresenta os principais serviços, *plugins* e os agentes do *Neutron*, além das redes utilizadas no *OpenStack*. Já na Figura 7, são ilustradas as *bridges* utilizadas e o fluxo

[3] <https://www.rabbitmq.com/>.

de comunicação entre as VMs pertencentes ao mesmo *compute node*, entre diferentes *compute nodes* e a comunicação das VMs com a rede externa.

Figura 6 – Componentes *Neutron*



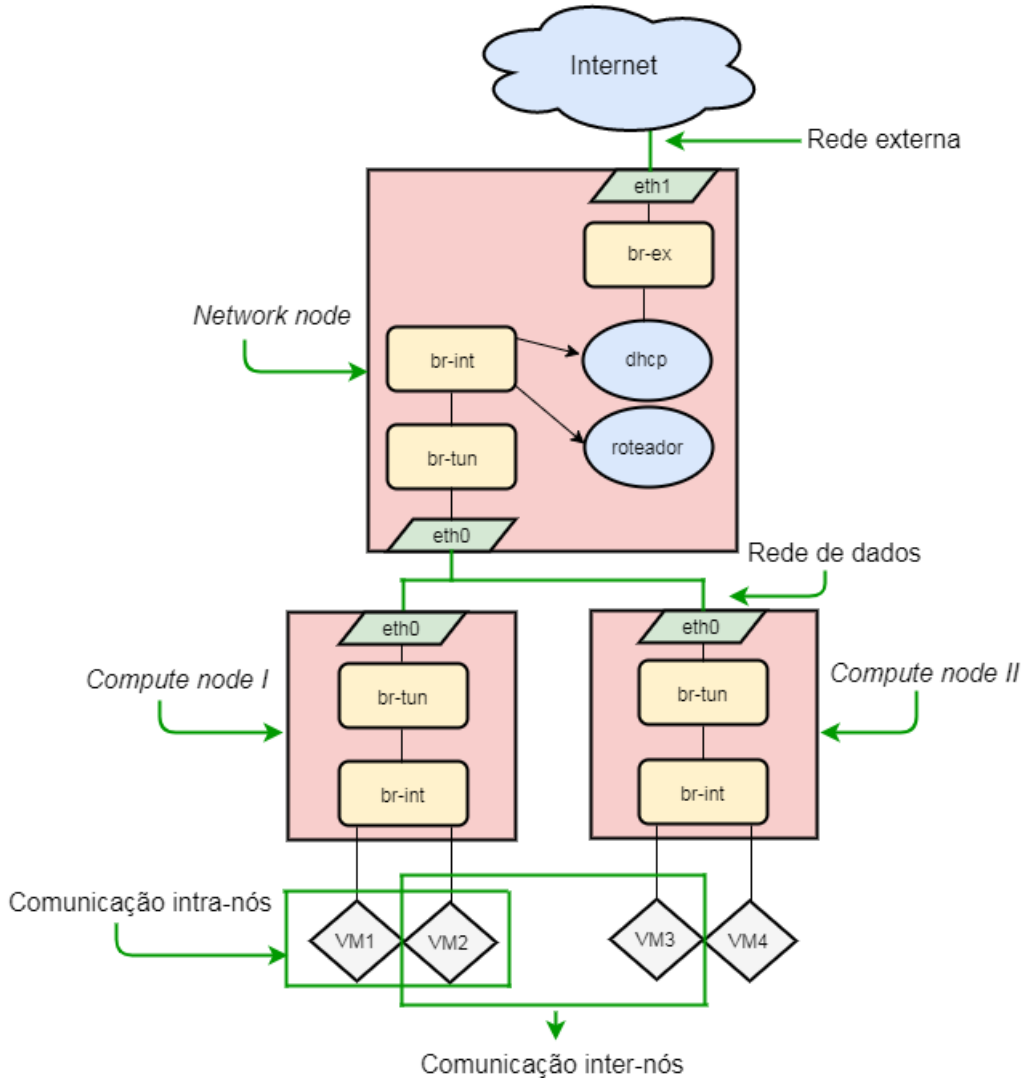
Fonte: Elaborada pela autora

As redes no *OpenStack* são dividas em quatro tipos conforme ilustrado na Figura 6:

- **Rede de gerenciamento:** Fornece conectividade entre os serviços do *OpenStack*, facilitando e isolando a comunicação.
- **Rede de dados:** Possibilita a conectividade entre VMs de acordo com a opção de configuração utilizada para a criação das redes virtuais.
- **Rede externa:** Permite que as VMs acessem a rede externa (e.g., a Internet).
- **Rede de API:** Estabelece a comunicação entre usuários e serviços do *OpenStack*.

Além dos componentes e redes apresentados, o *Neutron* utiliza elementos de redes virtualizados para garantir a comunicação entre as VMs. A Figura 7 apresenta três tipos de *bridges* gerenciadas pelo *Neutron*, utilizando o *Open vSwitch*.

Figura 7 – Fluxo de comunicação usando *Open vSwitch*



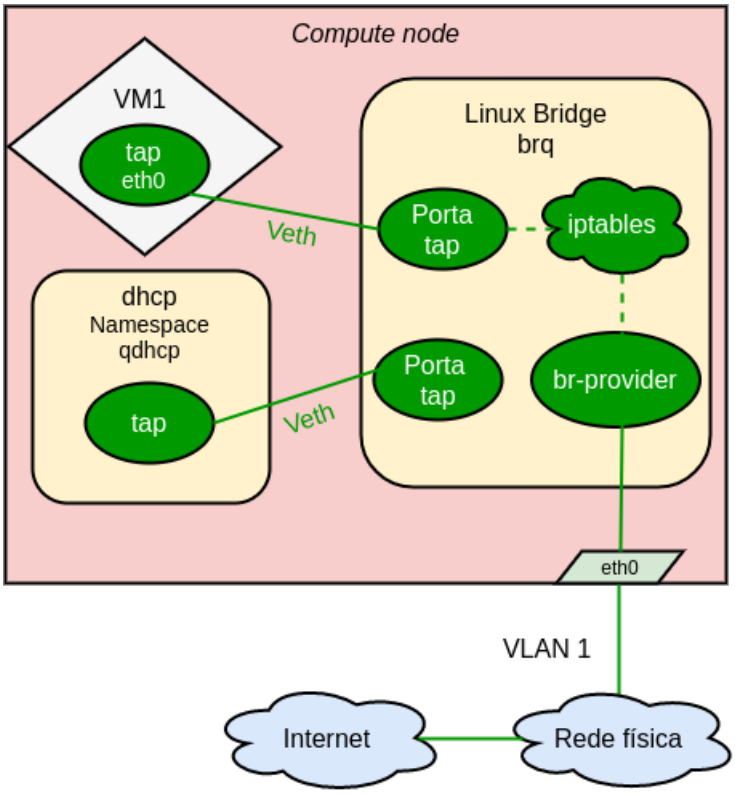
Fonte: Elaborada pela autora

- **Comunicação intra-nós (*bridge br-int*):** *Bridges* utilizadas para estabelecer a comunicação entre VMs que estão sendo executadas no mesmo *compute node*. Dessa forma, se a VM1 deseja se comunicar com a VM2, ela irá utilizar a *bridge br-int* e o fluxo é configurado dentro da tabela do *switch* virtual.
- **Comunicação inter-nós (*bridge br-tun*):** Possibilita conectividade entre VMs que estão sendo executadas em diferentes *compute nodes*. Diferente da comunicação intra-nós, na comunicação inter-nós, o tráfego passa pela *br-int*, e é encaminhado para a *br-tun*, que o encapsula usando um protocolo de tunelamento. Quando o trafégo chega no *node* de destino, ele é desencapsulado pela *br-tun* e encaminhado para a *br-int*.
- **Comunicação com a Internet (*bridge br-ex*):** É semelhante à comunicação inter-nós,

uma vez que o tráfego é enviado para a *br-int*, e em seguida, é encaminhado para a *br-tun*, sendo encapsulado com um protocolo de tunelamento e enviado através da rede de dados para o *network node*. Após ser desencapsulado no *network node*, é utilizado os serviços de roteamento do *Neutron* para encaminhar para a rede externa.

A Figura 8 mostra os componentes e o fluxo de comunicação usando o agente *Linux Bridge*. É possível observar no exemplo, que a instância VM1 reside no mesmo *compute node* que o agente DHCP. Além disso, cada instância possui uma interface tap associada para estabelecer comunicação com o agente *Linux Bridge*, que dessa forma se comunica com a rede externa.

Figura 8 – Fluxo de comunicação usando *Linux Bridge*



Fonte: Elaborada pela autora

2.3.1.1 Plugin Modular Layer 2 (ML2)

Em redes *OpenStack*, é essencial entender a variedade de tecnologias que podem ser utilizadas. O *plugin* ML2 permite definir qual tecnologia será configurada para cada instalação, seja para implementar serviços de camada 2 ou até mesmo fornecer serviços de redes adicionais, como balanceamento de carga, firewall e VPN.

A arquitetura do *plugin* ML2 é definida com base no driver que será gerenciado para

cada tipo de rede, e o mecanismo que será aplicado para acessar a rede (OPENSTACK, 2017). A Tabela 1 relaciona os drivers suportados por cada mecanismo.

Tabela 1 – Drivers e mecanismos de acesso

Drivers / Mecanismos	Flat	VLAN	VXLAN	GRE
Open vSwitch	Sim	Sim	Sim	Sim
Linux bridge	Sim	Sim	Sim	Não
SRIOV	Sim	Sim	Não	Não
MacVTap	Sim	Sim	Não	Não
L2 population	Não	Não	Sim	Sim

Fonte: OpenStack (2017)

- ***Open vSwitch:*** *Switch* virtual *multi-layer*, que pode operar nas camadas 2, 3 e 4.
- ***Linux Bridge:*** Agente de camada 2 para gerenciar *bridges* em cada *compute node*, ou em qualquer *node* que forneça serviços de camada 3.
- **SRIOV:** Funcionalidade de virtualização e compartilhamento, que permite virtualizar controladores PCIe Ethernet.
- **MacVTap:** Agente que usa os dispositivos MacVTap do kernel para implementar redes L2. Além disso, ele é usado em conjunto com o *Open vSwitch* ou *Linux Bridge* para fornecer serviços de camada 3.
- ***L2 population:*** Mecanismo com a função de otimizar o tráfego nas redes de sobreposição VXLAN e GRE, por preencher as tabelas de encaminhamento dos *switches* virtuais, diminuindo o tráfego em broadcast.

Em redes utilizando o driver Flat, todos os inquilinos compartilham o mesmo segmento de rede. Por outro lado, quando utilizado o tipo de rede VLAN, todos os inquilinos são separados logicamente conforme a necessidade. Os protocolos VXLAN e GRE também implementam separações lógicas, mas são utilizados para a criação de redes virtualizadas por sobreposição.

Todos os mecanismos de acesso listados na Tabela 1 permitem a criação de redes L2 utilizando os recursos de rede do *OpenStack*, porém o *Open vSwitch* e *Linux bridge* suportam ainda mais recursos de rede e por esse motivo são ainda mais aplicados.

Como a solução de nuvem escolhida para este trabalho trata-se do *OpenStack*, a próxima seção aborda em detalhes os protocolos de virtualização de redes suportados pela plataforma e que estão diretamente relacionados à proposta de avaliação deste estudo. As

subseções a seguir apresentam mais detalhes sobre tais tecnologias.

2.3.2 VXLAN

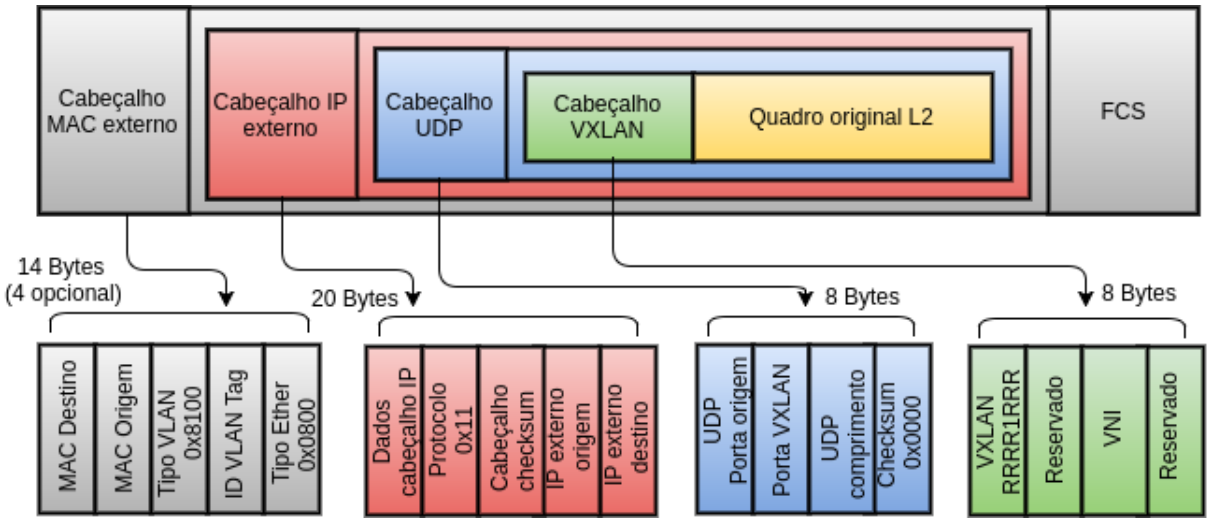
O protocolo VXLAN (*Virtual Extensible Local Area Network*) buscou resolver o problema de limitação da quantidade de VLANs suportadas em um mesmo *switch*, como as VLANs possuem um campo de identificação de 12 bits, é possível criar até 4094 redes virtuais. Embora seja um valor consideravelmente alto, não estava satisfazendo a demanda dos grandes centros de dados. Partindo dessa premissa, surgiu a padronização do protocolo VXLAN, que utiliza um identificador com 24 bits, o que equivale a 16 milhões de redes (MAHALINGAM et al., 2014).

Em VXLAN, cada rede sobreposta é chamada de segmento VXLAN, que garante a comunicação apenas entre VMs que pertencem ao mesmo segmento. Para cada segmento, é concedido um ID único, chamado de identificador de rede VXLAN (VNI, do inglês VXLAN *Network Identifier*). Logo, para identificar uma máquina, é realizada uma associação do endereço MAC com o VNI. Em resumo, quando um pacote é enviado, é atribuído um cabeçalho VXLAN com um identificador relacionado, em seguida, é adicionado o MAC e encapsulado o quadro pela extremidade do túnel, chamado VTEP (do inglês VXLAN *Tunnel Endpoint*) (SENGÉS; ALVARENGA; MOREIRA, 2017).

Para que haja comunicação entre as VMs, é preciso verificar os segmentos de origem e destino através do VTEP, que também tem o papel de certificar se existe algum mapeamento do endereço MAC. Após o envio, é feita uma validação do VNI pelo VTEP de destino, que irá identificar se existe alguma VM que possui o VNI e endereço MAC associado.

O formato do quadro VXLAN é detalhado em (MAHALINGAM et al., 2014), assim como na Figura 9.

Figura 9 – Formato do cabeçalho do quadro VXLAN



Fonte: Lifshitz (2017) - Adaptada pela autora

Cabeçalho Ethernet: Possui o endereço MAC de destino ou a especificação do próximo salto, normalmente se trata de um roteador *Layer* 3 intermediário. Já no campo Ethertype é identificado o tipo do pacote.

Cabeçalho IP: Indica as credenciais dos VTEPs de origem e destino.

Cabeçalho UDP: Apresenta as portas fornecidas pelo VTEP e um campo chamado UDP Checksum que realiza a soma de verificação UDP.

Cabeçalho VXLAN: Armazena o identificador de rede VXLAN, um campo de 24 bits e os campos reservados de 24 e 8 bits, definidos como zero na transmissão.

2.3.3 NVGRE

O protocolo de encapsulamento de roteamento genérico (GRE, do inglês *Generic Routing Encapsulation*) foi desenvolvido pela Cisco com a finalidade de encapsular múltiplos protocolos da camada de rede, criando conexões ponto a ponto.

O protocolo GRE funciona da seguinte forma: o cabeçalho GRE e o cabeçalho IP são adicionados ao pacote e encaminhado para o destino. Ao chegar no destino, ocorre o desencapsulamento, retornando ao estado original do pacote (FARINACCI et al., 1994).

Com os bons resultados no desempenho do protocolo GRE, foi idealizado um novo protocolo para virtualização de redes usando encapsulamento de roteamento genérico (NVGRE, do inglês *Network Virtualization Using Generic Routing Encapsulation*). Assim como o padrão

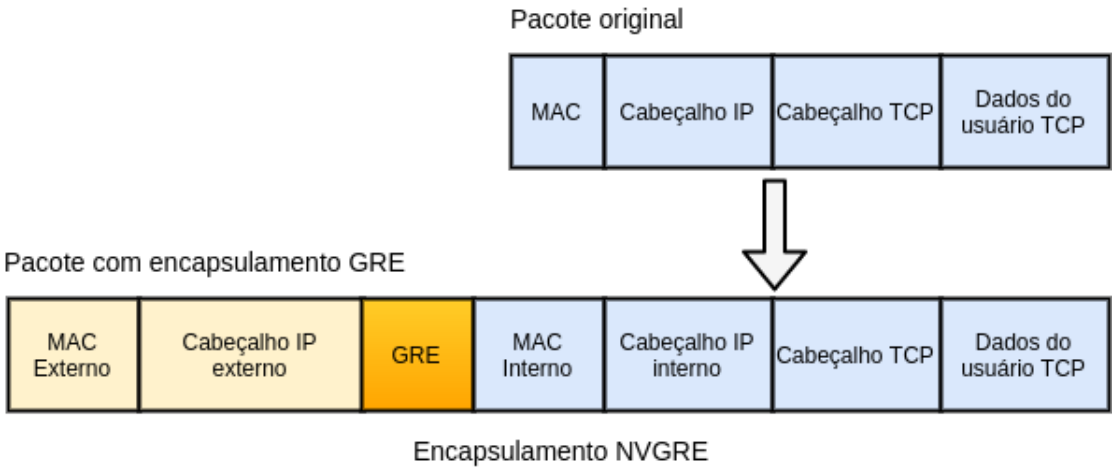
VXLAN, o protocolo NVGRE surgiu com o objetivo de atender às deficiências dos modelos arquiteturais das redes virtuais. Ele foi especificado pela RFC 7637 e desenvolvido em setembro de 2011 pelas empresas Microsoft, Arista Networks, Intel, Dell, Hewlett-Packard, Broadcom e Emulex.

Semelhante ao protocolo VXLAN, o NVGRE possui um identificador de 24 bits para cada sub-rede (VSID, do inglês *Virtual Stain Identifier*) que permite a identificação única de uma rede virtual. Com o uso do VSID, foi adaptado o cabeçalho original do GRE para que incorporasse esse identificador. Portanto, o cabeçalho do NVGRE consiste no cabeçalho original do protocolo GRE com um VSID vinculado (SENGÉS; ALVARENGA; MOREIRA, 2017).

O cabeçalho é organizado em duas extremidades: a primeira corresponde à extremidade externa, que consiste no cabeçalho Ethernet e nos prefixos de origem e destino. Esta extremidade possui ainda o cabeçalho do protocolo GRE e o cabeçalho IP com o endereço de destino correspondente, além de uma chave de 24 bits com mais 8 bits referente a um identificador de fluxo virtual (FlowID). Já a extremidade interna, também possui o cabeçalho Ethernet com os prefixos de origem e destino, mas são referentes às interfaces virtuais (SRIDHARAN et al., 2011).

Na Figura 10, é demonstrado o cabeçalho GRE com a nova proposta do NVGRE, apresentando um modelo de pacote original com a transição para o NVGRE. A proposta também foi especificada em Lifshitz (2017).

Figura 10 – Cabeçalho GRE modificado com a proposta NVGRE.



Fonte: Lifshitz (2017) - Adaptada pela autora

3 TRABALHOS RELACIONADOS

Com o desenvolvimento dos sistemas de computação em nuvem, tornou-se possível obter um melhor gerenciamento dos recursos computacionais. Atualmente existem diversas plataformas que permitem criar e gerenciar infraestruturas de nuvem, como o *OpenStack*, *OpenNebula*, *CloudStack* e o *Eucalyptus* (THOMÉ; HENTGES; GRIEBLER, 2013). Tais plataformas são soluções utilizadas no mercado e no meio acadêmico por possibilitar a implantação de diferentes tipos de nuvens.

Tendo como objetivo melhorar o desempenho das infraestruturas de nuvem, diversos estudos abordam como utilizar técnicas de virtualização de redes em diferentes contextos. Nesta seção, são descritos os principais trabalhos relacionados ao nosso estudo, através de uma análise comparativa, onde são detalhados os diferentes usos de tecnologias de virtualização de redes em ambientes de computação em nuvem.

No trabalho de Callegati et al. (2014), foi abordado o desempenho da virtualização de redes em ambientes com múltiplos inquilinos. Para avaliar o ambiente, foi implantada uma arquitetura de rede baseado em NFV considerando uma inspeção profunda dos pacotes (DPI, do inglês *Deep Packet Inspection*).

O DPI foi criado para avaliar sistemas distribuídos a partir do tráfego de rede, operando no modo de detecção ou prevenção para o desempenho da rede. Apesar dos resultados mostrarem a possibilidade de previsão de um possível ataque ou algum conteúdo que possa violar as políticas de segurança, a solução proposta apresenta algumas limitações, como a inviabilidade de obter informações do estado de aplicações não atendendo ambientes com uma grande quantidade de tráfego.

Apesar da solução proposta por Callegati et al. (2014) também ter utilizado a plataforma de nuvem *OpenStack* para criação dos cenários, ferramentas diferentes foram utilizadas para avaliar o tráfego. Enquanto eles utilizaram a ferramenta Iperf, o presente trabalho utiliza a ferramenta de geração e medição de tráfego Uperf [1]. O Uperf foi desenvolvido pelo grupo de engenharia de desempenho de aplicações da *Sun Microsystems*[2]. Embora seja uma ferramenta nova, o Uperf vem ganhando muitos apreciadores por se tratar de uma solução que suporta as implementações de vários padrões de redes.

Em (KAWASHIMA; MATSUO, 2013), os autores apresentaram um novo modelo

[1] Site do Uperf: <http://uperf.org>.

[2] <https://www.oracle.com/sun/index.html>.

de sobreposição que não utiliza os protocolos de encapsulamento habitualmente aplicados em infraestruturas de nuvem. A solução apresentada pelos autores partiu de dois métodos principais. O primeiro método consiste na reescrita do cabeçalho do quadro em *switches* OpenFlow para evitar a fragmentação e problemas de aprendizagem do MAC. Já no segundo método, é utilizado o identificador de VLANs baseado no *host*, permitindo a escalabilidade do número de redes virtuais.

Já Kawashima e Matsuo (2014) se basearam em novas perspectivas para aprimorar os estudos já consumados em (KAWASHIMA; MATSUO, 2013). O novo ambiente para a execução dos experimentos passaram de 1GbE para 40GbE, onde foram avaliados novos tamanhos de pacotes e os efeitos da segmentação.

Para avaliação do modelo proposto em (KAWASHIMA; MATSUO, 2013) e adaptado logo seguinte em (KAWASHIMA; MATSUO, 2014), foi utilizada a ferramenta Iperf para geração de fluxos TCP e UDP, comparando os resultados obtidos com as técnicas de virtualização de rede GRE e VXLAN. Assim como no nosso trabalho, os autores utilizaram o *Open vSwitch*, mas não deixaram claro no texto qual a solução de nuvem utilizada.

Propomos uma abordagem diferente do trabalho de Kawashima e Matsuo (2013) e Kawashima e Matsuo (2014). Avaliamos o desempenho dos protocolos de tunelamento VXLAN e GRE, em três cenários, em uma nuvem implantada com a plataforma *OpenStack*:

1. *Open vSwitch* com VXLAN;
2. *Open vSwitch* com GRE;
3. *Linux Bridge* com VXLAN.

Gebreyohannes (2014) apresentou uma análise de desempenho em redes *OpenStack*, na qual avaliou-se fluxos de conexão TCP e UDP, abordando o desempenho conforme o tipo de alocação das instâncias. Diferente do nosso trabalho, a técnica de virtualização de redes utilizada pelos autores foi apenas o GRE. Além disso, os autores utilizaram a ferramenta Iperf para geração e medição do tráfego.

Em síntese, o presente estudo buscou avaliar o desempenho das redes de virtualização por sobreposição na plataforma *OpenStack*, considerando a técnicas VXLAN e GRE em diferentes contextos. Assim como na maioria dos trabalhos citados, consideramos também os tipos de tráfegos TCP e UDP. Diferente de todos os trabalhos, variamos o tipo de mecanismo de comunicação e o tipo de conexão para cada experimento realizado.

A Tabela 2 apresenta as tecnologias utilizadas por cada um dos trabalhos

mencionados, mostrando um comparativo com o nosso trabalho. Entre as tecnologias, consideramos a plataforma de nuvem utilizada, o tipo de tráfego gerado (TCP e UDP), as técnicas de virtualização de redes (VXLAN e GRE), o mecanismo de acesso (*Open vSwitch* (OVS) e *Linux Bridge* (LB)), o tipo de conexão (Unidirecional (Uni.) e bidirecional (Bi.)), e se houve a variação entre *compute nodes*.

Tabela 2 – Comparação entre os trabalhos relacionados

Trabalhos relacionados	*Openstack*	TCP UDP	VXLAN GRE	LB	OVS	Uni.	Bi.	Variação nos *nodes*
(CALLEGATI et al., 2014)	X	-	-	-	X	X	-	-
(KAWASHIMA; MATSUO, 2013)	-	X	X	-	X	X	-	-
(KAWASHIMA; MATSUO, 2014)	-	X	X	-	X	X	-	-
(GEBREYOHANNES, 2014)	X	X	-	-	X	X	-	X
Este trabalho	X	X	X	X	X	X	X	X

Fonte:Elaborada pela autora

4 ANÁLISE DE DESEMPENHO

Este Capítulo apresenta detalhes do planejamento da avaliação de desempenho, bem como os cenários, parâmetros, fatores e métricas avaliados. Além disso, os resultados dos experimentos são apresentados e discutidos.

Para avaliar as tecnologias de virtualização de redes do *OpenStack*, uma nuvem privada foi implantada com a versão Mitaka do *OpenStack* no Laboratório de Sistemas e Banco de Dados (LSBD), no Campus do Pici da Universidade Federal do Ceará.

4.1 Planejamento

A instalação da plataforma *OpenStack* se deu de forma manual, seguindo o tutorial oficial[1] do *OpenStack*, com o objetivo de incorporar todas as funcionalidades fornecidas pelos seus serviços. Foram utilizadas máquinas com o sistema operacional Ubuntu 14.04 LTS de 64 bits (as configurações das máquinas são apresentadas na Tabela 3).

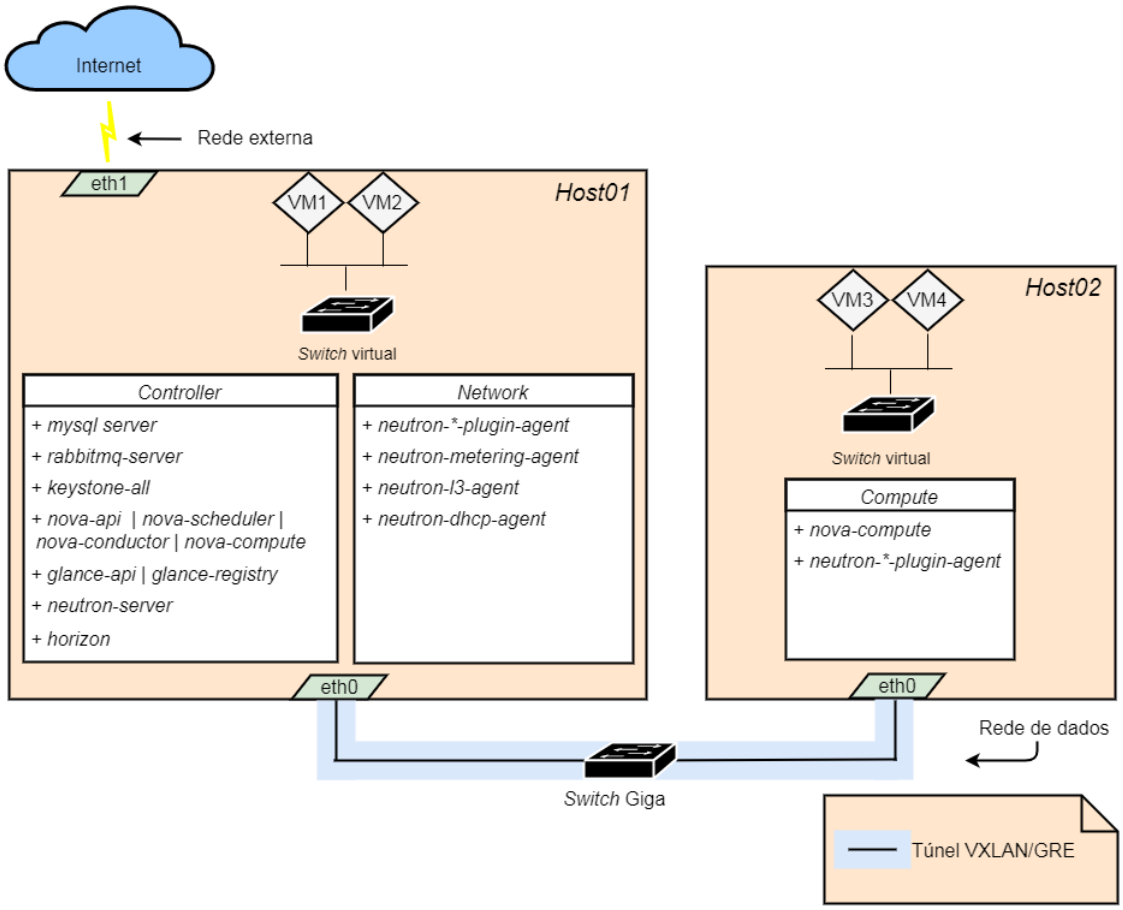
Tabela 3 – Ambiente de testes

Nodes	Memória	Processador	Disco
Controller / Network	32 GB	2 com 12 cores	100 GB
Compute	32 GB	2 com 12 cores	100 GB

Para a implantação, foram utilizados dois *nodes*, o *host01* com a função de *controller*, executando o *Keystone* para fornecer autenticação, autorização, catálogo de serviços e APIs para acesso às funcionalidades. O *Glance* para armazenar e recuperar as imagens de máquinas virtuais. O *Neutron* e o *Nova* foram configurados no *host01* e *host02*, onde parte do *Neutron* estava no *host01* para configurar e acomodar os equipamentos de rede, outra parte no *host02* com alguns agentes de rede. Já o *Nova* também foi configurado no *host01* e *host02* como *compute node* para instanciar as VMs. Além disso, no *host01* estava executando o *Horizon* para fornecer uma interface gráfica para todos os serviços. Uma ilustração da nuvem implantada pode ser vista na Figura 11, mostrando em detalhes a organização e a disposição de cada serviço.

[1] <https://docs.openstack.org/mitaka/install-guide-ubuntu/>.

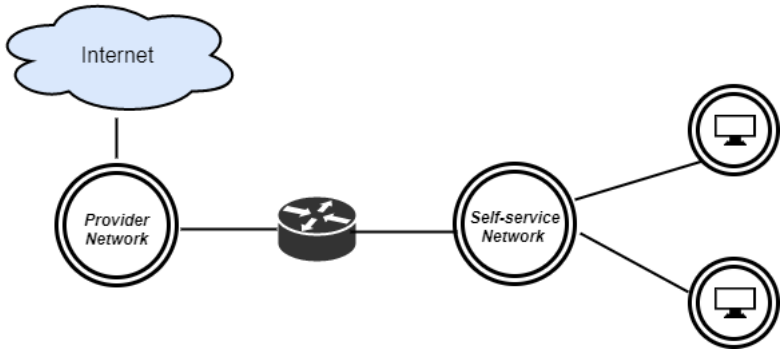
Figura 11 – Nuvem *OpenStack* implantada



Fonte: Elaborada pela autora

Em cada cenário, foram criadas duas redes virtuais, uma do tipo *provider* para acesso à rede externa e uma do tipo *self-service* como rede privada. Um roteador virtual foi adicionado para fazer a conexão entre as redes externas e internas (conforme ilustrado na Figura 12) e um *floating* IP foi atribuído a uma instância em cada rede privada criada, para que a rede externa possa se comunicar com a instância. A quantidade de instâncias variou entre 2, 4 e 8, onde elas foram alocadas no mesmo *compute node* e em diferentes *compute nodes*.

Figura 12 – Abstração da topologia proposta



Fonte: Elaborada pela autora

As configurações das placas de rede do sistema operacional são detalhadas na Tabela 4, demostrando o tamanho do MTU (do inglês, *Maximum Transmission Unit*) e o tamanho do buffer de entrada e saída.

Tabela 4 – Configurações das placas de rede

Configurações	*Nodes*	Instâncias
MTU	1500 Bytes	1500 Bytes
Buffer entrada / saída	255 Bytes	255 Bytes

4.2 Experimentos

Três ambientes foram configurados, se diferenciando de acordo com o mecanismo de acesso e o protocolo de virtualização de redes utilizados: (i) *Open vSwitch* com VXLAN, (ii) *Open vSwitch* com GRE, (iii) *Linux Bridge* com VXLAN. De acordo com a documentação do *OpenStack*, o agente *Linux Bridge* não suporta redes com o GRE, e por isso cenários com tais tecnologias não foram utilizados. As configurações necessárias para cada implantação está disponível no repositório do GitHub[2].

A Tabela 5 apresenta um sumário dos experimentos, detalhando as métricas, fatores e níveis avaliados para cada ambiente criado.

[2] <https://github.com/rose36/RedesVirtuaisOpenstack>

Tabela 5 – Sumário dos experimentos

Fatores	Número de instâncias, alocação das instâncias, mecanismo de comunicação, tecnologia de virtualização, tipo de tráfego e tipo de conexão.
Níveis	3 variações no número de instâncias (2, 4 e 8), 2 tipos de alocação (mesmo *compute node* e diferentes *compute nodes*), 2 mecanismos (*Open vSwitch* e *Linux Bridge*), 2 técnicas (VXLAN e GRE), 2 tipos de fluxos (TCP e UDP) e 2 conexões (Unidirecional e bidirecional).
Projeto	O experimento foi repetido 30 vezes para a combinação de níveis dos fatores, exceto para a configuração *Linux Bridge* com GRE que não foi avaliada.
Resposta	Vazão TCP e UDP, latência, jitter e taxa de perda de pacotes.

A criação de cada cenário foi feita de maneira automatizada usando um script[3] desenvolvido na linguagem Shell Script, que pode ser executado para a criação de redes, roteadores, adicionar interfaces aos roteadores e para a criação de instâncias. Outro script foi criado para avaliar o ambiente por meio da ferramenta de geração e medição de tráfego Uperf. Algumas métricas foram avaliadas através da execução de um ping durante o andamento do experimento, capturando o valor da latência e o jitter, além da taxa de perda de pacotes. Foi considerada ainda a vazão após a execução de cada experimento, coletada diretamente da saída do Uperf.

É importante destacar que durante a execução dos experimentos entre *compute nodes* diferentes, utilizando UDP, alguns problemas de instabilidade na rede foram detectados. Tais problemas impediram a execução dos experimentos, e por isso foi preciso planejar uma nova maneira de avaliar o cenário mencionado. A seção dos resultados apresenta mais detalhes sobre o problema.

4.3 Avaliação

O script[4] principal foi desenvolvido para automatizar a realização dos experimentos, criando conexões entre o *master*, que possui a função de servidor, e os *slaves*, que fazem o papel de clientes. Foram criados perfis para estabelecer conexões unidirecionais e bidirecionais entre as VMs. No Uperf, cada perfil é organizado da seguinte forma:

Grupos: É caracterizado por um conjunto de threads ou processos, cuja função é executar as transações que fazem parte desse grupo. Cada perfil pode ter vários grupos.

Transações: Corresponde a uma sequência de operações executadas como uma

[3] <https://github.com/rose36/RedesVirtuaisOpenstack/blob/master/scripts/criacao-automatizada-openstack.sh>
[4] <https://github.com/rose36/RedesVirtuaisOpenstack/blob/master/scripts/uperf.sh>

única unidade lógica de trabalho. Para cada transação, é preciso definir o tempo de execução e a quantidade de repetições para cada ação.

Flowops: Cada transação é formada por *flowops* que indicam as operações que serão realizadas, tais como: *connect, accept , disconnect, read, write, redv, sendto, sendfilev, think,* e NOP.

Para executar o script, é preciso passar por parâmetro a quantidade de vezes que será realizado o experimento, se será unidirecional ou bidirecional, o IP do *host* remoto, o protocolo que será utilizado, o tamanho do buffer, a duração de cada transação em segundos e o tamanho da mensagem. Após testar diferentes valores para os parâmetros a fim de maximizar a vazão, os seguintes valores foram fixados para todos os experimentos: (i) 30 repetições, (ii) 10 threads, (iii) 256k o tamanho do buffer, (iv) 30 segundos de duração, (v) 56k o tamanho da mensagem. Outros parâmetros sofreram variação, tais como o tipo de execução (unidirecional ou bidirecional), o IP do *host* remoto e o protocolo que seria utilizado (TCP ou UDP).

Em execuções unidirecionais, o *slave* realiza a operação de escrita enquanto o *master* de leitura. Já em execuções bidirecionais, o *slave* e o *master* realizam ambas operações. Abaixo é demonstrado um exemplo de perfil criado para a realização dos experimentos. Os *templates* de cada perfil criado também pode ser vistos no repositório do GitHub.

Código-fonte 1 – Exemplo de perfil do Uperf

```
1  <?xml version="1.0"?>
2  <profile name="Cliente Unidirecional">
3    <group nthreads="10" >
4          <transaction iterations="1" >
5              <flowop type="accept" options="remotehost=10.0.0.3 protocol=tcp wndsz=256k
                   tcp_nodelay"/>
6          </transaction>
7          <transaction duration="30" >
8              <flowop type="write" options="size=56k "/>
9          </transaction>
10         <transaction iterations="1" >
11             <flowop type="disconnect" />
12         </transaction>
13   </group>
14 </profile>
```

O Código-fonte 1 apresenta o perfil criado para clientes unidirecionais, composto de um grupo e uma transação de 30 segundos. A linha 5, possui o IP do *host* remoto, o tipo de fluxo que será utilizado e o tamanho do buffer. O tipo de operação e o tamanho do pacote são

definidos na linha 8.

4.4 Resultados

Para cada avaliação realizada, buscamos determinar a taxa de transferência em fluxos TCP e UDP considerando a quantidade de conexões simultâneas e a alocação das máquinas virtuais. Outro fator analisado foi o desempenho apresentado entre as técnicas de virtualização em execuções unidirecionais e bidirecionais. Além disso, avaliou-se o comportamento do tráfego através da variação da localização das instâncias, visando comparar o desempenho apresentado quando as instâncias são alocadas no mesmo *compute node* e entre diferentes *compute nodes*.

As subseções a seguir exibem tabelas e gráficos gerados para a análise comparativa dos resultados. Em experimentos realizados entre instâncias do mesmo *compute node*, as taxas de transferência estão em Gigabit por segundo (Gb/s), já entre *compute nodes* diferentes em Megabit por segundo (Mb/s). Para criar os gráficos, a média foi calculada de acordo com o número de amostras (30 amostras com 2 VMs, 60 amostras com 4 VMs e 120 amostras com 8 VMs), visto que a partir de 4 VMs passou a ter mais de uma conexão gerando tráfego simultaneamente. Além disso, foi calculado também o intervalo de confiança utilizando um nível de confiança de 95%.

4.4.1 Vazão TCP e UDP

Em todos os experimentos realizados entre instâncias pertencentes ao mesmo *compute node*, as taxas de transferência em conexões UDP foram maiores quando comparadas com conexões TCP. Os resultados obtidos foram ainda mais significativos em experimentos bidirecionais.

As Tabelas 6 e 7 mostram um comparativo entre a média da vazão TCP e UDP, levando em consideração a variação na quantidade de conexões simultâneas. É possível observar que em conexões unidirecionais a única tecnologia que apresentou maior diferença na média entre fluxos TCP e o UDP foi *Open vSwitch* com VXLAN, quando utilizou 8 instâncias. A média UDP obtida foi em torno de 20% a mais quando se comparada com a média TCP.

Em experimentos bidirecionais, a maior diferença entre as médias TCP e UDP também foi ocasionada utilizando *Open vSwitch* com VXLAN. Diferente dos experimentos de conexões unidirecionais, os maiores resultados foram em testes com 2 instâncias, onde a diferença foi de 43% a mais.

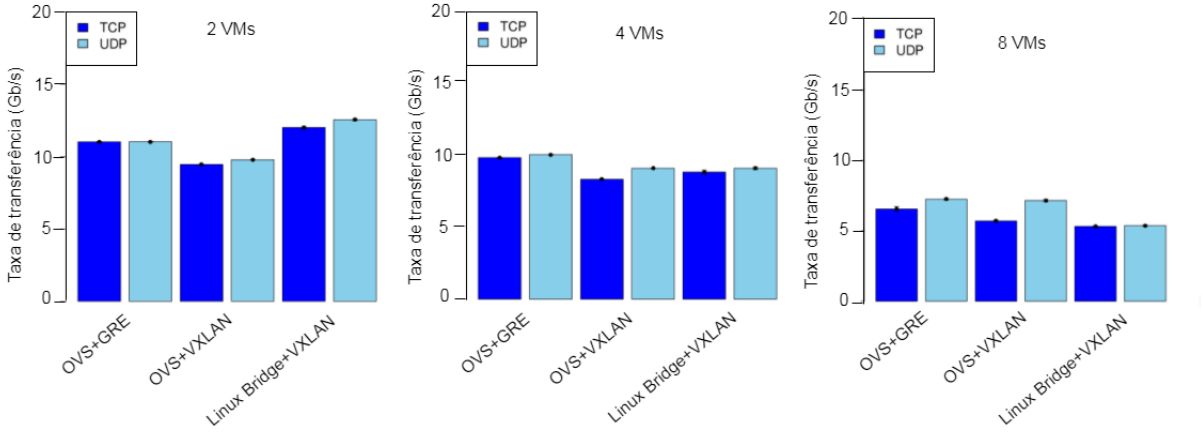
Os gráficos das Figuras 13, 14, 15 e 16 apresentam uma visão geral do desempenho de cada tecnologia para melhor compreensão dos resultados.

Tabela 6 – Vazão de conexões unidirecionais (mesmo *compute node*) em Gb/s

Quantidade de VMs	*Open vSwitch* com GRE		*Open vSwitch* com VXLAN		*Linux Bridge* com VXLAN	
	TCP	UDP	TCP	UDP	TCP	UDP
2	11.0133 ± 0.0299	11.021 ± 0.0286	9.447 ± 0.0227	9.776 ± 0.0266	12.007 ± 0.0915	12.542 ± 0.0798
4	9.742 ± 0.0303	9.9478 ± 0.0186	8.2487 ± 0.02	9.0023 ± 0.0168	8.7477 ± 0.1321	9.003 ± 0.0838
8	6.5541 ± 0.1745	7.2505 ± 0.076	5.7022 ± 0.0641	7.1588 ± 0.091	5.3192 ± 0.0477	5.3753 ± 0.0621

Nota – As células destacadas correspondem às tecnologias que apresentaram maiores médias na taxa de vazão em fluxos TCP e UDP de acordo com a quantidade de VMs.

Figura 13 – Vazão de conexões unidirecionais (mesmo *compute node*) em Gb/s



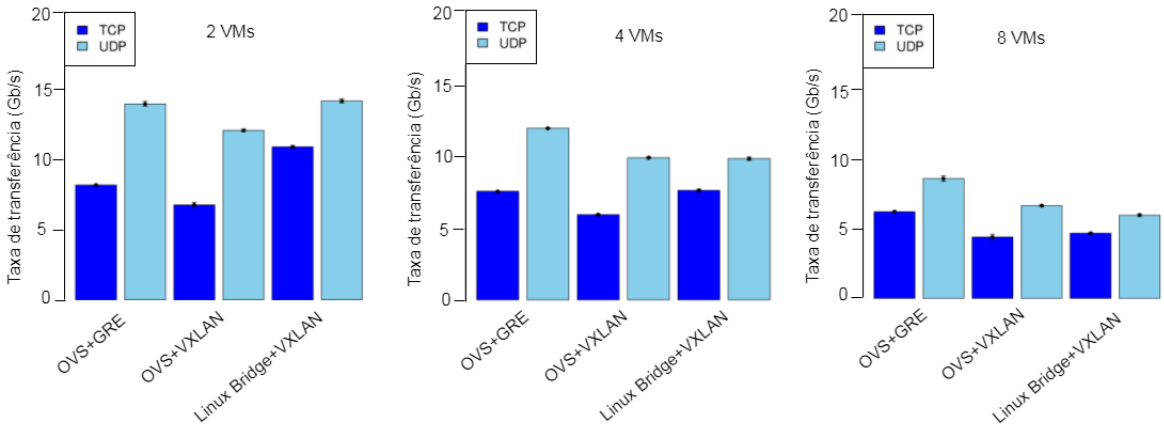
Fonte: Elaborada pela autora

Tabela 7 – Vazão de conexões bidirecionais (mesmo *compute node*) em Gb/s

Quantidade de VMs	*Open vSwitch* com GRE		*Open vSwitch* com VXLAN		*Linux Bridge* com VXLAN	
	TCP	UDP	TCP	UDP	TCP	UDP
2	8.264 ± 0.0199	14.105 ± 0.1925	6.8373 ± 0.0175	12.204 ± 0.0976	11.0157 ± 0.0976	14.3157 ± 0.1527
4	7.6462 ± 0.0089	12.0858 ± 0.037	5.9933 ± 0.0315	10.0077 ± 0.0637	7.7103 ± 0.0864	9.9475 ± 0.1253
8	6.2368 ± 0.0715	8.6235 ± 0.203	4.4148 ± 0.0315	6.6709 ± 0.0663	4.6902 ± 0.0501	5.9851 ± 0.0903

Nota – As células destacadas correspondem às tecnologias que apresentaram maiores médias na taxa de vazão em fluxos TCP e UDP de acordo com a quantidade de VMs.

Figura 14 – Vazão de conexões bidirecionais (mesmo *compute node*) em Gb/s



Fonte: Elaborada pela autora

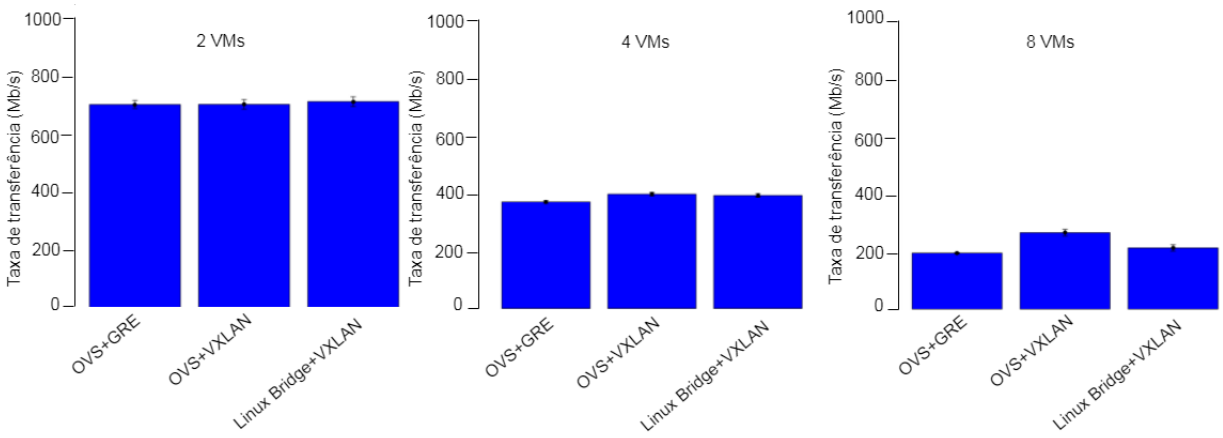
As Tabelas 8 e 9 apresentam o comportamento do tráfego apenas com fluxos TCP entre *nodes* diferentes. É possível observar que não houve grandes diferenças entre conexões unidirecionais e bidirecionais, e os resultados atingidos foram semelhantes. O uso de *Open vSwitch* com VXLAN obteve maiores taxas de vazão na maioria dos resultados, e também demonstrou melhor desempenho quando utilizado um maior número de instâncias.

Tabela 8 – Vazão de conexões unidirecionais TCP (diferentes *compute nodes*) em Mb/s

Quantidade de VMs	*Open vSwitch* com GRE	*Open vSwitch* com VXLAN	*Linux Bridge* com VXLAN
2	694.568 ± 14.57	694.996 ± 17.2087	704.599 ± 6.1493
4	374.5775 ± 5.0268	401.7392 ± 6.9796	397.327 ± 3.7435
8	193.4045 ± 2.2117	263.6853 ± 11.2148	211.1464 ± 5.3286

Nota – As células destacadas correspondem às tecnologias que apresentaram maiores médias na taxa de vazão em fluxos TCP de acordo com a quantidade de VMs.

Figura 15 – Vazão de conexões unidirecionais (diferentes *compute nodes*) em Mb/s



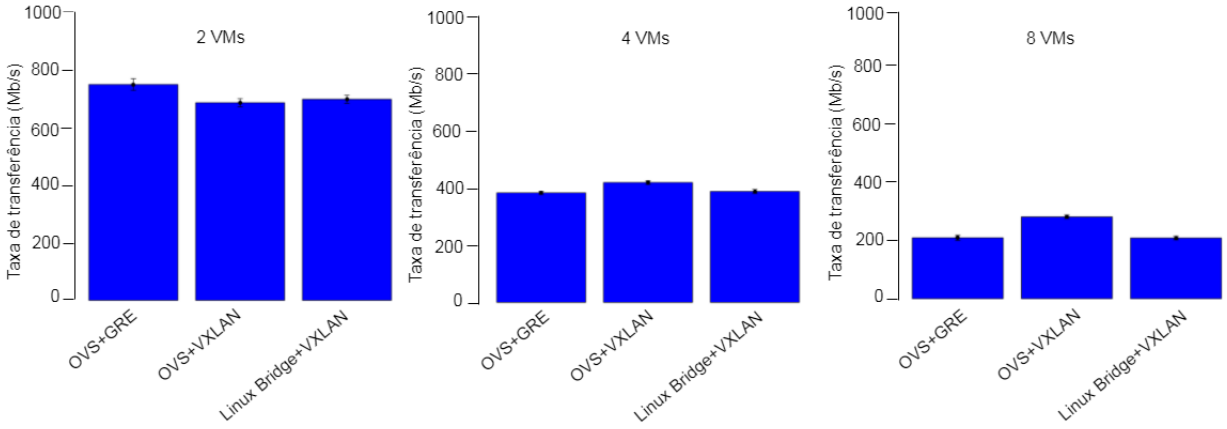
Fonte: Elaborada pela autora

Tabela 9 – Vazão de conexões bidirecionais TCP (diferentes *compute nodes*) em Mb/s

Quantidade de VMs	Open vSwitch com GRE	Open vSwitch com VXLAN	Linux Bridge com VXLAN
2	753.245 ± 20.9642	689.63 ± 15.0131	701.704 ± 9.6395
4	384.4295 ± 5.5972	420.5452 ± 6.8097	389.4242 ± 2.9332
8	208.7302 ± 8.9769	280.879 ± 6.4303	208.2483 ± 2.5494

Nota – As células destacadas correspondem às tecnologias que apresentaram maiores médias na taxa de vazão em fluxos TCP de acordo com a quantidade de VMs.

Figura 16 – Vazão de conexões bidirecionais (diferentes *compute nodes*) em Mb/s



Fonte: Elaborada pela autora

4.4.2 Latência e jitter

O intervalo de confiança da latência e jitter são apresentados nas Tabelas 10, 11, 12 e 13. Em conexões utilizando o protocolo TCP, foram apresentados todos os resultados para cada variação na quantidade de instâncias, levando em consideração ainda como estão alocadas. Já para as conexões utilizando o UDP, apenas os resultados dos experimentos entre instâncias alocadas no mesmo *compute node* são apresentados.

Os cenários utilizando *Linux Bridge* com VXLAN obtiveram menores taxas de

latência em conexões unidirecionais, sendo superior às demais configurações apenas quando 4 e 8 instâncias são utilizadas em diferentes *nodes* com tráfego TCP, e com 4 e 8 instâncias no mesmo *node* com fluxos UDP.

Em conexões bidirecionais usando tráfego TCP, foram alcançados resultados equivalentes aos de conexões unidirecionais, a única diferença identificada foi em experimentos com 8 instâncias no mesmo *compute node*. A maioria dos resultados em conexões bidirecionais UDP, o uso de *Linux Bridge* com VXLAN exibiu menores taxas de latência.

Na maioria dos resultados, a variação do atraso teve menores médias em experimentos utilizando *Open vSwitch* com GRE, tanto em conexões unidirecionais como bidirecionais.

Tabela 10 – Latência de conexões unidirecionais em milissegundos (ms)

Alocação das VMs	*Open vSwitch* com GRE			*Open vSwitch* com VXLAN			*Linux Bridge* com VXLAN		
	2 VMs	4 VMs	8 VMs	2 VMs	4 VMs	8 VMs	2 VMs	4 VMs	8 VMs
Mesmo *compute node* TCP	0.9178 ± 0.0142	1.0763 ± 0.0071	1.8218 ± 0.0638	1.1113 ± 0.0143	1.3083 ± 0.017	2.0093 ± 0.0376	0.3323 ± 0.0271	0.9713 ± 0.0506	1.6069 ± 0.0536
Mesmo *compute node* UDP	7.8519 ± 0.131	8.7554 ± 0.0862	14.7682 ± 0.3921	8.724 ± 0.1453	9.5899 ± 0.1135	13.7591 ± 0.2746	6.513 ± 0.2602	10.4815 ± 0.5431	20.7958 ± 0.5149
Diferentes *compute nodes* TCP	6.2513 ± 0.2421	8.7428 ± 0.2138	10.4028 ± 0.1667	8.1304 ± 0.2288	7.5 ± 0.1509	8.4775 ± 0.2161	5.3223 ± 0.2254	8.7468 ± 0.2927	11.4284 ± 0.3448
Diferentes *compute nodes* UDP	N.A.	N.A.	N.A.	N.A.	N.A.	N.A.	N.A.	N.A.	N.A.

Nota – As células destacadas correspondem às tecnologias que apresentaram melhores resultados.

Tabela 11 – Latência de conexões bidirecionais em milissegundos (ms)

Alocação das VMs	*Open vSwitch* com GRE			*Open vSwitch* com VXLAN			*Linux Bridge* com VXLAN		
	2 VMs	4 VMs	8 VMs	2 VMs	4 VMs	8 VMs	2 VMs	4 VMs	8 VMs
Mesmo *compute node* TCP	0.6511 ± 0.0144	0.6928 ± 0.0124	0.888 ± 0.0216	0.7872 ± 0.0246	0.9316 ± 0.0208	1.3067 ± 0.0269	0.3875 ± 0.0221	0.5022 ± 0.0301	0.933 ± 0.0389
Mesmo *compute node* UDP	0.3935 ± 0.012	0.4555 ± 0.0085	0.6123 ± 0.0279	0.4469 ± 0.0146	0.5662 ± 0.0204	0.9009 ± 0.028	0.2238 ± 0.0186	0.3955 ± 0.0277	0.814 ± 0.0369
Diferentes *compute nodes* TCP	7.7585 ± 0.1276	6.4557 ± 0.1637	6.6535 ± 0.1757	6.3518 ± 0.2547	6.4258 ± 0.1979	6.7855 ± 0.1386	5.272 ± 0.1235	7.0729 ± 0.1425	9.3651 ± 0.192
Diferentes *compute nodes* UDP	N.A.	N.A.	N.A.	N.A.	N.A.	N.A.	N.A.	N.A.	N.A.

Nota – As células destacadas correspondem às tecnologias que apresentaram melhores resultados.

Tabela 12 – Jitter de conexões unidirecionais em milissegundos (ms)

Alocação das VMs	*Open vSwitch* com GRE			*Open vSwitch* com VXLAN			*Linux Bridge* com VXLAN		
	2 VMs	4 VMs	8 VMs	2 VMs	4 VMs	8 VMs	2 VMs	4 VMs	8 VMs
Mesmo *compute node* TCP	0.22 ± 0.025	0.225 ± 0.0065	1.0593 ± 0.0484	0.2475 ± 0.0126	0.4096 ± 0.0184	1.0779 ± 0.0352	0.2868 ± 0.0696	0.6527 ± 0.0584	1.0805 ± 0.0551
Mesmo *compute node* UDP	2.8533 ± 0.074	3.2817 ± 0.0462	9.7699 ± 0.3832	3.3205 ± 0.0608	3.618 ± 0.0519	8.2609 ± 0.3423	3.6121 ± 0.3688	7.304 ± 0.7593	13.9058 ± 0.3883
Diferentes *compute nodes* TCP	2.7474 ± 0.1135	3.3703 ± 0.1191	4.0191 ± 0.1008	3.5734 ± 0.1658	2.849 ± 0.1345	3.3655 ± 0.1259	2.2169 ± 0.1351	3.2027 ± 0.1895	3.301 ± 0.11
Diferentes *compute nodes* UDP	N.A.	N.A.	N.A.	N.A.	N.A.	N.A.	N.A.	N.A.	N.A.

Nota – As células destacadas correspondem às tecnologias que apresentaram melhores resultados.

Tabela 13 – Jitter de conexões bidirecionais em milissegundos (ms)

Alocação das VMs	*Open vSwitch* com GRE			*Open vSwitch* com VXLAN			*Linux Bridge* com VXLAN		
	2 VMs	4 VMs	8 VMs	2 VMs	4 VMs	8 VMs	2 VMs	4 VMs	8 VMs
Mesmo *compute node* TCP	0.2347 ± 0.014	0.248 ± 0.0113	0.5119 ± 0.0225	0.2895 ± 0.0154	0.4609 ± 0.0222	0.7951 ± 0.0345	0.2732 ± 0.061	0.3423 ± 0.0246	0.8082 ± 0.0797
Mesmo *compute node* UDP	0.1411 ± 0.0089	0.1732 ± 0.0306	0.3706 ± 0.0291	0.1982 ± 0.0361	0.2795 ± 0.0268	0.6508 ± 0.0537	0.2268 ± 0.0776	0.2488 ± 0.0214	0.6804 ± 0.0397
Diferentes *compute nodes* TCP	2.6975 ± 0.0596	2.9157 ± 0.0783	2.993 ± 0.0864	2.9394 ± 0.0756	2.9363 ± 0.1081	3.1021 ± 0.0818	2.1992 ± 0.0848	2.5045 ± 0.0888	2.7939 ± 0.0918
Diferentes *compute nodes* UDP	N.A.	N.A.	N.A.	N.A.	N.A.	N.A.	N.A.	N.A.	N.A.

Nota – As células destacadas correspondem às tecnologias que apresentaram melhores resultados.

4.4.3 Taxa de perda de pacotes UDP

As maiores taxas de perda de pacotes no mesmo *compute node* ocorreram nos experimentos unidirecionais, onde cenários utilizando *Linux Bridge* com VXLAN obtiveram maiores taxas. Os resultados mais significativos foram em experimentos com 8 instâncias, tendo aproximadamente 6% de pacotes perdidos. Tal valor representa 93% a mais de perda em experimentos usando *Open vSwitch* com VXLAN, e 84% a mais se comparado a *Open vSwitch* com GRE. Em experimentos bidirecionais, praticamente não teve perda de pacotes, apenas quando foi utilizado *Linux Bridge* com VXLAN com 8 instâncias (valores demonstrados na Tabela 14).

Tabela 14 – Taxa de perda de pacotes UDP (mesmo *compute node*)

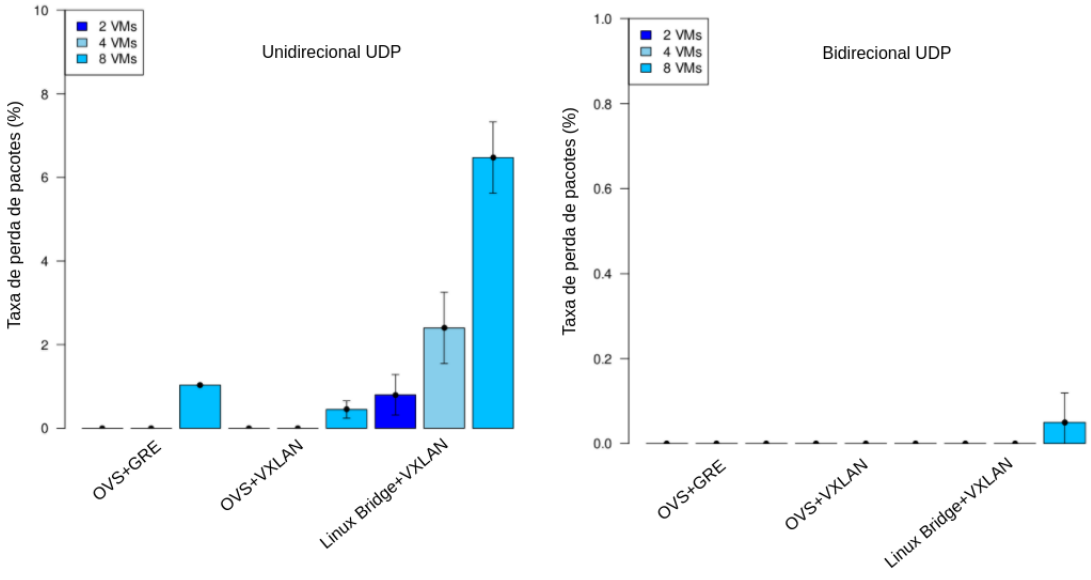
Tipo de conexão	*Open vSwitch* com GRE			*Open vSwitch* com VXLAN			*Linux Bridge* com VXLAN		
	2 VMs	4 VMs	8 VMs	2 VMs	4 VMs	8 VMs	2 VMs	4 VMs	8 VMs
Unidirecional UDP	*	*	1.0333 ± 0.3403	*	*	0.45 ± 0.2068	0.8 ± 0.505	2.4 ± 0.869	6.475 ± 0.8629
Bidirecional UDP	*	*	*	*	*	*	*	*	0.05 ± 0.0697

Nota – As células destacadas correspondem às tecnologias que apresentaram menores taxas de perda de pacotes.

Nota – As células que apresentam um asterisco (*) representa às tecnologias que não obtiveram perda de pacotes.

Os gráficos da Figura 17 representam um comparativo entre as taxas de perda de pacotes em experimentos unidirecionais e bidirecionais, onde os resultados foram agregados com base na quantidade de instâncias e na tecnologia utilizada.

Figura 17 – Taxa de perda de pacotes em conexões unidirecionais e bidirecionais UDP (mesmo *compute node*)



Fonte: Elaborada pela autora

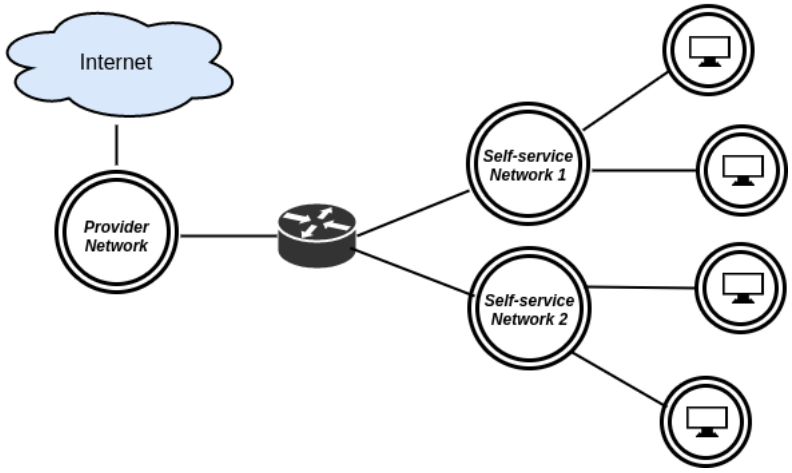
Foram detectados alguns problemas durante os experimentos utilizando UDP entre *compute nodes* diferentes, e tais problemas são retratados na seção seguinte, onde são detalhadas as dificuldades encontradas, as medidas tomadas e os resultados obtidos.

4.4.4 UDP entre compute nodes diferentes

A ideia inicial era realizar os mesmos número e tipos de experimentos para cada configuração, levando em consideração a alocação das VMs. No entanto, ao realizar experimentos gerando fluxos UDP entre *compute nodes* diferentes, foi identificada uma instabilidade na rede a ponto de impedir a comunicação entre as instâncias, o que inviabilizou a execução dos experimentos. Após pesquisar na literatura pelo problema ocorrido, constatou-se que o experimento com UDP estava gerando tráfego como um ataque de negação de serviço, conhecido como UDP *Flood Attack*, podendo ser iniciado quando se envia um grande número de pacotes UDP para portas aleatórias de um determinado *host* remoto.

Com a descoberta, um novo experimento foi realizado para investigar se o problema afetava também redes virtuais de diferentes clientes do *OpenStack*, que deveriam ser isoladas. O novo cenário possuía duas redes privadas com instâncias alocadas entre *compute nodes* diferentes (como ilustrado na Figura 18). O problema persistiu mesmo utilizando apenas uma conexão para cada rede privada. Constatou-se que quando um dos clientes gera muito tráfego UDP, é possível congestionar e criar instabilidade nos *compute nodes* envolvidos na comunicação, a ponto de afetar outros clientes.

Figura 18 – Cenário com duas redes do tipo *self-service*



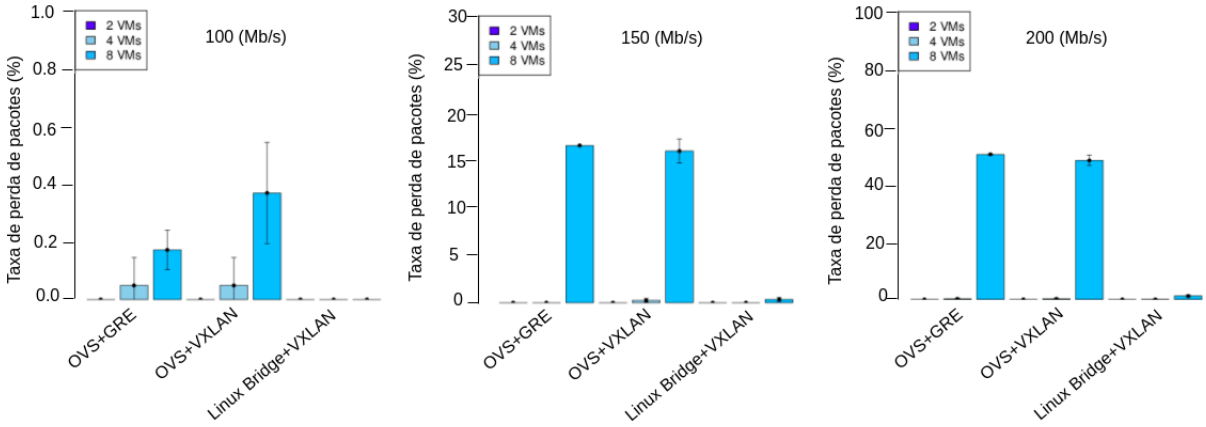
Fonte: Elaborada pela autora

Diante do problema detectado, um novo experimento foi planejado com UDP, mas usando a ferramenta Iperf, que permite controlar a taxa de geração de tráfego. Foram avaliadas apenas conexões unidirecionais, com taxas de transferências de 100(Mb/s), 150(Mb/s) e

200(Mb/s). A escolha desses valores foi baseada nos resultados obtidos nos experimentos usando o TCP.

A Figura 19 exibe os gráficos com a taxa de perda de pacotes em conexões unidirecionais UDP, *entre compute nodes* diferentes. Em todos os cenários avaliados, a configuração *Linux Bridge* com VXLAN teve menores índices de perda de pacotes (valores apresentados na Tabela 15).

Figura 19 – Taxa de perda de pacotes em conexões unidirecionais UDP (diferentes *compute nodes*)



Fonte: Elaborada pela autora

Tabela 15 – Taxa de perda de pacotes em conexões unidirecionais UDP (diferentes *compute nodes*)

Vazão	*Open vSwitch* com GRE			*Open vSwitch* com VXLAN			*Linux Bridge* com VXLAN		
	2 VMs	4 VMs	8 VMs	2 VMs	4 VMs	8 VMs	2 VMs	4 VMs	8 VMs
100 (Mb/s)	*	0.05 ± 0.1	0.175 ± 0.1618	*	0.05 ± 0.1	0.375 ± 0.1801	*	*	*
150 (Mb/s)	*	*	16.6667 ± 1.2715	*	0.25 ± 0.2159	16.1 ± 1.2837	*	*	0.35 ± 0.213
200 (Mb/s)	*	0.25 ± 0.2587	51.575 ± 1.6355	*	0.25 ± 0.2159	49.4917 ± 1.8813	*	0.05 ± 0.1	1.275 ± 0.3272

Nota – As células destacadas correspondem às tecnologias que apresentaram menores taxas de perda de pacotes.

Nota – As células que apresentam um asterisco (*) representa às tecnologias que não obtiveram perda de pacotes.

As Tabelas 16 e 17 contém a latência e o jitter dos experimentos com a vazão controlada. É possível constatar que os testes usando *Linux Bridge* com VXLAN obtiveram menores taxas.

Tabela 16 – Latência *compute nodes* diferentes

Vazão	*Open vSwitch* com GRE			*Open vSwitch* com VXLAN			*Linux Bridge* com VXLAN		
	2 VMs	4 VMs	8 VMs	2 VMs	4 VMs	8 VMs	2 VMs	4 VMs	8 VMs
100 (Mb/s)	1.2363 ± 0.2118	1.3186 ± 0.1564	2.2022 ± 0.198	0.8819 ± 0.1708	1.1156 ± 0.1333	2.316 ± 0.2243	0.5284 ± 0.011	0.515 ± 0.0068	0.5459 ± 0.0115
150 (Mb/s)	1.2101 ± 0.2336	1.6599 ± 0.2238	15.7558 ± 0.1644	0.7813 ± 0.1394	1.7106 ± 0.2295	15.27 ± 0.242	0.6277 ± 0.1296	0.5243 ± 0.0095	0.8241 ± 0.0811
200 (Mb/s)	0.6748 ± 0.064	2.2687 ± 0.2251	20.304 ± 0.964	0.8681 ± 0.1756	2.2554 ± 0.2853	18.9057 ± 0.3097	0.5587 ± 0.0262	0.5452 ± 0.0152	2.948 ± 0.2291

Nota – As células destacadas correspondem às tecnologias que apresentaram melhores resultados.

Tabela 17 – Jitter *compute nodes* diferentes

Vazão	*Open vSwitch* com GRE			*Open vSwitch* com VXLAN			*Linux Bridge* com VXLAN		
	2 VMs	4 VMs	8 VMs	2 VMs	4 VMs	8 VMs	2 VMs	4 VMs	8 VMs
100 (Mb/s)	1.4876 ± 0.3361	1.664 ± 0.2133	3.734 ± 2.9081	0.9119 ± 0.2199	1.2544 ± 0.1855	2.1886 ± 0.1913	0.3675 ± 0.1274	0.5952 ± 0.0678	0.4864 ± 0.0577
150 (Mb/s)	1.4243 ± 0.3727	1.8817 ± 0.2661	4.3536 ± 0.21	0.8194 ± 0.1461	1.7683 ± 0.2695	4.3651 ± 0.1315	0.4932 ± 0.2033	0.5624 ± 0.0729	0.605 ± 0.1098
200 (Mb/s)	0.588 ± 0.1889	2.4817 ± 0.1427	5.9404 ± 3.7953	0.8385 ± 0.2666	2.1363 ± 0.3117	3.6671 ± 0.1834	0.5162 ± 0.1097	0.4531 ± 0.0824	3.0258 ± 0.2366

Nota – As células destacadas correspondem às tecnologias que apresentaram melhores resultados.

5 CONSIDERAÇÕES FINAIS

Neste trabalho, foi apresentada a importância da virtualização para a computação em nuvem, com foco nas redes de virtualização por sobreposição. Vale ressaltar que infraestruturas de grandes centros de dados são compostas de várias máquinas virtuais e de muitos usuários, uma vez que o ambiente é multi-inquilino. Diante disso, é comum a comunicação entre máquinas pertencentes a diferentes servidores, necessitando a adesão de tecnologias que possibilitem o tráfego dos dados em uma rede pública, dando a ideia de conexão ponto a ponto.

Com a escolha do *OpenStack* como solução de nuvem, foi demonstrado que é possível a implantação de uma infraestrutura completa usando uma tecnologia de código aberto. Estudos utilizando o *OpenStack* são cada vez mais frequentes, qualificando ou aprimorando as funcionalidades fornecidas.

Para alcançar o objetivo deste trabalho, foi desenvolvida uma maneira automatizada para a criação de cada cenário e para a realização de um grande número de experimentos, considerando diferentes métricas, fatores e níveis. Com os experimentos, foi possível comparar o desempenho das técnicas VXLAN e GRE em diferentes contextos, visando avaliar o uso de redes virtuais no *OpenStack*.

Os resultados coletados demonstraram que cenários utilizando a configuração *Open vSwitch* com GRE obtiveram maiores taxas de vazão em experimentos executados no mesmo *compute node*. No entanto, o uso de *Open vSwitch* com VXLAN foi melhor entre *compute nodes* diferentes. Dessa forma, tais tecnologias são recomendadas para ambientes que necessitem de maiores taxas de vazão para um bom funcionamento.

Em contrapartida, *Linux Bridge* com VXLAN apresentou menores taxas de latência e jitter nos experimentos entre *compute nodes* diferentes, requisito essencial quando se trata de cenários que possuem tolerância a atrasos ou variações no atraso. Por outro lado, atingiu maiores índices de perda de pacotes em testes realizados no mesmo *compute node*, e *Open vSwitch* com VXLAN entre *compute nodes* diferentes.

Um comparativo dos resultados é apresentado na Tabela 18, visando demonstrar as tecnologias que obtiveram um melhor desempenho para cada métrica avaliada.

Tabela 18 – Comparativo dos resultados

Resultados	Mesmo *compute node*			*Compute node* diferentes		
	2 VMs	4 VMs	8 VMs	2 VMs	4 VMs	8 VMs
Menor latência	*Linux Bridge* + VXLAN	*Linux Bridge* + VXLAN	*Open vSwitch* + GRE	*Linux Bridge* + VXLAN	*Open vSwitch* + GRE	*Open vSwitch* + GRE
Menor jitter	*Open vSwitch* + GRE	*Open vSwitch* + GRE	*Open vSwitch* + GRE	*Linux Bridge* + VXLAN	*Linux Bridge* + VXLAN	*Open vSwitch* + VXLAN
Menor perda	*Open vSwitch* + VXLAN	*Open vSwitch* + VXLAN	*Open vSwitch* + VXLAN	*Linux Bridge* + VXLAN	*Linux Bridge* + VXLAN	*Linux Bridge* + VXLAN
Maior vazão TCP	*Linux Bridge* + VXLAN	*Open vSwitch* + GRE	*Open vSwitch* + GRE	*Open vSwitch* + GRE	*Open vSwitch* + VXLAN	*Open vSwitch* + VXLAN
Maior vazão UDP	*Linux Bridge* + VXLAN	*Open vSwitch* + GRE	*Open vSwitch* + GRE	N.A.	N.A.	N.A.

Nota – As células que apresentam N.A. representa os cenários onde a taxa de vazão foi controlada.

Uma nova avaliação com um maior número de *compute nodes* e instâncias é sugerido como trabalhos futuros, mensurando o grau de instabilidade da rede para ambientes multi-inquilinos. Além disso, é importante investigar o problema do tráfego UDP entre *compute nodes* diferentes em diversos contextos, buscando mecanismos de detecção e prevenção.

Os scripts desenvolvidos também podem ser utilizados e estendidos para realização de novos experimentos com foco na otimização do desempenho, onde novas configurações podem ser aplicadas ao sistema operacional das máquinas físicas e virtuais, bem como ao *OpenStack*.

REFERÊNCIAS

ABOULNAGA, A.; SALEM, K.; SOROR, A. A.; MINHAS, U. F.; KOKOSIELIS, P.; KAMATH, S. Deploying database appliances in the cloud. **IEEE Data Eng. Bull.**, v. 32, n. 1, p. 13–20, 2009.

AMORIM, T.; BORGES, B.; NACIF. **OpenStack**. 2009. Disponível em: <http://www.gta.ufrj.br/ensino/eel879/trabalhos_vf_2015_2/OpenStack/index.html/>. Acesso em: 13 junho 2016.

ARMBRUST, M.; FOX, A.; GRIFFITH, R.; JOSEPH, A. D.; KATZ, R. H.; KONWINSKI, A.; LEE, G.; PATTERSON, D. A.; RABKIN, A.; STOICA, I. et al. **Above the clouds:** a berkeley view of cloud computing (february 2009). University of California Report No. UCB/EECS-2009-28, 2009, 2009.

BARROS, S.; CARVALHO, R.; REDíGOLO, A.; MAGRI. Applying software-defined networks to cloud computing. In: **XXXIII Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos**. [S.l.: s.n.], 2015.

BRANTNER, M.; FLORESCU, D.; GRAF, D.; KOSSMANN, D.; KRASKA, T. Building a database on s3. In: ACM. **Proceedings of the 2008 ACM SIGMOD international conference on Management of data**. [S.l.], 2008. p. 251–264.

BUENO, H. **Virtualização – Um pouco de história**. 2009. Disponível em: <https://hbueno.wordpress.com/2009/04/29/virtualizacao-um-pouco-de-historia/>. Acesso em: 07 abril 2016.

BUYYA, R.; VECCHIOLA, C.; SELVI, S. T. **Mastering cloud computing:** foundations and applications programming. Newnes, 2013.

CALLEGATI, F.; CERRONI, W.; CONTOLI, C.; SANTANDREA, G. Performance of multi-tenant virtual networks in openstack-based cloud infrastructures. In: **IEEE**. [S.l.: s.n.], 2014. p. 81–85.

CARAPINHA, J.; JIMÉNEZ, J. Network virtualization: a view from the bottom. In: ACM. **Proceedings of the 1st ACM workshop on Virtualized infrastructure systems and architectures**. [S.l.], 2009. p. 73–80.

CAROLAN, J.; GAEDE, S.; BATY, J.; BRUNETTE, G.; LICHT, A.; REMMELL, J.; TUCKER, L.; WEISE, J. Introduction to cloud computing architecture. **White Paper**, 1st edn. Sun Micro Systems Inc, 2009.

CLOUDSTACK. **web page CloudStack**. 2016. Disponível em: <https://cloudstack.apache.org/>. Acesso em: 08 maio 2016.

EUCALYPTUS. **web page Eucalyptus**. 2016. Disponível em: <https://www.eucalyptus.com/>. Acesso em: 08 maio 2016.

FARINACCI, D.; TRAINA, P.; HANKS, S.; LI, T. **Generic routing encapsulation (gre)**. 1994.

GEBREYOHANNES, M. B. **Network Performance study on OpenStack Cloud Computing**. Dissertação (Mestrado), 2014.

HIGGINBOTTOM, G. **CloudStack 101 – Tudo O Que Você Precisa Em Apenas Um Artigo**. 2013. Disponível em: <http://www.shapeblue.com/pt-br/cloudstack-101-2/>. Acesso em: 13 junho 2016.

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

JANSEN, W.; GRANCE, T. Sp 800-144. guidelines on security and privacy in public cloud computing. **National Institute of Standards & Technology**, 2011.

KAWASHIMA, R.; MATSUO, H. Non-tunneling edge-overlay model using openflow for cloud datacenter networks. In: **IEEE**. [S.l.: s.n.], 2013. v. 2, p. 176–181.

KAWASHIMA, R.; MATSUO, H. Performance evaluation of non-tunneling edge-overlay model on 40gbe environment. In: **IEEE**. [S.l.: s.n.], 2014. p. 68–74.

LEDYAYEV, R.; RICHTER, H. High performance computing in a cloud using openstack. **CLOUD COMPUTING**, p. 108–113, 2014.

LIFSHITZ, R. **Tunneling And Network Virtualization: NVGRE, VXLAN**. 2017. Disponível em: <http://www.ran-lifshitz.com/2014/08/24/tunneling-and-network-virtualization-nvgre-vxlan/>. Acesso em: 15 junho 2017.

MAHALINGAM, M.; DUTT, D.; DUDA, K.; AGARWAL, P.; KREEGER, L.; SRIDHAR, T.; BURSELL, M.; WRIGHT, C. **Virtual extensible local area network (VXLAN): A framework for overlaying virtualized layer 2 networks over layer 3 networks**. [S.l.], 2014.

MATTOS, D. M. F. Virtualização: Vmware e xen. **Grupo de Teleinformática e Automação da UFRJ**, p. 13, 2008.

MELL, P.; GRANCE, T. The nist definition of cloud computing. **Computer Security Division, Information Technology Laboratory, National Institute of Standards and Technology Gaithersburg**, 2011.

NURMI, D.; WOLSKI, R.; GRZEGORCZYK, C.; OBERTELLI, G.; SOMAN, S.; YOUSEFF, L.; ZAGORODNOV, D. The eucalyptus open-source cloud-computing system. In: IEEE. **Cluster Computing and the Grid, 2009. CCGRID'09. 9th IEEE/ACM International Symposium on**. [S.l.], 2009. p. 124–131.

OPENNEBULA. **web page OpenNebula**. 2016. Disponível em: <http://opennebula.org/>. Acesso em: 08 maio 2016.

OPENSTACK. **web page OpenStack**. 2017. Disponível em: <http://openstack.org/>. Acesso em: 18 maio 2017.

RABELO, R. J. **Computação em Nuvem (Cloud Computing)**. 2016. Disponível em: <http://docplayer.com.br/632292-Computacao-em-nuvem-cloud-computing.html>. Acesso em: 07 abril 2016.

SANTOS, T. L. **Grades Computacionais**. 2008. Disponível em: <http://www.gta.ufrj.br/grad/08_1/grades/index.html>. Acesso em: 07 abril 2016.

SENGÉS, G.; ALVARENGA, I.; MOREIRA, J. **Redes de Virtualização por Sobreposição: VXLAN, NVGRE e STT**. 2017. Disponível em: <https://www.gta.ufrj.br/ensino/eel879/trabalhos_vf_2012_2/nvgre/>. Acesso em: 15 junho 2017.

SHARATH, S.; BASU, A. Performance of eucalyptus and openstack clouds on futuregrid. **International Journal of Computer Applications**, Foundation of Computer Science, v. 80, n. 13, 2013.

SINGH, P.; SINGH, V. P.; PACHAURI, G. Critical analysis of cloud computing using openstack. **International Journal of Computer Science and Mobile Computing**, v. 3, n. 3, 2014.

SOTO, J. A. Opennebula: implantação de uma nuvem privada e orquestração das máquinas virtuais no paradigma da computação em nuvem. **Monografia, Departamento de Engenharia de Teleinformática-Universidade Federal do Ceará-Fortaleza**, 2011.

SOUSA, F. R.; MOREIRA, L. O.; MACHADO, J. C. Computação em nuvem: Conceitos, tecnologias, aplicações e desafios. **II Escola Regional de Computação Ceará, Maranhão e Piauí (ERCEMAPI)**, p. 150–175, 2009.

SRIDHARAN, M.; GREENBERG, A.; VENKATARAMIAH, N.; WANG, Y.; DUDA, K.; GANGA, I.; LIN, G.; PEARSON, M.; THALER, P.; TUMULURI, C. Nvgre: Network virtualization using generic routing encapsulation. **IETF draft**, 2011.

THOMÉ, B.; HENTGES, E.; GRIEBLER, D. Computação em nuvem: Análise comparativa de ferramentas open source para iaas. **Anais da 11a Escola Regional de Redes de Computadores**, 2013.

VMWARE. **web page VMware**. 2017. Disponível em: <http://www.vmware.com/>. Acesso em: 20 maio 2017.