Análise de Desempenho das Tecnologias de Virtualização de Rede da Plataforma *OpenStack*

Roseli da R. Barbosa[1], **Paulo A. L. Rego**[1], **Michel S. Bonfim**[1] **e Arthur de C. Callado**[1]

[1]Redes de Computadores – Universidade Federal do Ceará (UFC)
Av. José de Freitas Queiroz, 5003 – 63902-580 – Quixadá – CE – Brasil

roselirocha@alu.ufc.br, {pauloalr, michel, arthur}@ufc.br

Abstract. *With the popularity of cloud computing, the need to support multiple tenants on logically independent networks has increased, fostering the advent of network virtualization. In order to evaluate the performance of network virtualization technologies, this paper uses the OpenStack platform to compare the performance of virtual networks using the VXLAN and GRE tunneling protocols with Linux Bridge and Open vSwitch. With the experiments, it was possible to verify that scenarios using Open vSwitch with VXLAN have obtained higher throughput rates between different compute nodes, while Linux Bridge with VXLAN has shown lower latency and jitter rates in most scenarios.*

Resumo. *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. Com o intuito de avaliar o desempenho das tecnologias de virtualização de redes, este trabalho utilizou a plataforma OpenStack visando comparar o desempenho de redes virtuais utilizando os protocolos de tunelamento VXLAN e GRE, com Linux Bridge e Open vSwitch. Com os experimentos, foi possível constatar que cenários utilizando Open vSwitch com VXLAN obtiveram maiores taxas de vazão entre nós de computação diferentes. No entanto, Linux Bridge com VXLAN apresentou menores taxas de latência e jitter na maioria dos cenários.*

1. Introdução

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 é necessária a sua adoção nos grandes centros de dados. 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 virtuais 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.

Visando contornar as limitações das arquiteturas de redes de camada 2, surgiu o conceito de redes de virtualização por sobreposição. Na prática, isso 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.

Dentre as diversas soluções de nuvem existentes, a plataforma *OpenStack* vem ganhando destaque nos últimos anos devido a sua alta escalabilidade, grande flexibilidade e por ser uma solução de código aberto (BARROS et al., 2015). O *OpenStack* utiliza uma variedade de tecnologias para criação de infraestruturas de redes virtualizadas (e.g., *Open vSwitch*, *Linux Bridge*, VXLAN e GRE), ficando o administrador da nuvem responsável por escolher a tecnologia apropriada para seu ambiente.

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 contexto, o objetivo deste trabalho é avaliar o desempenho das tecnologias de virtualização de rede disponíveis na plataforma *OpenStack*. Para tal, o presente estudo implantou uma nuvem privada com a versão Mitaka do *OpenStack* e realizou diferentes experimentos.

O restante deste trabalho está organizado da seguinte forma: na Seção 2, são apresentados conceitos fundamentais referentes à virtualização de redes, com foco na plataforma *OpenStack*; a Seção 3 apresenta metodologia e resultados da avaliação de desempenho; a Seção 4 discute os estudos relacionados a este trabalho; e por fim, Seção 5 apresenta as considerações finais e trabalhos futuros.

2. *OpenStack*

O *OpenStack* é uma plataforma de nuvem de código aberto que surgiu em 2010 através de uma iniciativa da *Rackspace Hosting*[1] 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 a *Rackspace*. Fornece ainda suporte a diferentes hipervisores (Xen, KVM, HyperV, Qemu).

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* (do inglês *Infrastructure as a Service*) 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 bom desempenho em suas funcionalidades.

Dentre os módulos fornecidos pelo *OpenStack* estão os do tipo Core, compostos pelo *Nova, Keystone, Neutron, Glance, Swift e Cinder*, além de outros módulos 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. Na Tabela 2.1 são definidas as principais características de cada módulo.

[1]Rackspace: https://www.rackspace.com.

Tabela 2.1. Principais módulos do *OpenStack*

Nova (*Cloud Management*)	Gerencia o ciclo de vida das instâncias de máquinas virtuais.
Keystone (Identity service)	Valida as credenciais e informações dos usuários e grupos de usuários; Provê dados sobre projetos e domínios; Fornece dados sobre *roles* às entidades geridas pelos serviços de *Identity* e de *Resource*; Valida e administra solicitações de autenticação; Fornece o catálogo de todos os serviços, e por fim fornece 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.
Cinder (Block Storage)	Prover um *block storage* persistente para as instâncias em execução, garantindo que continue existindo mesmo após uma máquina virtual seja apagada.
Horizon (Control Panel/Dashboard)	Interface gráfica para todos os módulos, 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.
Heat (Orchestration)	Mecanismo que permite uma automatização da implantação da infraestrutura, visto que gerencia o ciclo de vida do ambiente.

A próxima seção irá descrever detalhadamente o funcionamento do módulo *Neutron*, também considerado o módulo 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.

2.1. *OpenStack Neutron*

O módulo de rede do *OpenStack* passou por grandes mudanças desde sua idealização, visando se adequar à inclusão de novos protocolos de roteamento e a novas regras de segurança (GEBREYOHANNES, 2014). A proposta do módulo *Neutron* é prover rede como serviço, permitindo a criação de infraestruturas de redes dinâmicas. Por meio de uma API, os demais módulos do *OpenStack* utilizam o *Neutron* para fornecer conectividade entre as VMs.

Em redes *OpenStack*, é essencial entender a variedade de tecnologias que podem ser utilizadas. O *plugin* ML2 (Modular *Layer* 2) 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. A Tabela 2.2 relaciona os *drivers* suportados por cada mecanismo.

- ***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 nó computação, ou em qualquer nó 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.

Tabela 2.2. *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

- ***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 2.2 permitem a criação de redes L2 utilizando os recursos de rede do *OpenStack*, porém o *Open vSwitch* e o *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 é o *OpenStack*, as próximas subseções abordam em detalhes os protocolos de virtualização de redes suportados pela plataforma e que estão diretamente relacionados à proposta de avaliação deste estudo.

2.2. 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, garantindo 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 MAC associados.

2.3. 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 GRE funciona da seguinte forma: o cabeçalho GRE e o cabeçalho IP são adicionados ao pacote e encaminhados 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, embora também possua o cabeçalho Ethernet com os prefixos de origem e destino, nela esses são referentes às interfaces virtuais (SRIDHARAN et al., 2011).

3. Análise de Desempenho

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 da Universidade Federal do Ceará. A instalação se deu de forma manual, seguindo o tutorial oficial[2], com o objetivo de incorporar todas as funcionalidades fornecidas pelos seus módulos. Para tal, foram utilizadas duas máquinas com sistema operacional Ubuntu Server 14.04 LTS de 64 bits com as seguintes configurações: 32 GB de memória RAM, 100 GB de disco e 2 processadores com 12 núcleos cada.

Para a implantação, foram utilizados dois nós. O *host01* implementou a função de *controller*, executando o módulo *Keystone* para fornecer autenticação, autorização, catálogo de serviços e APIs para acesso às funcionalidades. O *Glance* foi usado para armazenar e recuperar as imagens de máquinas virtuais. O *Neutron* e o *Nova* foram configurados no *host01* e no *host02*, onde parte do *Neutron* estava no *host01* para configurar e acomodar os equipamentos de rede e outra parte no *host02* com alguns agentes de rede.

[2]Guia de instalação do OpenStack: https://docs.openstack.org/mitaka/install-guide-ubuntu.

O *Nova* também foi configurado no *host01* e no *host02*, mas como nó computação para que fosse possível instanciar VMs. Além disso, o *host01* estava executando o *Horizon* para fornecer uma interface gráfica de acesso aos módulos. Uma ilustração da nuvem implantada pode ser vista na Figura 1, mostrando em detalhes a organização e a disposição de cada módulo implantado.

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 foi adicionado para fazer a conexão entre as redes externa e interna, e um *floating* IP foi atribuído a uma instância em cada rede privada criada, servindo como um NAT para acessar a instância externamente. A quantidade de instâncias variou entre 2, 4 e 8, onde elas foram alocadas no mesmo nó computação e em nós de computação diferentes (e.g., 1 instância em cada nó, 2 instâncias em cada nó ou 4 instâncias em cada nó).

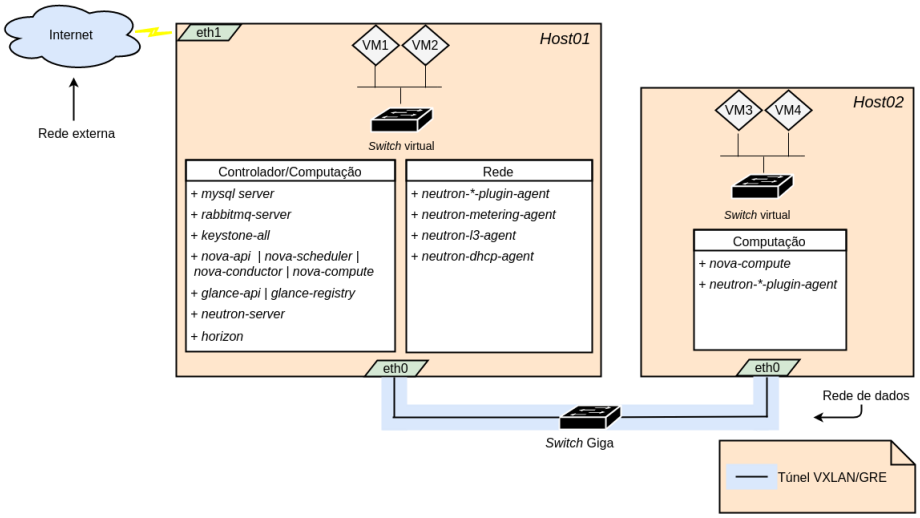

Figura 1. Nuvem *OpenStack* implantada

3.1. Experimentos

Três ambientes foram configurados, alternando o mecanismo de acesso e o protocolo de virtualização de redes utilizados: (i) *Open vSwitch* com VXLAN, (ii) *Open vSwitch* com GRE, e (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 estão disponíveis no repositório do GitHub[3].

Para cada ambiente criado, as seguintes métricas foram avaliadas: (i) vazão TCP e UDP, (ii) latência e jitter, (iii) taxa de perda de pacotes. Tratando-se dos fatores e níveis, consideramos o número de VMs (2, 4 e 8), a disposição ou alocação das VMs (mesmo nó computação e em nós de computação diferentes), mecanismo de comunicação (*Open vSwitch* ou *Linux Bridge*), tecnologia de virtualização (VXLAN ou GRE), tipo do tráfego (TCP ou UDP) e a quantidade de execuções (30 execuções unidirecionais e bidirecionais).

[3]Repositório: https://github.com/rose36/RedesVirtuaisOpenstack.

A criação de cada cenário foi feita de maneira automatizada usando um *script* 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[4]. 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.

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* ou processos de execução, (iii) 256KB de tamanho do buffer, (iv) 30 segundos de duração, (v) 56KB o tamanho da mensagem. Os parâmetros que sofreram variações foram: o tipo de execução (unidirecional ou bidirecional), o IP do *host* remoto e o protocolo que seria utilizado (TCP ou UDP).

3.2. Resultados

Para cada avaliação realizada, buscamos determinar a vazão, latência, jitter e taxa de perda de pacotes 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 (média apresentada em ambas direções). 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 nó computação e entre diferentes nós de computação.

As subseções a seguir exibem os gráficos gerados para a análise comparativa dos resultados. Em experimentos realizados entre instâncias do mesmo nó computação, as taxas de vazão estão em Gigabits por segundo (Gb/s). Já entre nós de computação diferentes, estão em Megabits por segundo (Mb/s). Para criar os gráficos, a média foi calculada de acordo com o número de amostras e fluxos gerados, sendo 30 amostras com 2 VMs (uma conexão), 60 amostras com 4 VMs (duas conexões) e 120 amostras com 8 VMs (quatro conexões). Além disso, foi calculado também o intervalo de confiança utilizando um nível de confiança de 95%.

Nas subseções 3.2.1, 3.2.2 e 3.2.3 são apresentados todos os resultados de conexões utilizando o protocolo TCP. Já para as conexões utilizando o UDP, apenas os resultados dos experimentos entre instâncias alocadas no mesmo nó computação são apresentados, pois foram detectados alguns problemas durante os experimentos utilizando UDP entre nós de computação diferentes. Tais problemas são retratados na Seção 3.2.4, onde são detalhadas as dificuldades encontradas, as medidas tomadas e os resultados obtidos.

[4]Uperf: http://uperf.org.

3.2.1. Vazão TCP e UDP

Em todos os experimentos entre instâncias alocadas no mesmo nó computação, as taxas de vazão em conexões UDP foram maiores quando comparadas com TCP. Os resultados obtidos foram ainda mais significativos em experimentos bidirecionais.

Os gráficos da Figura 2 mostram um comparativo entre as médias da vazão do TCP e do UDP em conexões unidirecionais e bidirecionais, levando em consideração a variação na quantidade de conexões simultâneas (isto é, o valor é por conexão). 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 do UDP obtida foi em torno de 20% a mais quando comparada à média do TCP. Além disso, em todos os casos, os intervalos de confiança ficaram muito próximos à média.

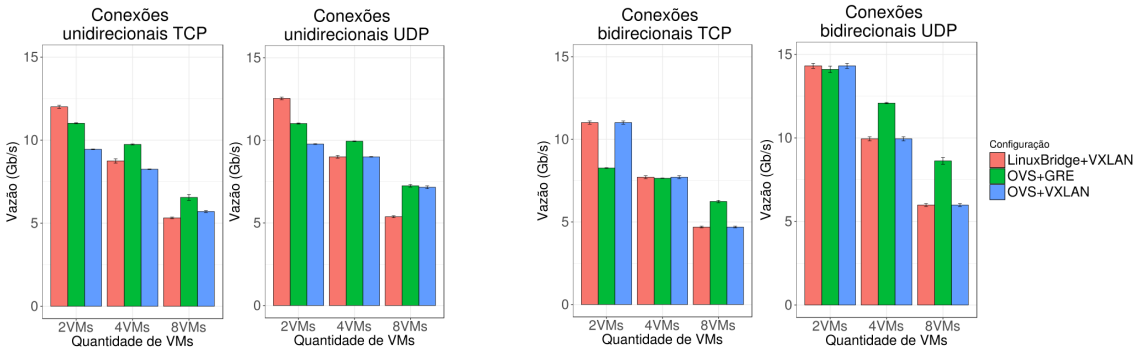

Figura 2. Vazão em Gb/s (mesmo nó computação)

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 Figura 3 apresenta o comportamento do tráfego apenas com fluxos TCP entre nós diferentes. É possível observar que não houve diferenças significativas entre conexões unidirecionais e bidirecionais, e os resultados atingidos foram semelhantes.

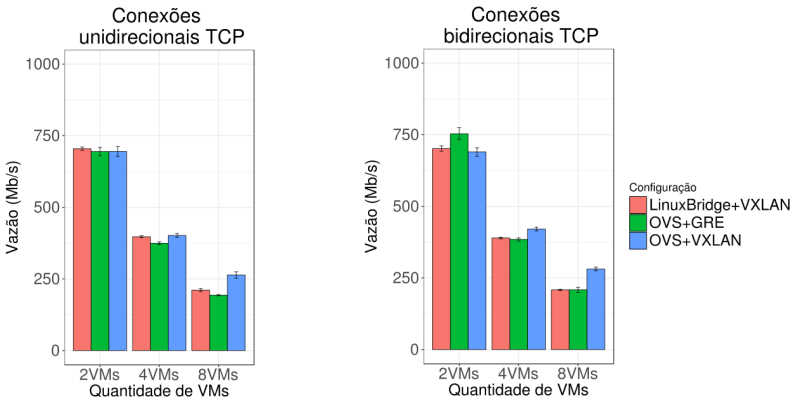

Figura 3. Vazão em Mb/s (diferentes nós de computação)

Outro fator importante de ser destacado nas Figuras 2 e 3 é que o uso do *Open vSwitch* com VXLAN e GRE obteve maiores taxas de vazão na maioria dos resultados.

Além disso, cenários com tais tecnologias apresentaram um melhor desempenho quando utilizados com um maior número de instâncias.

3.2.2. Taxa de Perda de Pacotes UDP

As maiores taxas de perda de pacotes no mesmo nó computação 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, não houve perdas de pacotes.

Os gráficos da Figura 4 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.

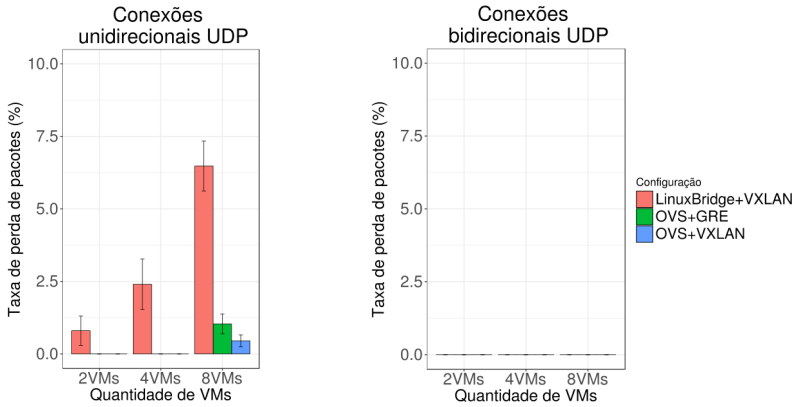

Figura 4. Taxa de perda de pacotes (mesmo nó computação)

3.2.3. Latência e Jitter TCP e UDP

Um comparativo dos resultados da latência e o jitter são apresentados nas Figuras 5, 6, 7 e 8. É possível observar nos gráficos que 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 nós com tráfego TCP, e com 4 e 8 instâncias no mesmo nó 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 nó computação. Já para as conexões bidirecionais UDP, as menores taxas de latência foram alcançadas com *Linux Bridge* e VXLAN (com 2 e 4 VMs).

Com relação ao jitter, na maioria dos resultados as menores médias foram obtidas utilizando *Open vSwitch* com GRE e VXLAN, tanto em conexões unidirecionais como bidirecionais.

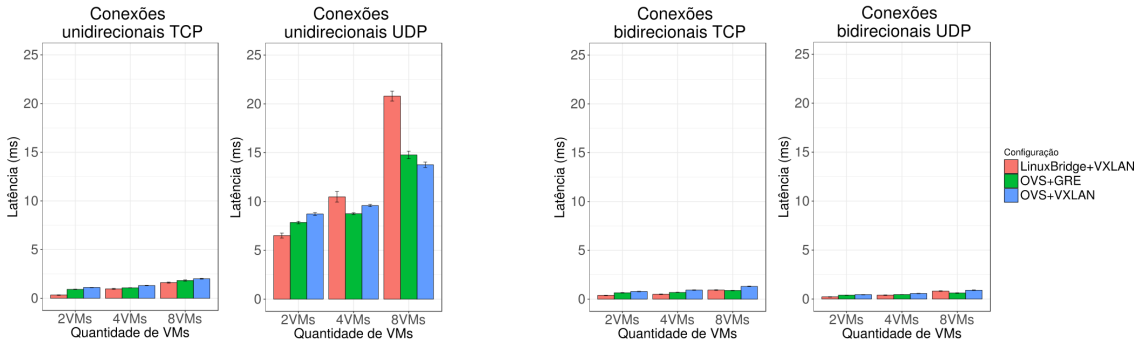

Figura 5. Latência (mesmo nó computação)

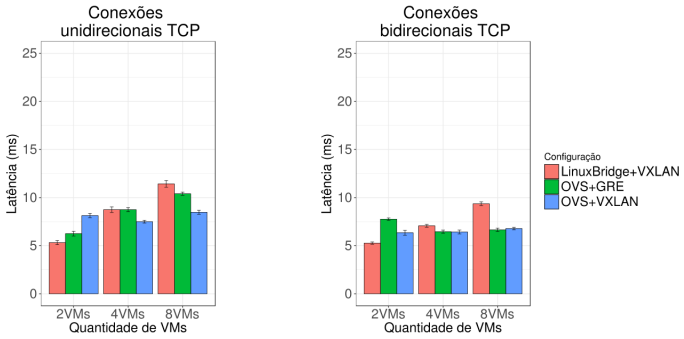

Figura 6. Latência em conexões TCP (diferentes nós de computação)

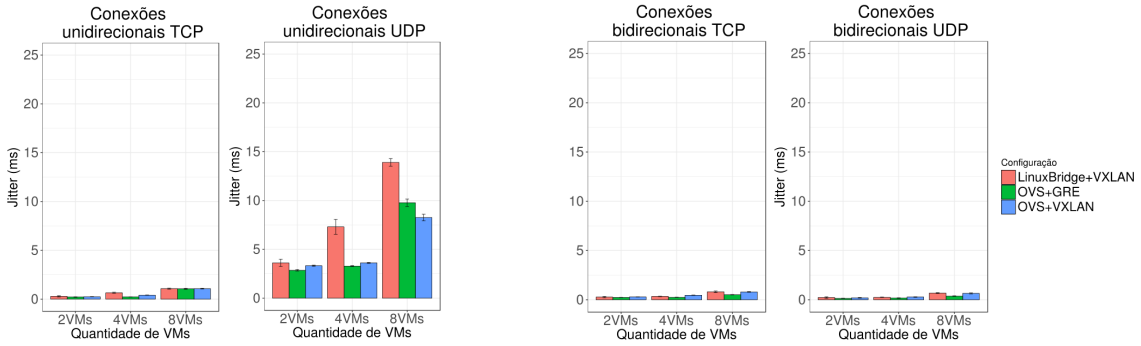

Figura 7. Jitter (mesmo nó computação)

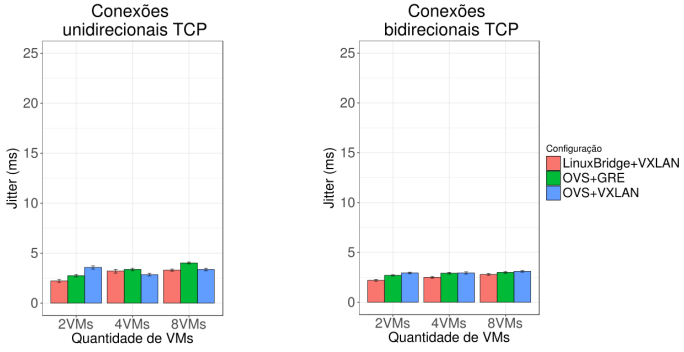

Figura 8. Jitter em conexões TCP (diferentes nós de computação)

3.2.4. UDP entre nós de computação diferentes

A ideia inicial era realizar o mesmo número e os mesmos tipos de experimentos para cada configuração, levando em consideração a alocação das VMs. No entanto, ao realizar experimentos gerando fluxos UDP entre nós de computação diferentes, foi identificada uma instabilidade na rede, a ponto de impedir a comunicação entre as instâncias, o que inviabilizou a execução dos experimentos. Ao avaliar o problema ocorrido, constatou-se que o experimento com UDP estava gerando tráfego como um ataque de negação de serviço, UDP *Flood Attack*, que é caracterizado pelo envio de um grande número de pacotes UDP para portas aleatórias de um determinado *host* remoto.

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 nós de computação diferentes. Constatou-se que o problema persistiu, mesmo utilizando apenas uma conexão para cada rede privada. Quando um dos clientes gera muito tráfego UDP, é possível congestionar e criar instabilidade nos nós de computação envolvidos na comunicação, a ponto de afetar outros clientes.

Diante do problema detectado, outro experimento foi planejado com UDP, mas usando a ferramenta Iperf[5], que permite controlar a taxa de geração de tráfego. Foram avaliadas apenas conexões unidirecionais, com taxas de transferências de 100, 150 e 200 Mb/s. Para a escolha desses valores, experimentos com três conexões foram realizados a fim de maximizar a taxa de vazão. Com os experimentos, foi constatado que acima de 200 Mb/s já apresentava instabilidade entre as conexões.

A Figura 9 exibe os gráficos com a taxa de perdas de pacotes em conexões unidirecionais UDP entre nós de computação diferentes. Em todos os cenários avaliados, a configuração *Linux Bridge* com VXLAN obteve menores índices de perda de pacotes. Já as Figuras 10 e 11 contêm 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.

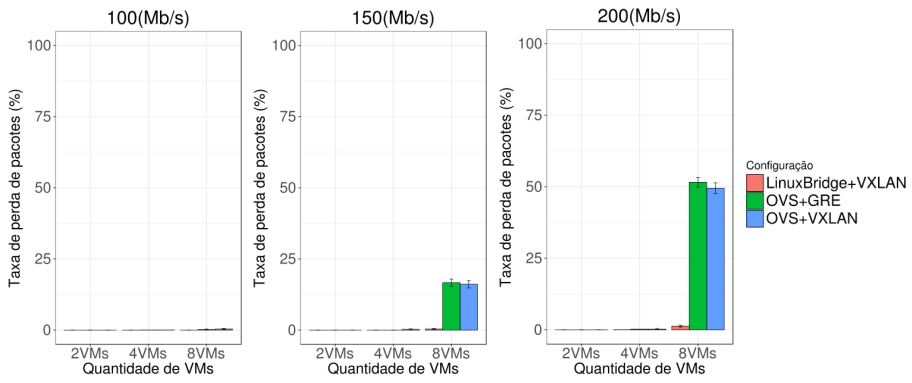

Figura 9. Taxa de perdas de pacotes em conexões unidirecionais UDP (diferentes nós de computação)

A Tabela 3.1 apresenta um resumo dos resultados dos experimentos, visando

[5]Iperf: https://iperf.fr.

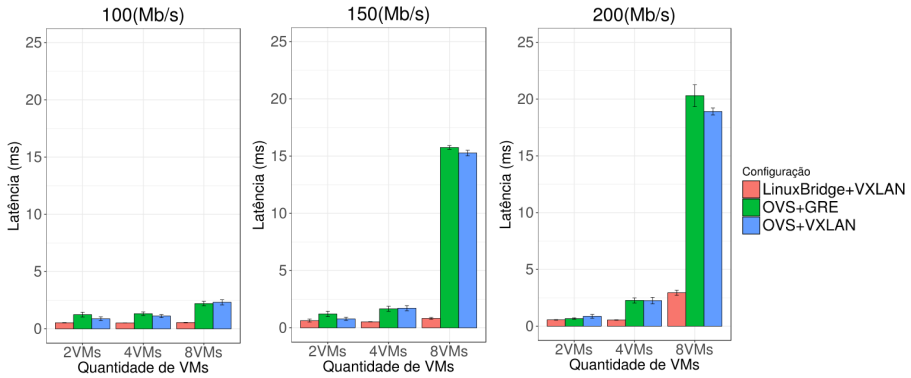

Figura 10. Latência em conexões unidirecionais UDP (diferentes nós de computação)

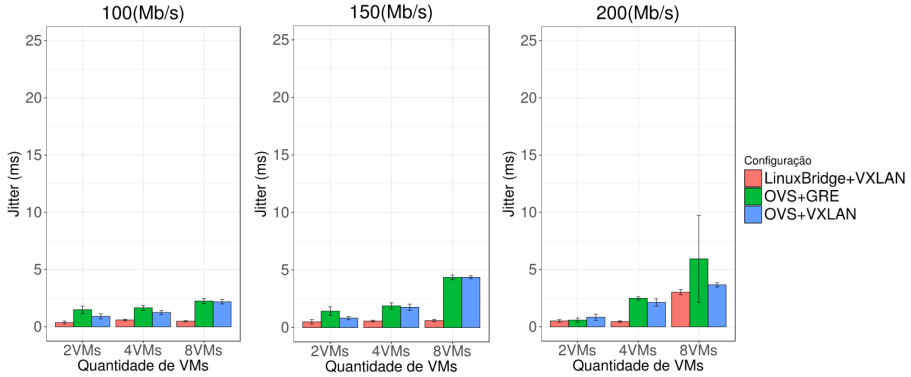

Figura 11. Jitter em conexões unidirecionais UDP (diferentes nós de computação)

demonstrar as tecnologias que obtiveram melhor desempenho para cada métrica avaliada.

4. Trabalhos Relacionados

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 baseada em NFV (do inglês *Network Functions Virtualization*) considerando uma inspeção profunda dos pacotes (DPI, do inglês *Deep Packet Inspection*).

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

Tabela 3.1. Comparativo dos resultados

Resultados	Mesmo nó computação			Nós de computação 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.

tado 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 utilizaram *Linux Bridge* nem deixaram claro no texto qual a solução de nuvem utilizada.

Gebreyohannes (2014) apresentou uma análise de desempenho em redes *OpenStack*, na qual avaliou fluxos de conexão TCP e UDP abordando o desempenho conforme o tipo de alocação das instâncias. Diferente deste trabalho, os autores utilizaram a ferramenta Iperf para geração e medição do tráfego, e criaram cenários apenas com GRE.

5. Conclusão e Trabalhos Futuros

Este trabalho abordou a importância da virtualização para a computação em nuvem, com foco nas redes de virtualização por sobreposição. Neste trabalho, foi implantada uma nuvem privada usando a plataforma *OpenStack*, e foram desenvolvidos *scripts* para automatizar a criação de cenários de experimentação e a realização de um grande número de testes, considerando diferentes métricas, fatores e níveis. Com os experimentos, foi possível comparar o desempenho das tecnologias suportadas pelo *OpenStack* (*Open vSwitch*, *Linux Bridge*, VXLAN e GRE), visando avaliar o uso de redes virtuais na plataforma.

Como pode-se observar, os cenários utilizando a configuração *Open vSwitch* com GRE obtiveram maiores taxas de vazão em experimentos executados no mesmo nó computação. No entanto, o uso de *Open vSwitch* com VXLAN foi melhor entre nós de computação diferentes. Dessa forma, estas tecnologias são recomendadas para ambientes que necessitem de maiores taxas de vazão para um bom funcionamento.

Em contrapartida, em relação às taxas de latência e jitter, o uso de *Linux Bridge* com VXLAN apresentou menores taxas nos experimentos entre nós de computação diferentes. Por outro lado, ao se verificar as perdas de pacotes, o mesmo atingiu maiores índices em experimentos no mesmo nó computação, enquanto o *Open vSwitch* atingiu maiores índices de perdas com VXLAN e GRE entre nós de computação diferentes.

Uma nova avaliação com mais nós de computação e instâncias é sugerida 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 nós de computação diferentes em diversos contextos, buscando mecanismos de detecção e prevenção.

Os *scripts* desenvolvidos também são uma contribuição deste trabalho e estão disponíveis para a comunidade, conforme Seção 3.1. Eles podem ser utilizados para automatizar a criação de cenários e a realização de experimentos, a fim de apresentar um relatório que pode auxiliar o administrador da nuvem a escolher as tecnologias de virtualização de redes a serem utilizadas no seu ambiente. Além disso, os *scripts* podem ser estendidos para utilizar outros módulos do *OpenStack* e realizar diferentes configurações no sistema operacional das máquinas físicas e virtuais, bem como nos mecanismos e *drivers* do *OpenStack*.

Referências

BARROS, S. et al. Applying software-defined networks to cloud computing. In: *33rd Brazilian Symposium on Computer Networks and Distributed Systems*. [S.l.: s.n.], 2015.

CALLEGATI, F. et al. Performance of multi-tenant virtual networks in openstack-based cloud infrastructures. In: IEEE. *Globecom Workshops, 2014*. [S.l.], 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.

FARINACCI, D. et al. Generic routing encapsulation (gre). 1994.

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

KAWASHIMA, R.; MATSUO, H. Non-tunneling edge-overlay model using openflow for cloud datacenter networks. In: IEEE. *Cloud Computing Technology and Science (CloudCom), 2013 IEEE 5th International Conference on*. [S.l.], 2013. v. 2, p. 176–181.

KAWASHIMA, R.; MATSUO, H. Performance evaluation of non-tunneling edge-overlay model on 40gbe environment. In: IEEE. *Network Cloud Computing and Applications (NCCA), 2014 IEEE 3rd Symposium on*. [S.l.], 2014. p. 68–74.

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

MAHALINGAM, M. et al. *Virtual extensible local area network (VXLAN): A framework for overlaying virtualized layer 2 networks over layer 3 networks*. [S.l.], 2014.

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: 15/06/2017.

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.

SRIDHARAN, M. et al. Nvgre: Network virtualization using generic routing encapsulation. *IETF draft*, 2011.