UNIVERSIDADE FEDERAL DO PARÁ
INSTITUTO DE CIÊNCIAS EXATAS E NATURAIS
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

FERNANDO NAZARENO NASCIMENTO FARIAS

vSDNLight: Uma Arquitetura Leve Para Provisionamento de Redes Virtuais Definidas por Softwares

Tese de Doutorado

Belém
2019

UNIVERSIDADE FEDERAL DO PARÁ
INSTITUTO DE CIÊNCIAS EXATAS E NATURAIS
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

FERNANDO NAZARENO NASCIMENTO FARIAS

vSDNLight: Uma Arquitetura Leve Para Provisionamento de Redes Virtuais Definidas por Softwares

Tese de Doutorado apresentado ao Programa de Pós-Graduação em Ciência da Computação. Instituto de Ciências Exatas e Naturais. Universidade Federal do Pará.
Área de Concentração: Redes de Computadores
Orientador: Prof. Dr. Antônio Jorge Gomes Abelém

Belém
2019

Dados Internacionais de Catalogação na Publicação (CIP) de acordo com ISBD
Sistema de Bibliotecas da Universidade Federal do Pará
Gerada automaticamente pelo módulo Ficat, mediante os dados fornecidos pelo(a) autor(a)

F224v Farias, Fernando Nazareno Nascimento
vSDNLight: Uma Arquitetura Leve Para Provisionamento de Redes Virtuais Definidas por Softwares / Fernando Nazareno Nascimento Farias. — 2019.
91 f. : il. color.

Orientador(a): Prof. Dr. Antônio Jorge Gomes Abelém
Tese (Doutorado) - Programa de Pós-Graduação em Ciência da Computação, Instituto de Ciências Exatas e Naturais, Universidade Federal do Pará, Belém, 2019.

1. Redes de Computadores . 2. Redes Definidas por Softwares. 3. Virtualização de Redes SDN. I. Título.

CDD 004.65

UNIVERSIDADE FEDERAL DO PARÁ
INSTITUTO DE CIÊNCIAS EXATAS E NATURAIS
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

FERNANDO NAZARENO NASCIMENTO FARIAS

vSDNLight: UMA ARQUITETURA LEVE PARA PROVISIONAMENTO DE REDES VIRTUAIS DEFINIDAS POR SOFTWARE

Tese de Doutorado apresentada ao Programa de Pós-Graduação em Ciência da Computação da Universidade Federal do Pará como requisito para obtenção do título de Doutor em Ciência da Computação, defendida e aprovada em 12/11/2019, pela banca examinadora constituída pelos seguintes membros:

Prof. Dr. Antonio Jorge Gomes Abelém
Orientador – PPGCC/UFPA

Prof. Dr. Aldebaro Barreto da Rocha Klautau Junior
Membro Interno – PPGCC/UFPA

Prof. Dr. Eduardo Coelho Cerqueira
Membro Interno – PPGCC/UFPA

Dr. Billy Anderson Pinheiro
Membro Externo – UFPA

Prof. Dr. José Ferreira de Rezende
Membro Externo – UFRJ

Visto:
Prof. Dr. Nelson Cruz Sampaio Neto
Coordenador do PPGCC/UFPA

Prof. Dr. Nelson Cruz Sampaio Neto
Coordenador do PPGCC/UFPA
SIAPE: 2659210

Este trabalho é dedicado aos meus pais, irmãos, minha adorável esposa e aos meus dois filhos Enzo e Théo.

AGRADECIMENTOS

Agradeço primeiramente a Deus por guiar o meu caminho a mais está vitória em minha vida, sem a fé e força, seria impossível de realizá-lo.

Agradeço também, ao meus pais, Vera e Flaviano, que abdicaram de muita coisas para me oferecer o maior dom de todos, o conhecimento. Essa vitória é nossa, esse título de doutor é compartilhado com vocês também.

Agradeço aos meus irmãos e sobrinho, Flávio, Bruna, Flávia e Gabriel, pelo torcida e suporte, nos momentos tristes e alegres desta jornada. Mas que apesar de tudo sempre estiveram do meu lado, obrigado!

Agradeço a minha família, em especial a minha esposa, Ingrid. Que foi a minha principal apoiadora e responsável por está vitória, sem ela acredito que teria desistido no meio do caminho, porém, com seu amor e carinho que me alegrou nos momentos tristes, sorriu comigo na alegrias e segurou a minha mão quando precisava seguir em frente. Portanto, meu Amor, esse título eu dedico a você, esse título é seu também. Além disso, não poderia esquecer das duas alegrias da minha vida, primeiramente, o Enzo, que apesar da sua condição de autista, me prova todos os dias, que todos nós somos capazes de grandes feitos independente de nossas dificuldades e ele é prova viva desses grandes feitos e evoluindo a cada dia. E o meu pequenino Théo, que alegra todas as minhas manhãs com o seu sorriso, quando me acorda às 5 da manhã. Obrigado por existirem em minha vida!

Agradeço a minha outra família, o GERCOM, em especial ao meu orientador, mestre e amigo, Prof. Antônio Abelém, que confiou em mim, na minha competência e me direcionou a está vitória. Também gostaria de agradecer a outros amigos, Billy , Igor e Vagner pelos momentos de descontração, conversas e conselhos. E aos demais alunos e colegas que ajudei durante seus trabalhos de conclusão de curso e dissertações de mestrados.

Finalmente agradeço a todos que, de alguma forma, contribuíram para com este trabalho, incluindo os membros da banca de qualificação e defesa, os professores das disciplinas que cursei no PPGCC e colegas de sala de aula. Também gostaria de deixa meu agradecimento aos órgãos da UFPA, CNPq e CAPES, pelo suporte tecnológico e apoio financeiro no desenvolvimento desta pesquisa e tese de doutorado.

“Primeiro eles te ignoram, depois riem de você, depois brigam, e então você vence."
“Você nunca sabe que resultados virão da sua ação. Mas se você não fizer nada, não existirão resultados."
(Mahatma Gandhi).

RESUMO

O surgimento das redes definidas por softwares (*SDN*) e características como programabilidade e abstração na infraestrutura possibilitaram criar novas soluções de virtualização de redes, não apenas compartilhando utilização de um recurso específico da infraestrutura, mas também, compartilhando parte ou toda visão topológica da mesma, ou seja, qualquer recurso disponível da infraestrutura. Para construção de redes virtuais definidas por softwares (*vSDN*), também conhecidas como slices da infraestrutura física, é necessário a utilização de soluções de hipervisores *SDN*. No entanto, essas soluções atuais vêm apresentando limitações de escalabilidade e desempenho, por causa da arquitetura baseada em proxy, que centraliza e sobrecarrega os hipervisores “inchando-os"de atribuições de controle e gerenciamento, e centralizando o plano de controle. Esta tese apresenta uma arquitetura de provisionamento de *vSDNs* através da orquestração da alocação de instâncias de switches virtuais diretamente em dispositivos de comutação, tornando-a uma solução mais leve e distribuída. Resultados mostram que a adoção da proposta *vSDNLight* melhorou o desempenho as redes virtuais quando comparadas a outras soluções de hipervisores SDN.

Palavras-chave: Redes Definidas por *Softwares*. Virtualização de Redes. Hipervisores SDN.

ABSTRACT

The emergence of software-defined networks (SDN) and features such as programmability and abstraction in the infrastructure have made it possible to create new network virtualization solutions, not only sharing the use of a specific infrastructure resource but also sharing some piece or all of its topological view, thus, any available infrastructure resources. For building virtual software-defined networks, which also are well-known as slices, is indispensable to use a hypervisor SDN. However, these solutions have introduced serious limitations of performance and scalability, because of your architecture based on a proxy, which has centralized and overloaded, the hypervisor with high responsibilities of control and management. This thesis presents a vSDN provisioning architecture by orchestrating the allocation of on-demand virtual switches instances, directly to low-cost switching whiteboxes, becoming them a lightweight and distributed solution. Results have shown that architecture can decrease the impact on performing of vSDNs applied by hypervisors SDN.

Keywords: Software-Defined Networking; Network Virtualization; Hypervisor SDN

LISTA DE ABREVIATURAS E SIGLAS

GRE Generic Routing Encapsulation

VXLAN Virtual Extensible LAN

ACLs Access Control List)

API Application Programming Interface

ASIC Application Specific Integrated Circuits

CBE Container-Based Emulator

CLI Command-line Interface

DPDK Data Plane Development Kit

FIBRE Future Internet BRazilian environment for Experimentation

FIND Future Internet Design

GENEVEGeneric Network Virtualization Encapsulation) em bridges 2.0

GENI Global Environment for Network Innovations

IF Internet do Futuro

IP Internet Protocol

MAC Media Access Control)

NAT Network Address Translation

NOS Network Operanting System

NOS Network Operating System

NV Network Virtualization

OVSDB Open vSwitch Database

OVS Open vSwitch

QoS Quality of Service

RPC Remote Procedure Call

SDN Software-Defined Network

SSL Secure Socket Layer

UDP User Datagram Protocol

VLANs Virtual Lan Network

VM Virtual Machines

VNF Virtuais Networking Functions

VPNs Virtual Private Network

vSDN Virtual Software-Defined Network

VSI Virtual Switch Instance

WAMP Web Application Message Protocol

LISTA DE ILUSTRAÇÕES

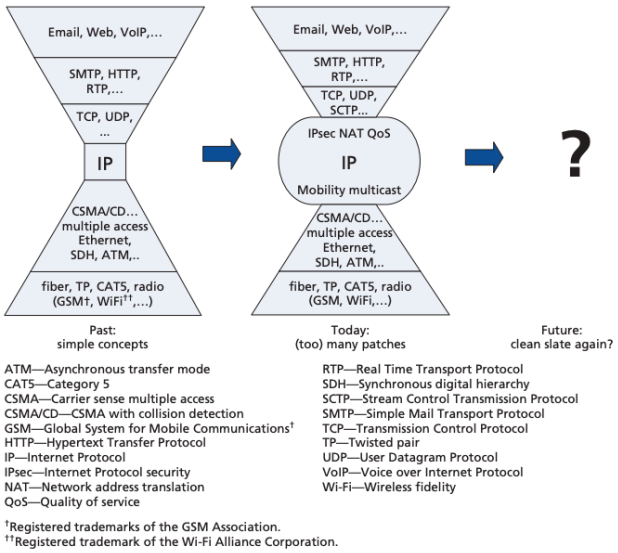
LISTA DE TABELAS

SUMÁRIO

1 INTRODUÇÃO

A Internet é um sucesso mundial que, atualmente, vem mudando a forma das pessoas interagirem, trabalharem e se divertirem. Boa parte deste sucesso se deve à grande flexibilidade oferecida pelo seu principal protocolo o TCP/IP. Apesar de todo o sucesso da Internet, o protocolo IP (*Internet Protocol*) tornou-se a principal limitação para avanços tecnológicos em sua arquitetura. Os principais objetivos da atividade científica, conhecida como Internet do Futuro (IF) (MOREIRA et al., 2009), são justamente formular e avaliar novas alternativas de arquiteturas para evoluir e até mesmo substituir o próprio TCP/IP. Ilustrando esse posicionamento, à Figura 1 apresenta o ciclo evolutivo do TCP/IP frisando tanto a necessidade de mudanças do protocolo quanto o consequente "inchaço" para se adequar às demandas dos usuário, tais como: mobilidade, qualidade de serviços, segurança e escalabilidade (FARIAS et al., 2011).

Figura 1 – Ciclo evolutivo da arquitetura da Internet.



Fonte: Banniza et al. (2009).

Ainda na Figura 1, além de esboçar a atualização da arquitetura TCP/IP, Banniza et al. (2009) também retrata qual caminho seguir em uma nova versão da arquitetura da internet através de uma interrogação. Nesse contexto, duas abordagens ainda estão sendo discutidas e investigadas: a primeira denominada limpa (*Clean Slate*), que visa substituir a arquitetura atual por uma nova totalmente reconstruída, e a outra chamada evolucionária (*Evolutionary*), que pretende evoluir a arquitetura atual sem perder a compatibilidade com a anterior (WAN; YIN, 2018; REXFORD; DOVROLIS, 2010).

As comunidades de pesquisas em IF vêm estudando e desenvolvendo soluções alternativas para substituir ou evoluir o TCP/IP. Como atividades iniciais dessas comunidades, surgiram as redes de experimentações que começaram nos EUA com o programa *Global Environment for Network Innovations* (GENI) (BERMAN; ELLIOTT; LANDWEBER, 2014), e foram se

expandindo para outros lugares como na Europa através da *Future Internet Design* (FIND) (STUCKMANN; ZIMMERMANN, 2009) e no Brasil com o *Future Internet BRazilian environment for Experimentation* (FIBRE) (FIBRE, 2019). Essas iniciativas confirmaram que o caminho para uma arquitetura de sucesso era baseada na programabilidade e virtualização, sendo assim, uns dos principais requisitos para diminuir a barreira dos custos de operação e manutenção e manter o ciclo evolutivo de inovação. Desta forma, as redes definidas por softwares (SDN – *Software-Defined Network*) (HU; HAO; BAO, 2014) vem se tornando a tecnologia principal para substituir a atual arquitetura Internet.

A SDN proporciona uma "solução totalmente reconstruída e desacoplada", por romper com padrões do modelo em camadas do TCP/IP e permitindo a evolução de novas soluções de rede independente dos fabricantes dos dispositivos e também mantendo a compatibilidade com as tecnologias legadas, por exemplo, o próprio TCP/IP. Em sua arquitetura SDN prover um plano de dados controlado por softwares externos aos dispositivos de rede, localizado no plano de controle através dos chamados controladores ou sistemas operacionais de rede (NOS – *Network Operating System*) (BOURAS; KOLLIA; PAPAZOIS, 2017).

Este salto na inovação foi permitido devido ao conceito de programabilidade trazido pelas SDNs, através dela foi possível ampliar as inovações, diferenciações e adaptações (NUNES et al., 2014), rapidamente neste modelo de rede. Além disso, a infraestrutura e seus componentes são abstraídas em uma visão central e global por meio do controlador. As SDNs permitiram muitas evoluções nas redes de computadores, em variadas áreas de pesquisa como: engenharia de tráfego, balanceamento de carga, segurança, roteamento e virtualização de redes (RANA; DHONDIYAL; CHAMOLI, 2019).

Além do quesito programabilidade, a virtualização de redes (NV – *Network Virtualization*) baseada em SDN, também foi uma inovação da arquitetura. O objetivo deste modelo de virtualização foi alocar e abstrair recursos do plano de dados para um controlador especial intermediário (i.e.; hipervisor SDN), localizado no plano de controle, e permitindo criar redes virtuais em paralelo, chamadas de *slices* ou "fatias" (LARA; KOLASANI; RAMAMURTHY, 2013). Esse modelo de redes virtuais permitiram o compartilhamento dos dispositivos físicos SDN através de várias redes virtuais SDN e respectivos controladores.

Prover múltiplas redes virtuais isoladas e em fatias sobre uma infraestrutura física, ainda é um grande desafio mesmo com utilização de SDN (AHMED; TALHI; CHERIET, 2015). A virtualização de redes integrados aos benefícios do SDN, tem atraído a atenção de pesquisadores, tanto na comunidade acadêmica quanto na indústria. Atualmente, está junção é denominada de Redes Virtuais Definidas Por Softwares (vSDN – *Virtual Software-Defined Network*) (BLENK et al., 2016a).

1.1 Motivação e Desafios

O uso de SDN vem sendo a principal abordagem para se aplicar virtualização de redes, porque a partir do uso deste modelo, permitiu-se dividir à infraestrutura em fatias (*slices*) ou redes virtuais (vSDN). Isto melhorou o compartilhamento de recursos – neste caso a infraestrutura física – através da separação ou segregação em diferentes instâncias lógicas de maneira concorrente e isolada, de forma que os recursos compartilhados são resumidamente: nós, portas e enlaces.

As vSDN têm a habilidade de redesenhar a topologia da infraestrutura física em topologias virtuais, as quais podem servir de suporte a uma aplicação específica, onde o processo de criação, remoção ou configuração utiliza o mínimo de intervenção do administrador da rede. No entanto, para o processo de implantação e operação de vSDN é necessário a utilização de um hipervisor SDN (*Hypervisor SDN*) (SHERWOOD et al., 2009).

Em geral, um hipervisor SDN está localizado entre o controlador da rede virtual e a infraestrutura física. Ele atua como "tradutor" ou *proxy*, que recebe as ações oriundas dos *switches* da infraestrutura, através de algum protocolo de controle (ex. *OpenFlow*), e "traduz" essas informações para o *slice* que pertence a mensagem, que enfim o reenvia ao seu controlador responsável, de modo, que este controlador pense que está administrando a infraestrutura física.

Porém, as soluções de hipervisores vêm apresentado problemas de desempenho e escalabilidade que impactam diretamente na funcionamento dessas redes virtuais (BLENK et al., 2016b). Entre os possíveis problemas relacionados a isto, destaca-se como principal contribuidor as traduções aplicadas pelos hipervisores que elevam o índice de processamento, e por consequência degrada o desempenho do mesmo. Além disso, esses hipervisores acumulam várias camadas de funções de controle e orquestração, causando um verdadeiro "inchaço" em sua estrutura elevando a probabilidade de falhas e atrasos, que impossibilitam o aumento da escala das vSDNs.

1.1.1 Desafios Científicos

Os desafios científicos encontrados na virtualização de redes SDN tem observado questões relacionadas ao arquitetura dos hipervisores, distribuições de suas funções e as diminuição das atribuições. Com o objetivo de otimizar seus valores de desempenho e escalabilidade.

A virtualização de computadores tem apresentando uma tendência de minimizar o controle provido pelo seus hipervisores com o objetivo de diminuir e até eliminar a sua participação, essa tendência tem permitido melhorar os resultados no uso de recursos computacionais, por exemplo, memória, armazenamento e processamento (KUMAR; KURHEKAR, 2016). Porém, na virtualização de redes em SDNs isso ainda não é uma tendência e seus hipervisores acumulam muitas atribuições que contribui para queda de desempenho do próprio hipervisor.

As soluções de hipervisores SDN têm algo em comum em sua arquitetura que é a visão

centralizada, ao qual um único elemento é responsável por manipular todas as funções que coordenam o controle e gerenciamento das redes virtuais. No entanto, este modelo de arcabouço limita as soluções com desvantagens referentes a escalabilidade e desempenho. Principalmente, com a leitura e o monitoramento de ações entre os controladores das redes virtuais e os *switches*. Estes desafios têm recebido atenção da academia (AL-SHABIBI et al., 2014; CORIN et al., 2012; SHERWOOD et al., 2009; HAN et al., 2018) e contribuindo para discussão do problema entre os pesquisadores de redes.

Na visão dos hipervisores SDN, também se observa a necessidade de diminuir essas atribuições e interferir o mínimo nas conversas entre dispositivos da rede virtual e seu controlador, e assim, distribuir algumas atribuições para elementos físicos do plano de dados (AL-SHABIBI et al., 2014). Portanto, apesar de novas soluções, o conceito de *proxy* é o principal fator da perda de desempenho das propostas dos hipervisores, ainda fazem partes das mesmas, necessitando a criação de novas arquiteturas menos "inchadas" ou mais "leves" para o gerenciamento de vSDNs.

Sendo assim, quando avalia-se o uso de virtualização de redes em SDN, podem-se destacar os seguintes desafios científicos:

a) Pouca explorações de soluções "leves" para prover a construção de redes virtuais e minimizando a interferência nos dados de controle entre a rede virtual e seu controlador.

b) A carência de estudos que avaliem as vantagens da utilização de soluções de hipervisores SDN sem a presença do conceito de *proxy*.

c) A necessidade de diminuir os problemas de desempenho e escalabilidade (ex. vazão, latência, CPU e memória) aplicados pelo uso do modelo de *proxy* em possíveis soluções de hipervisores SDN.

1.1.2 Desafios Tecnológicos

A principal vantagem apresentada pela SDN é a característica da "abertura" (*openness*), tanto no nível de software como de *hardware*. Sendo assim, neste ecossistema é observado vários níveis de abertura, tais como: padrões abertos (*Open Standards*), softwares livres (*Open Software*), kits de desenvolvimentos (*Software Developer Kits*), interfaces de programação de aplicações e hardwares abertos *(bare metal* ou *whitebox*) (SHIN; NAM; KIM, 2012).

Além disso, observa-se o estudo de soluções envolvendo o plano de dados como a utilização *Whiteboxes*, em referência aos equipamentos de redes corporativos que são verdadeiras caixas-pretas (*BlackBoxs*), para contribuir nos aperfeiçoamentos que completam o funcionamento dos softwares no plano de controle. Neste contexto, tem-se observado que os *whiteboxes* estão seguindo uma tendência de habilitar a criação de instâncias virtuais de *switches* (VSI – *Virtual*

Switch Instance). Isto poderia contribuir positivamente na forma de como as redes virtuais poderiam ser contruídadas.

Para avançar, rapidamente o desenvolvimento desses hipervisores, são necessários soluções que retratem, de maneira mais fiel possível, a infraestrutura ao qual a proposta de hipervisor será aplicada. Além disso, essa solução deve permitir explorar a capilaridade de elementos da infraestrutura como: *switches*, roteadores, servidores ou clientes, de forma escalável e isolada. Nesse contexto, a emulação é o caminho mais curto para realizar uma avaliação de desempenho, testes e depuração. Porém, as propostas encontradas de emuladores (LANTZ; HELLER; MCKEOWN, 2010; WETTE; DRÄXLER; SCHWABE, 2014) não permitem essa fidelidade e são limitadas quanto ao isolamento e independência de seus elementos.

Diante do exposto, identificou-se dois desafios tecnológicos:

a) Como explorar recursos do plano de dados para pensar em soluções que auxiliem os hipervisores SDN;

b) A necessidade de desenvolver emuladores capazes de retratar a fidelidade de infraestruturas de redes, servidores e clientes.

1.2 Questões de Pesquisa e Hipótese

A partir dos desafios apresentados em relação as vSDNs e seus hipervisores, o principal objetivo desta tese é responder a seguinte questão:

Questão Principal: Como oferecer uma arquitetura de virtualização em SDN de forma leve e escalável?

A resposta a essa questão levantada nesta tese apresenta a seguinte hipótese:

Hipótese: As limitações encontradas nas vSDNs podem ser reduzidas através de uma arquitetura leve e distribuída para o provisionamento de redes virtuais definidas por softwares.

Desta forma, a proposta desta tese para validar a hipótese é apresentar arquitetura que não utiliza o modelo de *proxy*, comuns nas soluções atuais de hipervisores, eliminando a camada de virtualização centralizada e distribuindo-a pelo plano de dados em forma de instâncias virtuais de *switches* (VSI), sendo que a união destas instâncias constitui uma rede virtual definida por software (vSDN).

Além disso, os desafios tecnológicos geraram também uma questão de pesquisa suplementar, de modo, a investigar como permitir avaliação da solução proposta e validação da

hipótese. Para suprir o que falta na resposta relacionada à questão principal desta tese, destaca-se a seguinte questão de pesquisa suplementar:

Questão de pesquisa suplementar: Como emular e avaliar essas soluções leves?

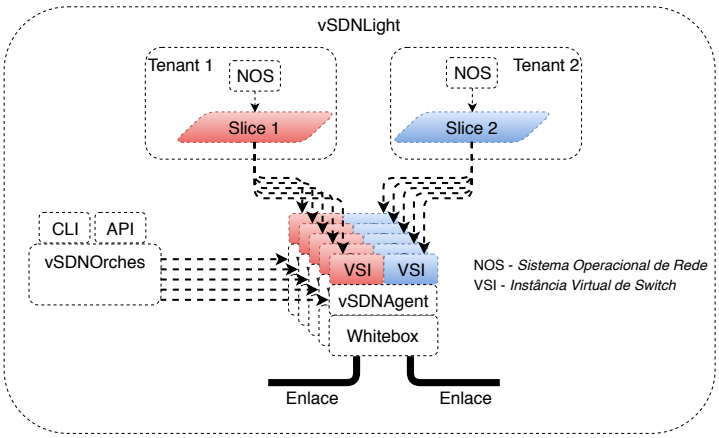
As limitações encontradas nos emuladores redes atuais inviabilizam a inovação e validação de novas propostas de arquiteturas SDN. Isto acontece, pelo fato desses emuladores não permitirem, em sua arquitetura, um certo grau de independência e isolamento nos elementos emulados, além disso, sua própria implementação inviabiliza a obtenção de resultados precisos. Desta forma, também impossibilitam a utilização de instâncias virtuais de *switches*, além de degradar a fidelidade do experimento à medida que escalabilidade vai aumentando.

1.3 Contribuições

Esta tese contribuiu para o avanço nos estudos das redes virtuais definidas por softwares, onde destaca-se as principais contribuições dela dividindo-as em três grupos:

1. **Uma arquitetura leve para provisionamento de redes virtuais definidas por softwares**, a proposta permitiu a implementação de funções de automação e virtualização destas redes através da arquitetura *vSDNLight*. Ilustrado na Figura 2, observa-se os componentes desta arquitetura. Nela, o *vSDNOrches* têm a função de orquestração das vSDN e corresponde um "hipervisor enxuto" com menos atribuições de controle quando comparado a outras soluções de hipervisores. Além disso, proposta remove a camada de virtualização do plano de controle e a distribui no plano de dados através de agentes (*vSDNAgent*) que negociam as alocações de instâncias virtuais de *switches* junto aos dispositivos físicos. De maneira geral, a arquitetura também é responsável pelas informações topológicas, configurações, manutenções e monitoramentos dos *slices* e elementos físicos da rede.

Figura 2 – Arquitetura vSDNLight.



Fonte: Autor.

2. **Um emulador de redes definidas por software capaz de suportar a solução proposta nesta tese**, esta solução possibilita a cada elemento da emulação à execução de suas ações de forma independente, diferente dos emuladores atuais que executam tudo e uma única instância, que por consequência apresentam problemas que podem afetar diretamente os resultados aferidos em experimentos. Além disso, a arquitetura da proposta nesta tese não é compatível com os emuladores disponíveis devido a necessidade da criação de múltiplas instâncias de *switches* em cada nó da topologia. No entanto, com a introdução de contêineres foi possível desenvolver um emulador de redes mais realista, pois cada elemento tem seu próprio recurso computacional (memória, CPU, sistema operacional e etc.), além de sua eficiência ser comprovada através de resultados de desempenho.

3. **Implementação e validação experimental de arquitetura *vSDNLight***, esta tese também criou um protótipo e uma metodologia de avaliação da proposta para aferir desempenhos referentes a memória, CPU, latência e vazão. Com esta avaliação foi possível estimar se arquitetura proposta apresenta resultados promissores.

1.4 Organização do Texto

Além deste capítulo introdutório, o restante do documento está dividido em mais 5 capítulos seguindo a ordem descrita: o Capítulo 2 apresenta os fundamentos tecnológicos e o estado da arte com principais conceitos relacionados a SDN, virtualização de redes e emulação de redes SDN. O Capítulo 3 apresenta uma revisão dos principais trabalhos relacionados as questões de pesquisa principal e suplementar deste tese. O Capítulo 4 discute a proposta de arquitetura leve (*vSDNLight*) para o provisionamento de vSDNs, assim como, seus principais componentes e o modelo de *slice* baseado em instâncias virtuais de *switches*. Além disso, também é descrito a proposta de emulador de redes SDN. O Capítulo 5 traz a descrição dos experimentos realizados e os resultados obtidos. Por fim, as considerações finais e trabalhos futuros são encontrados no Capítulo 6.

2 REFERENCIAL TEÓRICO

Este capítulo apresenta uma abordagem conceitual sobre os principais temas e tecnologias necessárias para o desenvolvimento e entendimento desta tese. As quais destaca-se a compressão do paradigma SDN e as tecnologias envolvidas para o controle e gerenciamento dos dispositivos de redes. Além disso, são abordados os modelos de virtualização de redes, incluindo o modelo baseado em SDN. Por fim, apresenta-se os conceitos de emulação de redes baseado em contêineres.

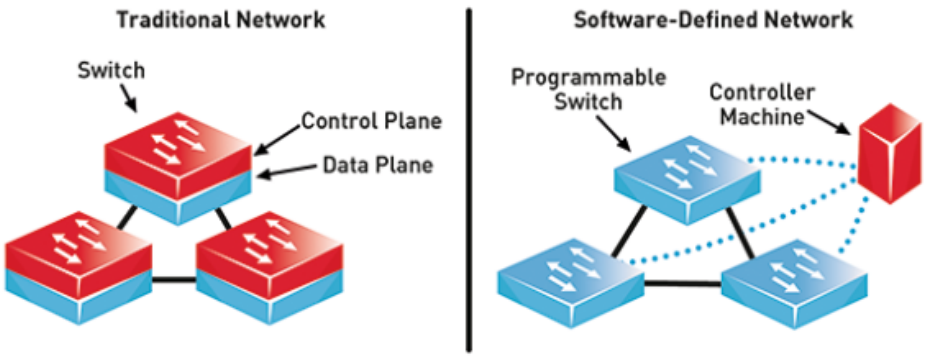
2.1 Redes Definidas por Softwares

Segundo Benson, Akella e Maltz (2009), as redes IP tradicionais são complexas e difíceis de gerenciar. Para atender as demandas de serviços desejados pelos usuários são necessários que os administradores da rede configurem cada dispositivo individualmente usando comandos de terminal (i.e., CLI – *Command-line Interface*) e que geralmente são específicos de cada fabricante. Além da complexidade da configuração, os ambientes de rede precisam suportar a dinâmica de falhas e adaptar-se às mudanças de carga impostas na rede.

A arquitetura das redes IP estão integrada verticalmente, ou seja, o plano de controle, que decide como lidar com tráfego de rede, e o plano de dados, que encaminha o tráfego de acordo com as decisões tomadas pelo plano de controle, estão agrupados dentro dos dispositivos, reduzindo a flexibilidade e dificultando a inovação e a evolução da infraestrutura de rede.

As redes definidas por software é um paradigma que vem sendo proposto como uma abordagem de separação dos planos de controle e de dados, habilitando as redes a suportarem novas funcionalidades como virtualização, automação nas tarefas de operação de forma programável, visão global através da centralização lógica e facilidade na implementação de novos tipos de serviços (KREUTZ et al., 2015). A Figura 3 mostra a diferença entre o modelo tradicional (TCP/IP) e o modelo SDN.

Figura 3 – Redes definidas por softwares vs Redes tradicionais (Legadas).



Fonte: Casado, Foster e Guha (2014).

Responsável pela inteligência da rede, o plano de controle toma as decisões de encaminhamento dos fluxos de pacotes de acordo com comportamento programado em uma entidade chamada de controlador ou também NOS (i.e., sigla em inglês de *Network Operanting System*). Controlador é um elemento remoto (ex. servidor dedicado ou recurso em nuvem computacional) que controla os equipamento físicos da redes SDN através de uma API de controle (ex., API *OpenFlow*). Por outro lado, o plano de dados deixa de executar funções de controle, ficando mais enxuto, e passa a apenas encaminhar os fluxos de pacotes de acordo com sua tabela de encaminhamento definida pelo controlador. Neste caso, o encaminhamento e baseado em informações nos cabeçalhos dos pacotes, onde as ações são mandar para um elemento físico (ex. uma porta *Ethernet*), descartar a entrada do fluxo ou mandar para controlador.

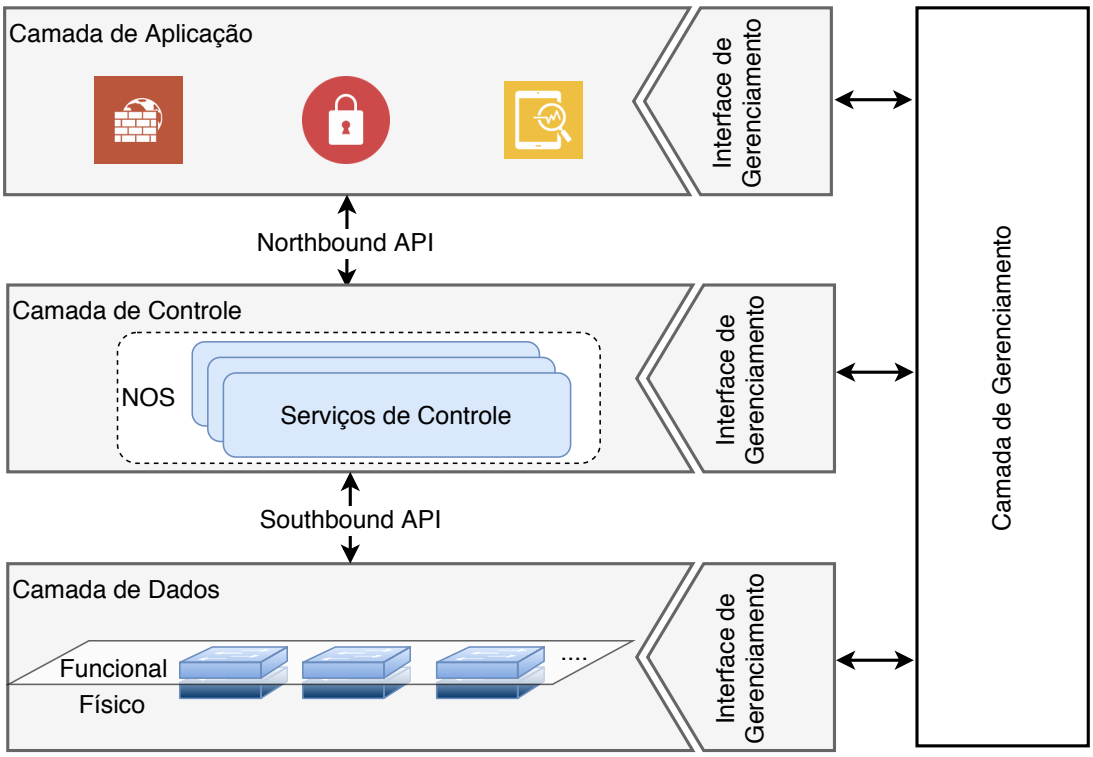
De acordo com Nunes et al. (2014), os benefícios apresentados pela SDN atingem uma variedade de características que atingem o gerenciamento e controle de dispositivos de múltiplos fabricantes, além de aplicações no plano de controle.Por isso pode-se destacar os seguintes benefícios:

- Aperfeiçoamento da automação e gerenciamento através de API abstraindo os detalhes das camadas inferiores de redes dos orquestradores, sistemas de aprovisionamento e aplicações;
- Rápida inovação da infraestrutura seja de hardware ou software, através da habilidade encaminhar novos recursos, serviços e características, sem a necessidade da configuração individual de dispositivos ou a espera por novos softwares de fabricantes;
- Programabilidade dos equipamentos por operadores, empresas, softwares de fabricantes independentes e usuários, usando um ambiente comum de desenvolvimento e programação;
- Aumento na confiabilidade e segurança como resultado de centralização e automação do gerenciamento dos dispositivos de rede, além de uma política uniforme e com pouco erros de configuração;
- Abstração de interfaces específicas de fabricantes ou camadas específicas, provendo configuração flexível do plano de dados por um plano de controle único, baseado em SDN.

2.1.1 Arquitetura

A arquitetura SDN é hierárquica e consiste das seguintes camadas: infraestrutura, controle, gerenciamento e aplicação. Além disso, a comunicação entre camadas são baseadas em APIs abertas, por exemplo, as aplicações de rede da camada de aplicação, se comunicam por meio de ferramentas disponível no controlador para comunicação de alto nível (ex. RESTFul ou WebService). Da mesma forma, o controlador se comunica com os dispositivos de rede através de APIs de controle, tal como: OpenFlow (MCKEOWN et al., 2008). A Figura 4 esquematiza a arquitetura do modelo SDN.

Figura 4 – Arquitetura do modelo SDN.



Fonte: Haleplidis et al. (2015).

Abaixo, têm-se as descrições detalhadas de cada componente e as funcionalidades presentes em cada camada da arquitetura ilustrada na Figura 4:

- ***API Northbound***: é responsável pela comunicação entre a camada de aplicação e a camada de controle.
- ***API Southbound***: é responsável pela comunicação entre os dispositivos de rede da camada de infraestrutura ou dados e os controladores da camada de controle (ex. *Openflow* ou *OVSDB*) (PFAFF; DAVIE, 2013)
- ***Camada de Aplicação***:é onde está presente as ferramentas de controle da rede. Cada aplicação SDN consiste de programas que comunicam suas ações e recursos necessários para o controlador SDN, por meio da API *northbound*. Além disso, as aplicações criam uma visão abstrata da rede através da coleta de informações vindas dos dispositivos da infraestrutura para fins de tomada de decisão. Por exemplo, uma aplicação de monitoramento da rede pode ser criada para reconhecer atividades suspeitas e assim tomar medidas de proteção adequadas.
- ***Camada de Controle***: também chamada de plano de controle, é onde estão os controladores SDN que possuem a visão global da rede e funcionam como uma entidade lógica

que recebe instruções ou requisitos da camada de aplicação retransmitindo suas decisões aos dispositivos de rede através da API *southbound*. O controlador também extrai informações sobre o comportamento da rede, por meio da API *northbound*, fornecendo a elas informações sobre o plano de dados, incluindo estatísticas e eventos sobre o que está acontecendo. Esta visão global facilita na tomada de decisões em larga escala, pois possibilita o gerenciamento e o monitoramento da rede como um todo.

- ***Camada de dados***:é composta de vários dispositivos de rede que formam a rede subjacente responsável por realizar o encaminhamento dos dados. Além disso, ainda há duas subcamadas de execução. Na subcamada física estão os elemento físicos dos dispositivos de cada fabricante (ex. portas elétricas ou ópticas). Já na subcamada funcional, encontra-se os agentes que abstraem a complexidade de acesso às características dos elementos físicos e provêm APIs para camadas superiores da arquitetura.

- ***Camada de Gerenciamento***: é uma camada vertical utilizada basicamente para fazer monitoramento e configuração em dispositivos e aplicações. Sua principal funcionalidade é fazer orquestração das operações disponíveis pelas demais camadas SDN. Apesar do contexto de gerenciamento, esta camada também pode executar ações de controle, no entanto, com a frequência bem menor que a camada de controle.

2.1.2 Protocolos de Controle SDN

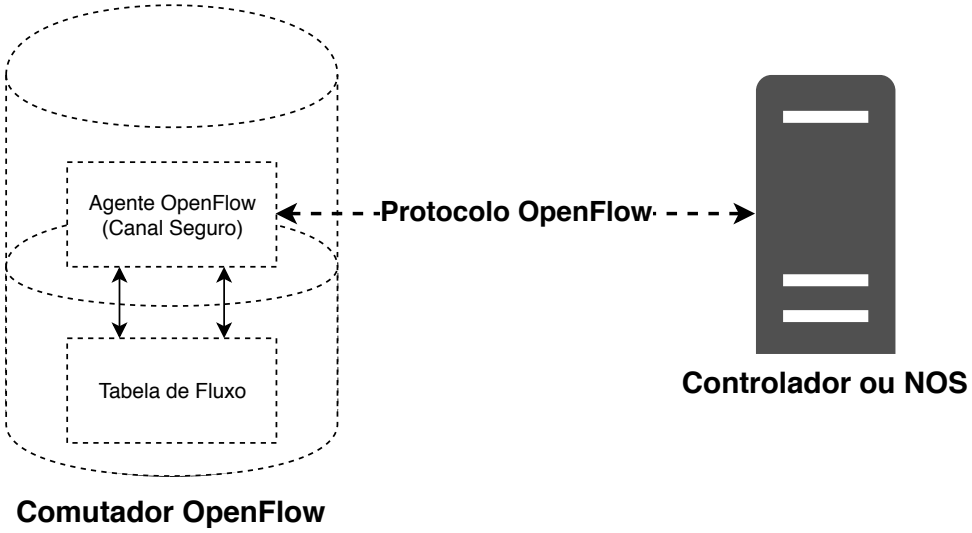
Os protocolos de controle são APIs utilizadas para abstrair a complexidade e heterogeneidades dos fabricantes de equipamentos de rede. Conhecidos como agentes esses elementos fornecem acesso a funcionalidades da camada física dos dispositivos de rede. Atualmente, há duas principais API *Southbound* para fazer o controle e gerenciamento em SDN: *OpenFlow* e *OVSDB*.

2.1.2.1 OpenFlow

O *OpenFlow* é uma proposta tecnológica que criou o alicerce para criação do modelo atual de SDN (Sindhura; Shobha, 2017), seu principal objetivo foi permitir aos pesquisadores executarem seus experimentos em redes de computadores sem interferir no tráfego de produção. Além disso, a proposta permitiu que os fabricantes pudessem incluir novas funcionalidades ao *OpenFlow* em seus comutadores sem necessitarem expor o projeto desses equipamentos. Basicamente, ela é fundamentada em comutadores *Ethernet* e um protocolo padrão para controlar o estado destes comutadores (ex., tabela de fluxos ou estatísticas).

O protocolo OpenFlow explora a tabela de fluxo existentes nos comutadores atuais que são utilizadas para implementar serviços como ACLs, NAT, Firewall ou VLANs. Na Figura 5 tem-se uma visão geral do funcionamento do protocolo.

Figura 5 – Visão geral do Protocolo *OpenFlow*.



Fonte: McKeown et al. (2008).

A Figura 5 apresenta um resumo dos componentes principais que atual no funcionamento do protocolo. Sendo, que neste caso a solução *OpenFlow* é composta por três elementos principais:

- **Tabela de Fluxos**: cada entrada na tabela de fluxos constitui uma tupla de informações para identificação do fluxo, tais como: combinação de campos do cabeçalho do pacote, ações que define o comportamento do fluxo de pacotes – como devem ser processados ou encaminhados, por exemplo, enviar para controlador, descartar fluxo ou encaminhar para porta – e contadores (utilizados para estatísticas e remoção de fluxos inativos).

- **Agente OpenFlow ou Canal Seguro**: Para que a rede não sofra ataques de elementos mal intencionados, o canal seguro aplica a confiabilidade na troca de informações entre o comutador e o controlador. A interface utilizada para acesso ao tráfego é o protocolo *Secure Socket Layer* (SSL). Os principais controladores suportam outras interfaces (passivas ou ativas) como UDP e *UNIX Socket*. Essas são bem úteis em ambientes virtuais, pela simplicidade de utilização, pois não necessitam de chaves criptográficas.

- **Protocolo OpenFlow**: um protocolo aberto para estabelecer a comunicação entre o comutador e o controlador. Ao fornecer uma interface externa que atue sobre os fluxos de um comutador, o protocolo O*penFlow* evita a necessidade de interfaces proprietárias.

2.1.2.2 OVSDB

OVSDB (*Open vSwitch Database*) é um protocolo de gerenciamento da arquitetura SDN. O *OVSDB* foi criado pela *NICIRA* que mais tarde foi adquirido pela *VMWare*. Atualmente é parte do *Open vSwitch* (OVS), que permite a configuração remota de características de s*witches* virtuais, como: elementos de controle a criação de várias instâncias de interfaces de rede virtuais

(VSI), configurar políticas de qualidade de serviço, configurar túneis e gerenciar filas. Por isso, ela é uma interface complementar ao *OpenFlow* para o uso do *Open vSwitch*, pois ela não define comportamento de encaminhamento e sim o gerenciamento de elementos físicos e operações no *switch*.

A interface *OVSDB* é baseada no protocolo *JSON-RPC* [1] que permite executar chamadas de procedimentos remotas, onde os principais procedimentos suportados pelo *OVSDB* incluem:

- Criação, modificação e deleção de *c*ontroladas por *OpenFlow* (i.e., comutadores virtuais *OpenFlow*).
- Configuração dos conjuntos de controladores que cada *bridges* deve conectar.
- Configuração dos conjuntos de gerentes das instâncias do *Open vSwitch*.
- Criação, modificação e deleção de interfaces em bridges OpenFlow.
- Criação, modificação e deleção de interfaces de tuneis (ex. *GRE*, *VXLAN* ou GENEVEGeneric Network Virtualization Encapsulation) em bridgesOpenFlow.
- Criação, modificação e deleção de filas em interfaces de bridges *OpenFlow*.
- Configuração de politicas de qualidade de serviço (QoS) e inclusão dessas políticas as filas de interfaces.
- Coleta de estatísticas.

2.2 Virtualização de Redes

Assim como a virtualização de computadores provê o compartilhamento de recursos de um nó computacional por múltiplos sistemas, a virtualização de redes é um método para que múltiplas arquiteturas de rede heterogêneas compartilhem o mesmo substrato físico. Neste caso, componentes de uma rede como roteadores, comutadores ou topologias.

Segundo Chowdhury e Boutaba (2010) existem três abordagens para a implementação de virtualização de redes: Redes Locais Virtuais (VLANs), as Redes Privadas Virtuais (VPNs) e as redes de sobreposição (*Overlay*). No entanto, com chegada das redes SDN em conjunto do protocolos *OpenFlow* uma quarta solução de virtualização de redes foi proposta inicialmente em Sherwood et al. (2009) baseado no conceito redes programáveis.

[1] Projeto JSON-RPC: https://www.jsonrpc.org/

2.2.1 Redes Locais Virtuais (VLAN)

VLAN (IEEE 802.1Q, 2015) é um agrupamento lógico de dispositivos ou usuários que podem ser unidos por função, departamento ou aplicativo, independentemente da localização de seus segmentos físicos. A configuração de VLANs é feita no comutador, e possivelmente no roteador, através de software proprietário do fabricante.

Numa rede local à comunicação entre diferentes máquinas é governada pela arquitetura física. Devido às redes virtuais locais (VLANs), é possível livrar-se das limitações da arquitetura física (limitação geográfica ou restrições de endereçamento), definindo uma segmentação lógica, baseada num agrupamento de máquinas com critérios como endereços MAC, números de porta ou protocolo, criando assim domínios de broadcast separados.

De acordo com o padrão IEEE 802.1q, são definidos vários tipos de implementações de VLANs em redes locais que baseiam-se de acordo com o critério de comutação e o nível em ao qual se aplica:

- **VLAN de nível 1 (Port-Based VLAN)**: define uma rede virtual em função das portas de conexão no comutador;
- **VLAN de nível 2 (MAC Address-Based VLAN)**: consiste em definir uma rede virtual em função dos endereços MAC das estações;
- **VLAN de nível 3**: este nível se divide em dois modos: VLAN por subrede (*Network Address-Based VLAN*), que associa o segmento virtual de acordo com o endereço IP da subrede e VLAN por protocolo (*Protocol Based VLAN*) que permite criar uma rede virtual por tipo de protocolo, por exemplo: TCP/IP, IPX e *AppleTalk*, agrupando assim todas as máquinas que utilizam o mesmo protocolo numa mesma rede.

A VLAN permite definir uma nova rede sobre a rede física, oferecendo mais flexibilidade na administração e modificação da rede, porque a arquitetura pode ser alterada por simples parametrização dos comutadores. E ainda, há um ganho em segurança, pois as informações são encapsuladas e analisada, não permitindo o compartilhamento de informações com outro segmentos de redes.

2.2.2 Redes Privadas Virtuais (VPN)

A ideia de utilizar uma rede pública como a Internet em vez de linhas privativas para implementar redes corporativas é denominada de VPN (*Virtual Private Network*) ou Rede Privada Virtual (Nawej; Du, 2018). As VPNs são túneis seguros entre pontos autorizados, criados através da Internet ou outras redes públicas e/ou privadas para transferência de informações, com proteção criptográfica, entre redes corporativas ou usuários remotos.

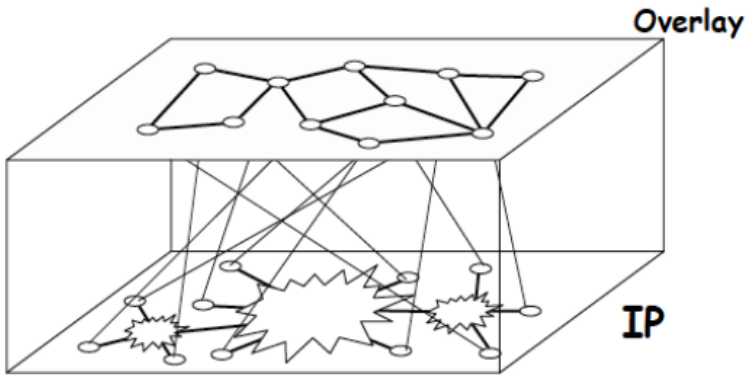
A segurança é a função mais importante das VPN. Uma vez que dados privados são transmitidos pela Internet, um meio de transmissão inseguro, eles devem ser protegidos de forma a não permitir que sejam modificados e identificados. Outro serviço oferecido pelas VPNs é a conexão entre corporações, as *Extranets*, através da Internet, além de possibilitar conexões cifradas que podem ser muito úteis para usuários móveis ou remotos, bem como filiais distantes de uma empresa. Além disso, uma das grandes vantagens do uso das VPNs é a redução de custos com comunicações corporativas, pois elimina a necessidade de enlaces dedicados de longa distância.

2.2.3 Redes de Sobreposição (Overlays)

Uma rede de sobreposição é uma rede ou topologia virtual criada sobre a topologia física de outras redes. Os nós em uma rede de sobreposição são unidos por meio de ligações virtuais que correspondem a caminhos na rede subjacente. A Figura 1.14 ilustra essa afirmação, na camada IP os nós correspondem aos roteadores e sistemas terminais, enquanto na camada *Overlay*, que é uma rede de sobreposição, tem-se a topologia virtual. As redes de sobreposição são tipicamente programadas na camada de aplicação, embora implementações nas camadas inferiores da pilha de redes.

As redes de sobreposição não são restritas geograficamente e são bastante flexíveis e adaptáveis a mudanças, se comparadas a qualquer outra rede. Como resultado, as redes sobrepostas têm sido usadas para implantar novos recursos e extensões na Internet.

Figura 6 – Exemplo de rede de sobreposição.



Fonte: Autor.

Muitos modelos de sobreposição têm sido propostos para resolver problemas que incluem: desempenho assegurado, disponibilidade de roteamento na internet, multicast, garantias de qualidade de serviço, ataque de negação de serviços e compartilhamento de arquivos (Galan Jimenez; Gazo Cervero, 2011). Redes de sobreposição também estão sendo usadas como ambientes de testes, por exemplo, o *PlanetLab* [2] um ambiente para o desenvolvimento e avaliação de

[2] Projeto PlanetLab: https://www.planet-lab.org/

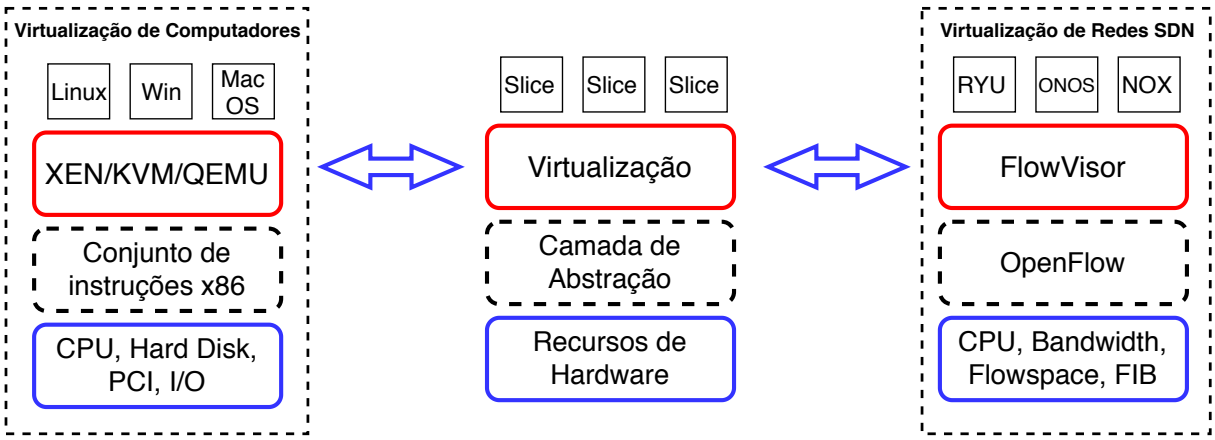
novas arquiteturas da internet.

2.2.4 Virtualização de Redes Baseado em SDN e OpenFlow

Similar à virtualização de computadores, a virtualização de redes promete melhorar a alocação de recursos além de permitir que seus operadores possam checar suas redes antes de eventuais mudanças, e também que clientes compartilhem o mesmo equipamento de forma controlada e isolada. Esta camada deve ser facilmente "fatiável", para que múltiplas redes possam ser executadas simultaneamente sobre uma variedade de hardwares disponíveis no plano de dados.

Em Sherwood et al. (2009), observa-se a solução inicial de virtualização de redes SDN. Nesta solução, a camada de abstração de hardware é provida pelo *OpenFlow* e a camada de virtualização é da pela solução *FlowVisor*. Ele foi a primeira solução de hipervisor SDN e a base para as soluções posteriores a ele. Este hipervisor é uma camada de abstração que reside entre o hardware e o software dos componentes da arquitetura, conforme ilustrado na Figura 7. Portanto, a integração *FlowVisor* e *OpenFlow* permitiu que uma rede SDN pudesse ter seus de recursos fatiados de forma simultânea e isolada.

Figura 7 – FlowVisor como camada de virtualização de redes.



Fonte: Sherwood et al. (2009).

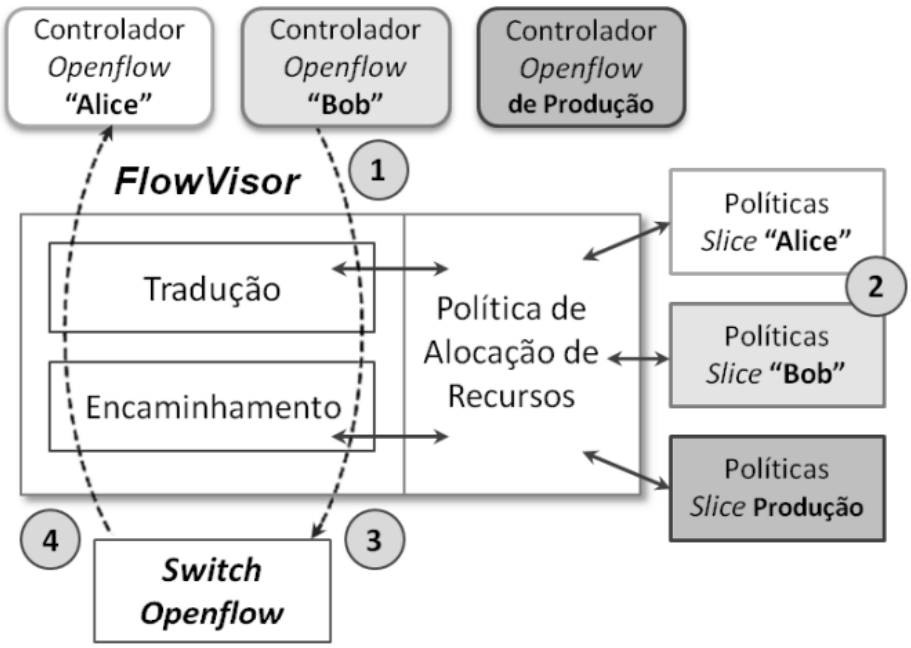
O *FlowVisor* é um controlador especializado que atua como um proxy transparente entre os comutadores da rede física e seus múltiplos controladores. Todas as mensagens do protocolo *OpenFlow* tanto aquelas originárias dos comutadores para os controladores quanto as dos controladores para os comutadores, são interceptadas através do *FlowVisor*. Desta forma, não é necessária a modificação dos controladores e dos comutadores da rede, uma vez que o uso do *FlowVisor* é transparente aos demais elementos de rede.

Cada fatia está vinculado a um controlador. O *FlowVisor* define uma fatia como um conjunto de fluxos também chamado de "espaço de fluxo" ou *FlowSpace*, que neste caso é uma combinação de até dez campos do cabeçalho de pacote, incluindo informações das camadas física, de enlace, de rede e de transporte, os quais estão disponíveis na versão 1.0 do

protocolo O*penFlow*. O *FlowVisor* possibilita que se implemente fatias com um elevado nível de granularidade, no que diz respeito à caracterização do *FlowSpace*. Além disso, as fatias podem ser definidas por ações de negação, união e intersecção.

A Figura 8 ilustra, os procedimentos implementados pelo *FlowVisor* para gerenciamento das fatias de rede. Além do controlador de produção, outros dois controladores são representados, identificados por "Alice" e "Bob", o que determina, consequentemente, a presença de três *slices*. A dinâmica de processamento das mensagens interceptadas pelo *FlowVisor* está representada pelos círculos numerados. Inicialmente, mensagens *OpenFlow* originadas de um controlador são interceptadas (1). Aplicando a política do espaço de fluxos definida para a fatia (2), as mensagens são reescritas de forma transparente, a fim de garantir o controle de determinada fatia da rede que compõe o plano de dados (3). Da mesma forma, as mensagens provenientes dos comutadores para o plano de controle (4) são novamente interceptadas e encaminhadas ao controlador correspondente, de acordo com a política de definição da rede virtual.

Figura 8 – *FlowVisor* como camada de virtualização de redes.



Fonte: Adaptado de Sherwood et al. (2009).

As características inerentes ao *FlowVisor*, como a virtualização transparente, o isolamento entre as fatias e a sua política de definição de *FlowSpace* tornam o mesmo uma grande ferramenta no que diz respeito à virtualização e implementação de redes SDN. No entanto ele não é a unica solução de hipervisor SDN.

2.3 Emulação Baseado em Contêiner

Um emulador de rede representa a execução de um software com configurações, características e componentes, existentes em uma rede real, tais como: switches, enlaces, tráfego de dados, clientes e servidores (ZHENG; NI, 2003). Além disso, um emulador suporta topologias de redes arbitrárias com hardware real ou virtual. Emuladores são verdadeiros testbed portáteis para a avaliação de condições da rede, tais como: latência, largura de banda, taxa de descarte, vazão e falhas em geral (ex., enlaces, nós e congestionamentos).

Inicialmente, os emuladores de redes eram baseados em virtualização total (Anish Babu et al., 2014) . No entanto, o uso de maquinas virtuais (VM - *Virtual Machines*) é um método sobrecarregado, pois podem trazer mais problemas que soluções. Neste caso, os problemas relacionados ao tamanho da VM e o overhead dos hipervisores (ex., XEN[3] , KVM[4] ou QEMU[5]) que limitam a escalabilidade da emulação dos nós, por alocar muitos recursos computacionais. Além disso, a memória alocada para cada VM tem restringido o desempenho de *switches* e *hosts*. Portanto, usar virtualização total para gerar emulação de redes requer servidores extremamente robustos para suportar uma ampla variedades de elementos na emulação.

A emulação de redes baseado em contêineres (CBE – *Container-Based Emulator*) (HANDIGOL et al., 2012), tem como base a técnica de virtualização em contêineres. Esse tipo técnica, tem se tornado popular por oferecer vantagens relacionadas a eficiência e escalabilidade, principalmente, por oferecer um modo mais leve de virtualização, pois não virtualiza o computador inteiro. Neste caso, cada VM torna-se um simples grupo de processos no sistema operacional. No entanto, esta solução é fortemente dependente do sistema operacional hospedeiro, tais como: *Linux Namespaces*, *FreeBSD Jails* e *OpenSolaris Zones*.

O *Linux Namespace* é a principal tecnologia para implementação de CBE e acelerou a chegada das SDNs, pois no desenvolvimento de soluções para SDN um contêiner retrata fielmente um nó de rede onde essas aplicações serão executadas. Inicialmente, para isso utilizava-se ambientes experimentais como *GENI* e *FIBRE*, porém nem sempre seus recursos estão totalmente disponíveis e a escalabilidade é limitada. Por isso, o uso de CBE no desenvolvimento de aplicações em SDN, permite explorar cenários mais ricos em um computador pessoal.

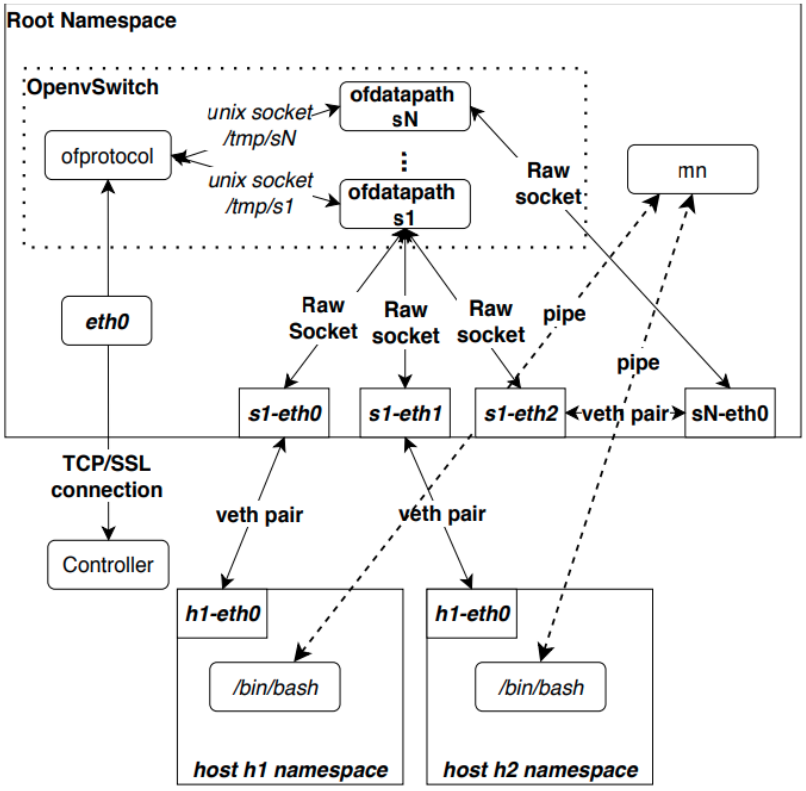
Nesse contexto, os emuladores de SDN tem como propriedade chave o uso de namespaces, no qual pode-se destacar o emulador *Mininet* (LANTZ; HELLER; MCKEOWN, 2010) como principal solução. A Figura 9 apresenta a arquitetura do *Mininet*.

Essencialmente, o *Mininet* cria suas topologias de redes através de processos de *namespaces*, como mostrado na Figura 9, ele possui um nam*espace root* onde ficam localizados os *switches* virtuais criados através do *OpenvSwitch* e vários *namespace host* emulando clientes ou

[3] Projeto XEN : http://xenproejct.org
[4] Projeto KVM: https://red.ht/2u1fPNk
[5] Projeto QEMU: http://qemu.org

usuários da rede. As conexões entre os dispositivos e hosts da rede são estabelecidas através de pares de interfaces ethernet virtuais (i.e., *veth pairs – virtual ethernet pairs*) que consistem na emulação de duas interfaces ethernet ligadas por um cabo par trançado virtual.

Figura 9 – Arquitetura do Mininet



Fonte: Adaptado de Lantz, Heller e McKeown (2010)

O fato de todos os *switches* serem executados em um único processo do Linux, acarreta em problemas ao emulador, pois não é possível garantir o isolamento dos dispositivos da rede, o que não permite impor, por exemplo, limites de recursos computacionais a cada um deles, o isolamento das interfaces de rede ou a execução de múltiplos sistemas operacionais entre eles. Além disso, caso ocorra alguma anomalia ao processo, toda a topologia será perdida, afetando negativamente o experimento a ser executado ou que estava em execução. Apesar dessas limitações em sua arquitetura, o *Mininet* ainda é um dos principais emuladores utilizados no estudo em SDN, devido sua facilidade de instalação e manuseio.

2.4 Conclusão do Capítulo

Este capítulo teve como objetivo apresentar um resumo a respeito de SDN, sua arquitetura e os principais protocolos de controle e gerenciamento disponíveis atualmente. Além disso, este capítulo mostrou as principais formas de se obter a virtualização de redes, em especial, a virtualização de rede baseada em SDN, onde foram descritas suas características e relevância com objetivo de fornecer o claro entendimento da proposta deste trabalho. Também foi feita uma

revisão sobre a utilização da virtualização de contêiner para permitir um emulação de redes mais escalável e leve.

3 TRABALHOS RELACIONADOS

Este capítulo apresenta os trabalhos mais relevantes relacionados aos temas explorados no contexto desta tese. Estes trabalhos estão organizados em duas áreas: (1) Virtualização de Redes SDN e (2) Emulação de Redes SDN. O objetivo é auxiliar nas respostas das questões de pesquisa, principal e suplementar, apresentadas no Capítulo 1.

3.1 Virtualização de Redes SDN

A virtualização de redes SDN requer a necessidade de hipervisores SDN (AL-SHABIBI et al., 2014; SHERWOOD et al., 2009), esses hipervisores são elementos com o objetivo de controlar e gerenciar tanto a rede virtual quanto a física. De acordo com Blenk et al. (2016b), um hipervisor é conceituado como um proxy que interage através de múltiplas interfaces no plano controle com os controladores das vSDNs e validando toda comunicação entre os dispositivos físicos e esses controladores.

Quanto as características dos hipervisores, nos artigos de Drustskoy, Keller e Rexford (2013) e Han et al. (2018), observa-se que esse controle excessivo traz muitas penalidades a virtualização utilizando hipervisores, pois, além controlar a rede física o hipervisor deve gerenciar as demais redes virtuais e ainda fazer verificação e rescrita de mensagens de controle, tornando essas soluções "pesadas" do ponto de vista de execução. Além disso, esse acúmulo de funções causam problemas relacionados a escalabilidade e desempenho.

Em Blenk, Basta e Kellerer (2015), comenta-se que a decomposição das atribuições do hipervisor em funções menores e distribuídas em diferentes posições da arquitetura flexibiliza o desempenho de maneira geral. Ou seja, quanto mais transparente ou leve o hipervisor vai ser tornando melhores são resultados de desempenho do mesmos. Ainda, neste artigo os autores apresentam vários modos organização dessas funções. Neste contexto, destaca-se o modo em que partes das funções do hipervisor são levadas para elementos de redes no plano de dados. E conclui que ao trazer as funções de virtualização para mais perto da infraestrutura física, isto poderia resultar num aperfeiçoamento da escalabilidade e desempenho da arquitetura como todos. No entanto, os autores não aplicaram nenhuma validação para confirmar a eficiência desse modo.

No trabalho de Chen e Benson (2017) , observa-se a primeira de prova de conceito em trazer as funções de virtualização do hipervisor para o plano de dados. Neste trabalho, Chen e Benson (2017) propõe o uso de virtualização de contêineres para representar o switch e com isso isolar recursos computacionais (ex., CPU e Memória) ao mesmo. Desta forma, ele cria instâncias separadas para acesso de diversos agentes do plano de controle. Por outro lado, no trabalho de (LANGENSKIÖLD, 2017) observa-se o mesmo processo, porém utilizado o *Open vSwitch* para criar *switches* virtuais e defini-los como um *slice*. Além disso, ele faz algumas análises de viabilidade que comprovam que o agrupamento de *switches* virtuais trazem vantagens

quando comparado ao hipervisor SDN baseado em proxy. No entanto, os autores não aplicam nenhuma orquestração ou arquitetura que determine como a solução deve ser implementada, a forma de como essas vSDNs devem ser definidas na redes ou gerenciamento do ciclo de vida dessas vSDNs.

Em relação as soluções de hipervisores, começa-se com Sherwood et al. (2009). Nele, tem-se os primeiros passos da virtualização de redes SDN através do *FlowVisor*. Este hipervisor, atua como um *proxy* transparente entre o *switch* e o controlador do usuário permitindo que a infraestrutura possa ser subdividida em *slices* para os vários usuários de maneira isolada. Cada *slice* é regulado por uma entrada de fluxo chamada de *FlowSpace*, especificamente o seu espaço de endereçamento segue as características dos campos disponíveis no protocolo *OpenFlow*, garantindo que as redes não se sobreponham.

A solução *FlowVisor* vem sofrendo críticas devido ao seu desempenho, pois a mesma entidade responsável pela gerência e controle das informações de virtualização. Ela também é responsável por ler e encaminhar as mensagens geradas entre infraestrutura e controlador da rede virtual, gerando uma sobrecarga no hipervisor quando a escala de dispositivos no *slice* vai aumentando. Isto acontece, pois, para fazer esta entrega o *FlowVisor* tem que realizar uma busca pelo controlador correto do *slice* entre seus *FlowSpaces* e reescrever os cabeçalhos do pacotes. Além disso, sua arquitetura é centralizada e os tratamentos de orquestração estão também vinculados ao hipervisor sobrecarregando-o à medida que a demanda por novos *slices* vai aumentando.

O próximo hipervisor definido em Salvadori e Corin (2012), observa-se a necessidade de começar a superar algumas limitações apresentadas pelo *FlowVisor*. O trabalho foca na construção de topologias virtuais, tendo em vista que o *FlowVisor* atual não suporta essa características e suas topologias estão restritas a subconjuntos da topologia física. Chamada de *AdVisor*, ele estende a proposta de Sherwood et al. (2009) em algumas direções, como: implementar uma extensão no mecanismo de abstração para esconder os switches físicos na topologia virtual objetivando mais flexibilidade na abstração da topologia, o compartilhamento do *FlowSpace* por múltiplos *slices* e definição do uso de "slice-tags" que abrangem utilização de identificadores de VLAN, rótulos MPLS ou pilhas VLANs (*802.1ad*). Com essas tags a proposta aproveita para diferenciar os enlaces das redes virtuais, tornando possível a criação de enlaces virtuais através da junção de diferentes enlaces físicos.

A solução limita-se ao uso de cabeçalhos de VLAN, MPLS e QinQ, que dependando do protocolo pode limitar a escalabilidade do hipervisor. Além disso, os testes mostraram um aumento da latência quando comparado ao *FlowVisor*, por adicionar uma sobrecarga no uso das tags, e assim limitando o desempenho das vSDNs. Adicionalmente, proposta ainda inclui a necessidade de configuração manual dos dispositivos de rede e o envio de comando através da CLI, favorecendo a introdução de erro de configuração.

Ainda sobre limitações na construção de topologias virtuais por hipervisores, o trabalho

de Corin et al. (2012) apresenta o *VeRTIGO*, proposto como uma extensão do *FlowVisor*, ele agrega uma inteligência adicional que é capacidade de expor diferentes visões ou formas de topologia à diferentes controladores, através de uma camada a mais de abstração das topologias. Este trabalho introduz o conceito de "*big switch*" em vSDN, ou seja, um único nó virtual pode abstrair uma topologia física inteira, porém, o controlador desta vSDN visualizará apenas um nó.

A proposta foi aplicada no testbed *OFELIA*, todavia, por mais que proposta tenha alcançado o seu objetivo de prover enlaces virtuais e virtualização completa, ela acaba acrescentando uma sobrecarga de processamento ao hipervisor, como demonstrado nos próprios resultados do trabalho, o que impacta ainda mais na escalabilidade e desempenho do *FlowVisor*.

No trabalho de Drustskoy, Keller e Rexford (2013) é apresentado o *OpenVirteX* que prover duas principais contribuições à virtualização de redes SDN: a virtualização do endereçamento e enlaces. A primeira permite ampliar o espaço de endereçamento, que era limitado nas outras soluções, por ser utilizado para identificar suas redes virtuais, agora é possível usar todo o espaço de endereçamento às vSDNs. A segunda, prover enlaces virtuais a proposta, fazendo o mapeamento da rede virtuais para física. Os resultados apresentados pelos *OpenVirteX* mostram uma latência bem menor quando comparado aos outros hipervisores (ex., *FlowVisor* e *FlowN*).

Já em Blenk, Basta e Kellerer (2015), tem-se a solução *Hyperflex*, uma versão distribuída de hipervisor SDN, que é uma proposta de arquitetura que utiliza a segregação da camada de virtualização para outros pontos da arquitetura, e com isso, alcançar resultados maiores de escalabilidade. Na prova de conceito o *FlowVisor* e o *Open vSwitch*, foi utilizado em conjunto para prover a camada de virtualização e isolamento do plano de controle. Porém, a proposta não apresenta comparativos com outras soluções de virtualização ou testes de carga, além de oferecer uma complexidade grande de implementação, uma vez que utiliza vários tipos de elementos físicos e virtuais. A prova de conceito apresentada utiliza o *FlowVisor* como parte da solução, e desta forma, acaba herdando seus problemas de escalabilidade e desempenho.

3.1.1 Conclusão da Seção

Na avaliação do estado da arte, identifica-se que as soluções de virtualização devem fornecer, isolamento, enlaces virtuais, virtualização completa, ser distribuído e estar preparado para atender o aumento da escala da rede, pois todos os demais serviços estão em execução utilizando o substrato virtualizado.

A Tabela 1, ordena as principais propostas através das características apresentadas nesta seção e destacando o tipo de abordagem, onde, no proxy o hipervisor atua como controlador da redes física e traduz as informações da rede virtual para o controlador externo. O suporte completo ao *OpenFlow* indica se ele é parcial, ou seja, quanto é possível utilizar apenas uma das versões disponíveis do protocolo ou completo quando é possível utilizar qualquer versão. O isolamento no plano controle é quando há um único ponto para integrar a redes virtual ao

seu controlador. Já o isolamento do plano de dados refere-se a cada rede virtual vinculada a um conjunto de recurso de redes (ex. largura de banda). A característica aberto, refere-se a sua disponibilidade de uso podendo ser aberto ou privado. Por fim, a escalabilidade da solução baseado nos resultados encontrados nos trabalhos relacionados.

Tabela 1 – Comparação dos hipervisores SDN

	Hipervisores				
Características	FlowVisor	VeRTIGO	AdVisor	OpenVirteX	HyperfleX
Abordagem	Proxy	Proxy	Proxy	Proxy	Proxy
Suporte complero ao OpenFlow	Parcial	Parcial	Parcial	Parcial	Parcial
Isolamento no plano de controle	Não	Não	Não	Não	Sim
Isolamento no plano de dados	Sim	Não	Não	Sim	Parcial
Arquitetura Distribuída	Não	Não	Não	Sim	Sim
Aberto	Sim	Não	Não	Sim	Sim
Escalabilidade	Não	Não	Não	Sim	Sim

Com base nos trabalhos relacionados, pode-se concluir o modelo proxy traz várias desvantagens ao desempenho da vSDNs. Nota-se também, que um possível caminho para minimizar os problemas encontrados são diminuir os conjuntos de funções atribuídas aos hipervisores através da decomposição de suas funções (i.e., controle e/ou gerenciamento) entre os membros da arquitetura. Além disso, destacam-se que essas atribuições sejam aplicadas ou distribuídas em pontos mais próximas do elementos de redes ou no próprio plano de dados. No entanto, nos trabalhos não se observa algum novo modelos de *slices*, arquitetura ou orquestração, que possa substituir o modelo de *proxy*, e assim obter vantagens sobre o modelo atual.

Além disso, nenhum dos hipervisores conseguem atender plenamente a todos os requisitos citados anteriormente para fornecer redes virtuais confiáveis e escaláveis. Mas, percebe-se que aqueles hipervisores que utilizam o *FlowVisor* como base da sua solução, no quesito desempenho essas propostas continuam comprometidas, seja na latência ou vazão. Porém, quando as soluções começam a desafogar o hipervisor de funções de gerenciamento ou controle (ex. *OpenVirtex* e *HyperFlex*), o desempenho das mesma vai melhorando em relação as demais propostas. O que levanta uma premissa de que: quanto mais leve e distribuído for o hipervisor, melhor será o desempenho da vSDN.

3.2 Emulação de Redes SDN

Lantz, Heller e McKeown (2010) apresentam o *Mininet*, um emulador de redes que permite habilitar rapidamente protótipos de topologias utilizando *switches* SDN. A solução utiliza *Linux Namespaces* em sua arquitetura, onde há um *namespace root* que contém o processo *Open vSwitch* que é responsável por habilitar os *switches* virtuais que são utilizados nos experimentos. Esse *switches* virtuais são conectados através de pares de interfaces virtuais ethernet (ex., *veth-pairs*) disponíveis no *Kernel* do sistema operacional *Linux*. Além disso, cada *switch* pode se

conectar a vários *hosts*, que são *namespaces* dinâmicos que representam clientes ou servidores na redes emulada. A arquitetura também prover um CLI para que os usuários possam interagir com os nodes da emulação.

O mais popular emulador para SDN, o *Mininet* prover uma ambiente local para a inovação em redes SDN de maneira "portátil". Entretanto, há consideráveis limitações que influenciam a fidelidade no desempenho em cargas altas e escalabilidade quando aplicadas a topologias complexas. Por oferecer uma ambiente de virtualização compartilhado, o Mininet também limita suas características. Por exemplo, ele não suporta diferentes versões de Linux, não há isolamento e limitação de recurso nos namespaces e a vazão em experimentos é limitada, principalmente em cargas elevadas.

Wang, Chou e Yang (2013) propõem o *EstiNet*, um ambiente híbrido que permite experimentos em redes usando tanto simulação (switches e enlaces) quanto de emulação (hosts), com o objetivo de suportar testes de funcionalidade e desempenho em SDN. Suas simulações podem utilizar controladores externos e aplicações reais (ex., serviços http, proxy, ou dhcp) sem a necessidade de modificações no simulador.

Apesar de apresentar uma solução integrada (i.e., simulação e emulação), esta proposta possui questões abertas que à limita. Por exemplo, a plano de dados é simulado, logo seu desenvolvimento e simples e limitado, o que reduz a fidelidade no desempenho da aplicação e pode não replicar ou representar o mesmo comportamento quando aplicado a uma infraestrutura SDN real. Além disso, a expansão e teste de novos protocolos SDN são condicionadas ao desenvolvedores do EstiNet.

Peuster, Kampmeyer e Karl (2018) criaram à solução *Containernet*, uma expansão da solução Mininet, para suportar o uso de contêineres Docker. Esta proposta adiciona uma nova característica que permite ao Mininet inserir e remover contêineres de uma rede emulada. Com isso, a proposta tem como objetivo habilitar serviços de funções de rede (VNF – *Virtuais Networking Functions)* dentro de uma emulação, isto é feito através de imagens Docker[1] instanciadas durante a execução do experimento, permitindo um controle e interação real as VNFs durante a emulação por estar trabalhando em um ambiente isolado.

Apesar de utilizar contêineres, a solução limita-se a nós hosts e não incorporando os *switches* ou outros elementos de rede, o que permitiria solucionar problemas de isolamento, restrição de recurso e disponibilização de novos softwares *switches*. Como resultado avaliações aplicadas através de experimentos podem reproduzir resultados imprecisos.

Em Hibler et al. (2008), tem-se vEmulab um emulador de redes que permite virtualizar recursos de redes (ex., hosts, roteadores e switches) baseados em *BSD jail* (análogo ao *Linux namespace*). O objetivo da solução é permitir alta performance, fidelidade e isolamento e transparências as emulações. A característica chave deste emulador é usar o mínimo de virtualização

[1] Projeto Docker: http://www.docker.com

para construir experimentos de rede mais realísticos. O vEmulab é profundamente automatizado para facilitar o uso do emulador mesmo em condições em que são utilizados milhares de nós.

Na proposta vEmulab observa-se uso de CBEs na emulação de redes. Além disso, ela apresenta a vantagens de se aplicar *BSD jails* (i.e. contêineres) em todos os elementos da emulação que é validado através de experimentos práticos e resultados. Porém, a proposta se limita apenas a sistemas BSDs, o qual restringe o uso e suporte à soluções SDN.

3.2.1 Conclusão da Seção

Nos trabalhos relacionados desta seção, observa-se várias propostas de ambientes para avaliação e emulação de redes SDN. No entanto, destacam-se como as principais características: a falta de escalabilidade, fidelidade no comportamento da rede, isolamento do componentes da emulação, suporte a multi-ambientes a sistemas operacionais e escalabilidades. Na Tabela 2, têm-se um sumário comparativo das soluções abordadas.

Tabela 2 – Quadro de comparativo dos Emuladores de redes SDN

	Emuladores			
Características	Mininet	EstiNet	Containernet	vEmulab
Isolamento de recurso do contêiner	Não	Não	Sim	Sim
Limitação de recurso do contêiner	Não	Não	Não	Sim
Escalabilidade	Não	Sim	Não	Sim
Múltiplos ambientes operacionais	Não	Não	Parcial	Não
Suporte a protocolos SDN	Sim	Parcial	Sim	Não
Suporte a protocolos legados	Não	Não	Não	Sim
Comportamento realístico	Sim	Não	Sim	Sim

Analisando o resultado da Tabela 2, evidencia-se a necessidade de soluções mais completas para emulação de redes SDN. Apesar do vEmulab possuir as características necessárias, sua arquitetura é limitada e não adequa ao escopo desta proposta devido a falta de suporte aos protocolos SDN, porém, ela demonstra o uso de contêiner isolados a cada elemento da topologia trazem vantagens para emulador, tanto no desempenho quanto na facilidade e realismo.

Já as demais soluções referenciadas, são fortemente dependentes do Mininet resultando na herança dos problemas encontrados na solução base. Isto compromete diretamente coleta de resultados gerados, pela falta de escalabilidade e realismo da emulação. Então, conclui-se a necessidades de uma ferramenta escalável e mais realista em suas emulações.

3.3 Conclusão do Capítulo

Este capítulo teve como ênfase apresentar uma revisão do estado da arte sobre os pontos levantados nas questões de pesquisas principal e suplementar, apresentadas no Capítulo 1. Com

o objetivo de esclarecer a necessidade dos temas abordados e obter uma base para elaboração das respostas as questões de pesquisa.

Através da análise dos trabalhos apresentados, identifica-se a necessidade de uma arquitetura mais leve para melhorar os desempenhos das vSDNs. A possível maneira de fazer isso é minimizando as funções agregadas pelos hipervisores SDN e distribuí-las para mais perto do plano de dados. Além disso, observou-se também que essa desagregação pode trazer vantagens as vSDNs, principalmente no que diz respeito a seu desempenho. Adicionalmente, nota-se a carência de ferramentas de emulação que permita retratar de forma mais fiel possível a emulação de redes SDNs. E com isso aplicar o ambiente proposto com objetivo de analisar seus desempenhos em relação a outras propostas.

4 UMA ARQUITETURA LEVE PARA PROVISIONAMENTO DE REDES VIRTUAIS DEFINIDAS POR SOFTWARES

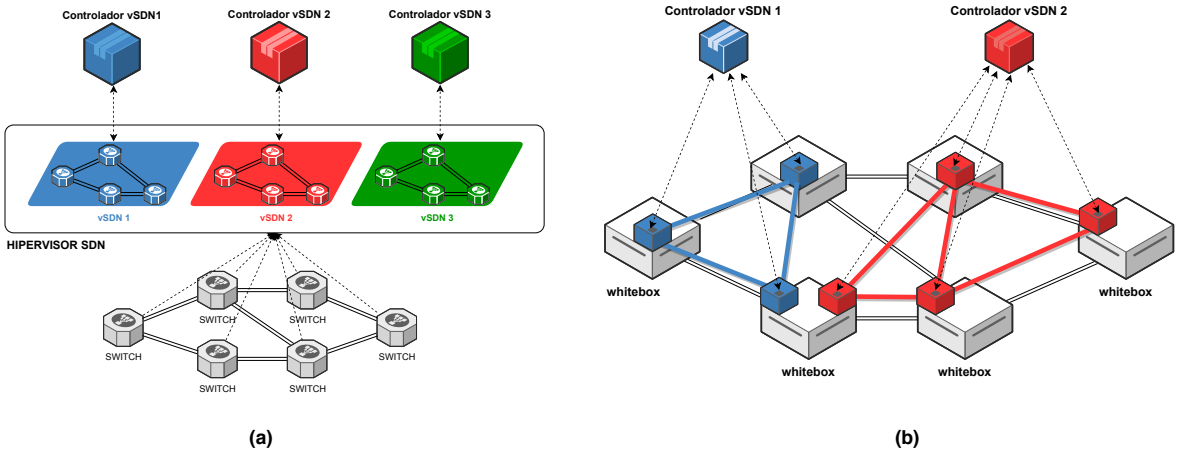
Este capítulo apresenta o *vSDNLight*, uma proposta de arquitetura capaz prover vSDNs de forma leve e escalável. A arquitetura da proposta é constituída por três pilares: (i) *whiteboxes* da infraestrutura; (ii) visão de slices baseados em VSIs; e (iii) a proposta de agente (*vSDNAgent*) e orquestrador (*vSDNOrches*). Adicionalmente, neste capítulo também se apresenta o *vSDNEmul*, uma ferramenta de suporte para emular ambientes de redes SDN e permitir a execução da arquitetura *vSDNLight*, e com isso responder a questão suplementar.

Conforme contextualizado no Capítulos 3, os problemas de desempenhos encontrados nas vSDNs estão relacionados ao modelo de proxy adotado pelas soluções e na sobrecarga de incumbências atribuídas aos hipervisores SDN. Nossa hipótese é que as limitações encontradas nas vSDNs podem ser reduzidas através de uma arquitetura leve e distribuída para o provisionamento de vSDNs. A redução de atribuições também contribuirá para obter resultados melhores.

4.1 *vSDNLight*

O *vSDNlight* é uma arquitetura leve para prover redes virtuais definidas por softwares, que através de agentes espalhados pela infraestrutura reduz os *overheads* que ocasionam problemas e que comprometem tanto a escalabilidade do hipervisor quanto das vSDNs. Por isso, a arquitetura adota um novo formato de criação de *slices* em redes SDN que elimina necessidade do *proxy* entre controlador da redes virtual e a infraestrutura. A Figura 10 mostra a diferença do modelo de *proxy* e o novo modelo proposto.

Figura 10 – Diferença entre o modelo de *slice* tradicional (a) e o adotado na tese (b).



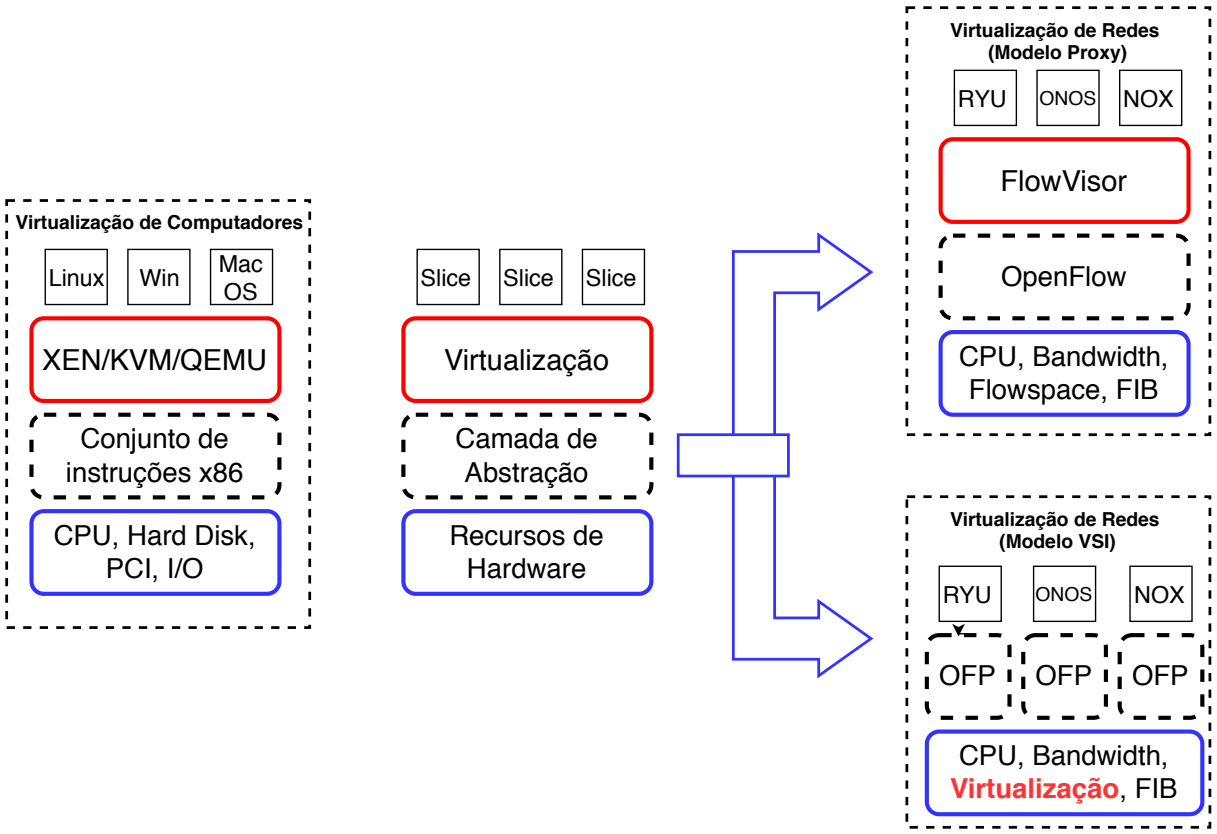
Fonte: Autor.

No modelo tradicional, ilustrado na Figura 10 (a), o hipervisor tem que tratar e identificar todas as mensagens recebidas de cada *switch* participante no *FlowSpace*. Logo depois,

essas mensagens são reescritas e entregues ao controlador responsável desta redes virtual. Na proposta de *proxy* a tabela de encaminhamento e o plano de controle são compartilhada entre os *FlowSpaces*. Além de ser um ponto de falha, isso compromete a escalabilidade, pois dependendo do número de redes virtuais essa tabela pode ser rapidamente congestionada de regras, o que inviabilizaria a sua utilização.

No modelo proposto, ilustrado pela Figura 10 (b), tem-se o *slice* como o conjunto de instâncias de *switches* virtuais (VSI) alocado no plano de dado. Além disso, a proposta é leve, pois não há tratamento de mensagens de controle pelo orquestrador e sim apenas o gerenciamento do ciclo de vida e monitoramento das redes virtuais. Eliminando assim, a camada de virtualização no plano de controle e distribuindo-a para mais próximo do plano de dados através dos *whiteboxes* SDN. Além disso, o orquestrador coordena agentes incluídos nestes *whiteboxes* da infraestrutura para tratar apenas das abstrações de *switches* virtuais e filtros de fluxos para portas ou túneis, só que parametrizado em cada nó da infraestrutura, diminuindo a sobrecarga de controle e gerenciamento no sistema como um todo. Desta forma, o controlador do usuário atua diretamente com o *switch* virtual. Figura 11 ilustra a comparação estrutural entre os dois modelos.

Figura 11 – Modelo baseado em proxy vs Modelo baseado em VSI.



Fonte: Adaptado de Sherwood et al. (2009).

Neste caso, observa-se na Figura 11 que o modelo VSI migra a camada de virtualização como componente da camada de recursos de hardware, através de elementos virtuais como: *switches*, portas ou enlaces virtuais (ex. *VLAN*, *VXLAN*, *GRE* ou *GENEVE*). Esses elementos

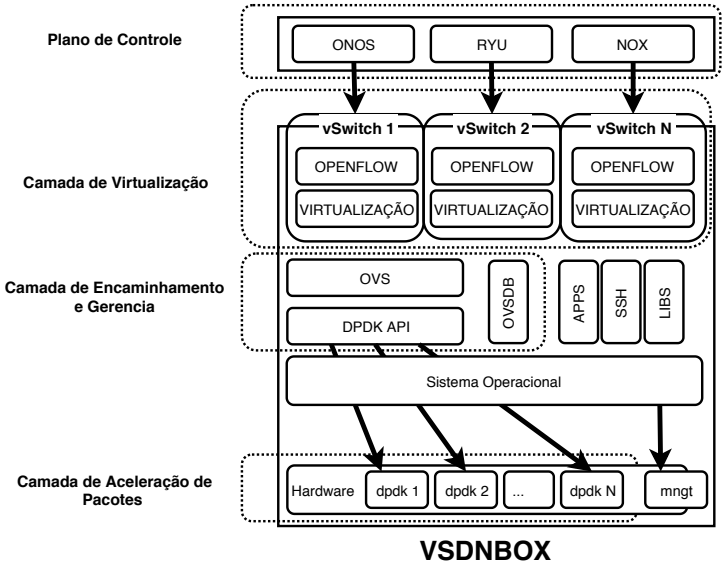
são abstraídos pelo *OpenFlow* sem a necessidade de modificações no protocolo.

4.1.1 *Whitebox* SDN

Um *whitebox* é um switch que possui um conjunto mínimo de softwares instalados, onde muitas das vezes, somente o sistema operacional e bibliotecas são de responsabilidade do administrador ou desenvolvedor. A vantagem disso é a diminuição da complexidade e os custos agregados com softwares controladores ou protocolos instalados no equipamento, que com a adoção do SDN deixam de ser uma obrigatoriedade dentro dos mesmos por estarem localizados em um elemento externo.

O *vSDNBox* é uma PoC (*proof-of-concept*) de *whitebox* baseado *software-switched*, ou seja, todos componentes são definidos em softwares, por exemplo, a aceleração de pacote é provido por um framework de aceleração de pacotes via software (ex. DPDK[1] – *Data Plane Development Kit*). Além disso, seu hardware é genérico o suficiente para suprir qualquer problema de incompatibilidade. O objetivo é criar uma "caixa" gerenciável por qualquer protocolo aberto SDN com desempenho de encaminhamento igual ou aproximado a um *whitebox hardware-switched*, ou seja, componente baseado em *hardware ASIC* (Application Specific Integrated Circuits). E aberto para facilitar instalação, configuração e atualização de ferramentas, protocolos e características. Adicionalmente, ele possui uma camada de virtualização que permite a criação de várias VSIs.

Figura 12 – Arquitetura interna do *vSDNBox*.



Fonte: Autor.

Na arquitetura do vSDNBox, usou-se apenas APIs abertas para facilitar o aprimoramento, evolução e a melhora do desempenho alcançado. Sendo assim, a Figura 12 apresenta um visão geral da arquitetura definida para *vSDNBox*. Nesta figura, é possível observar 3 níveis de

[1] Projeto DPDK: https://www.dpdk.org

implementações aplicadas à proposta, onde pode-se destacar, a camada de aceleração de pacotes, a camada de controle de encaminhamento e a camada de protocolo de gerenciamento.

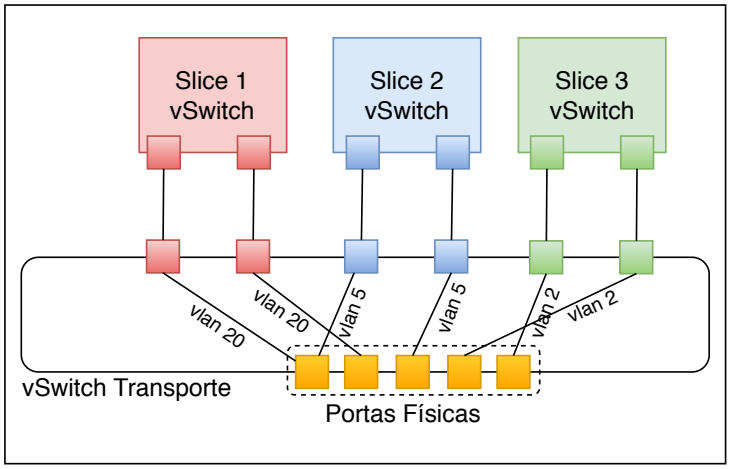
A camada de aceleração de pacotes é responsável pela gerência dos recursos de cada interface de rede, como: quais núcleos são responsáveis por processar as filas de entrada e saída e/ou memória disponíveis para caches de dados. Nesta iniciativa, a aceleração de pacotes é feita por núcleos do processador, e ignorando algumas camadas de virtualização e de kernel. O resultado disso é observado na ampliação da vazão, e na diminuição da latência de processamento de pacotes nas interfaces de redes (FARIAS et al., 2018). Para aceleração de pacotes via software optou-se pela API *DPDK* devido ao fato de ser uma solução que possui uma maior compatibilidade com as diversas soluções SDN, como também, apresenta o suporte a várias interfaces de redes genéricas.

Já na camada de controle de encaminhamento e gerência, aplica-se as decisões ou ações nas interfaces aceleradas via software, por exemplo, descarte de pacotes, encaminhamento para outra interface física ou virtual, ou até a retenção da taxa de transmissão nas filas de entrada e saída das portas. Esta camada faz uso do *Open vSwitch* (OVS) que é uma ferramenta capaz de controlar toda a dinâmica de encaminhamento de pacotes, a partir de comandos oriundos de um controlador SDN, como também, a criação e remoção de características de virtualização equipamento, por exemplo, um switch ou porta virtual. Nesse contexto, optou-se pelo OVS por ser compatível com vários protocolos tradicionais (ex. *Ethernet*, *GRE* e *IPv4*), bem como novos protocolos (ex. *VXLAN*, *Geneve*, *NVGRE* e *STT*).

Finalmente, a camada de virtualização que o resultado de configurações produzidas pela camada de encaminhamento e gerência. Basicamente, esta camada comporta as VSIs que possuem um componente de controle, neste caso o *OpenFlow*, e os componentes de hardware virtual (ex. interfaces e enlaces virtuais). A instância de switch virtual alocado nesta camada contém todos os seus recursos isoladas das demais instâncias. Logo, ele possui portas independentes, tabela de encaminhamento isolada e controle isolado.

Na Figura 13, observa-se um exemplo da estrutura interna do whitebox e como são feitos os mapeamentos dos switches virtuais (*vSwitch*). Internamente, a estrutura é composta de um *vSwitch* de transporte contendo todas as portas físicas disponíveis e *N vSwitch* vinculados a *N slices*. No exemplo da Figura 13, têm-se três *vSwitch* (vermelho , azul e verde). Logo, para cada porta virtual no *vSwitch* do *slice* vermelho haverá uma porta virtual correspondente no *vSwitch* de transporte. No *vSwitch* de transporte é feito o mapeamento para porta física através de regras *OpenFlow*, que filtrarão o tráfego das portas do *vSwitch* do *slice* para porta do *vSwitch* de transporte e vice-versa, no caso do slice vermelho, os fluxos pertencentes ao mesmo serão filtrados pelo identificador da VLAN número 20, ou seja, todos fluxos que chegarem com VLAN 20 serão encaminhados para as portas do *vSwitch* vermelho.

Figura 13 – Mapeamento interno dos VSIs para os slice.

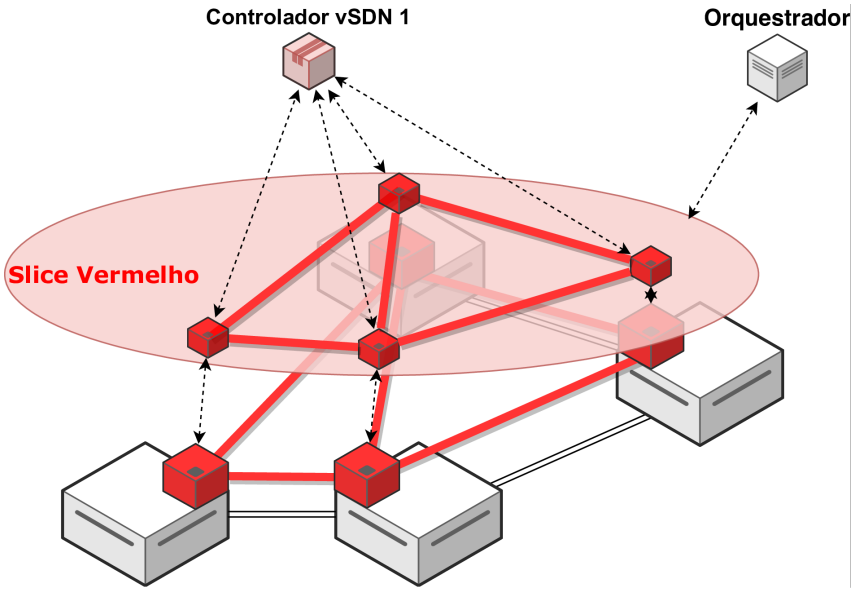


Fonte: Autor.

4.1.2 Slice Baseado em VSIs

O *slice* baseado em VSIs é um novo modelo de construção de *slice* aplicado pela arquitetura *vSDNLight*. Logo, dado uma infraestrutura de *whiteboxes* SDN, o *slice* é a composição de uma ou mais instâncias virtuais que possuem uma mesma característica de fluxo, por exemplo, fluxos com o identificador de VLAN igual à 20. Então, uma vez aplicado esse slice a infraestrutura, o mesmo é orquestrado do plano de gerenciamento e não no plano de controle, conforme , ilustrado na Figura 14.

Figura 14 – Exemplo do modelo de Slice proposto usando VSIs.



Fonte: Autor.

Nesta Figura 14, a elipse vermelha ilustra os recursos virtuais alocados infraestrutura física para *slice* e também sua localização. No entanto, os *whiteboxes* não tem noção do tamanho

da topologia a qual ele pertence, o único elemento que tem esta noção é justamente o orquestrador que é responsável tratar o ciclo de vida do *slice* e solicitar aos agentes a criação ou remoção das instâncias de *switches* virtuais.

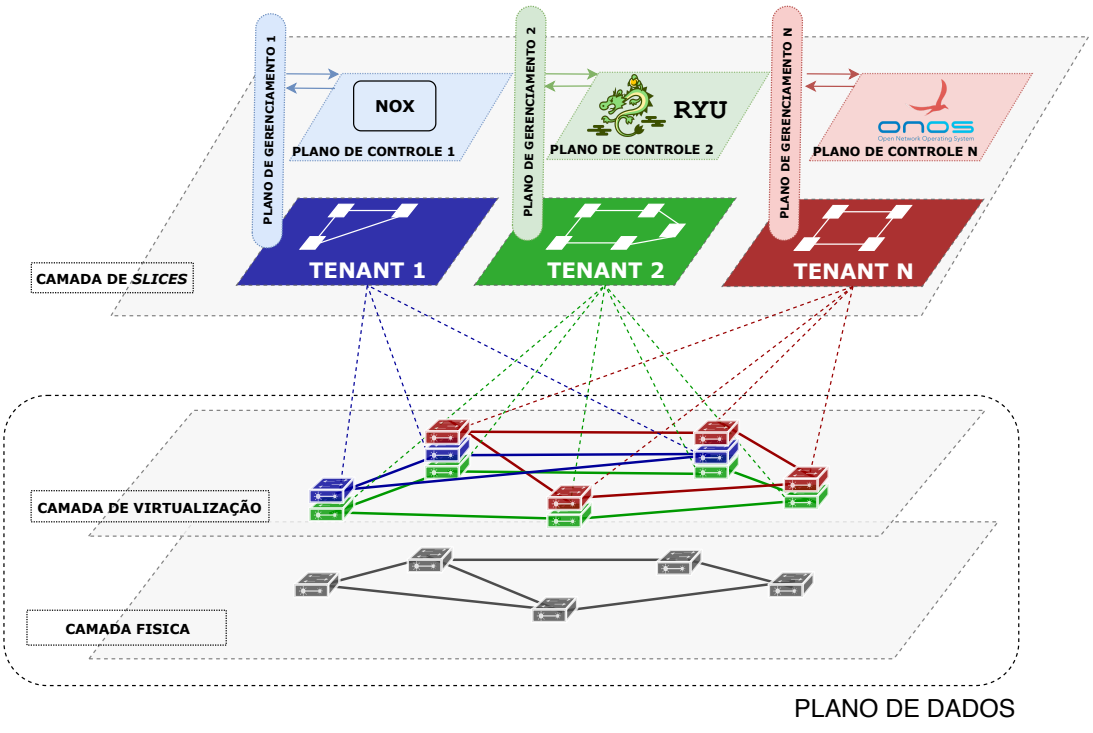
Com a remoção da camada de virtualização do plano de controle, a camada de tradução (i.e., camada de *proxy*) não é mais necessária, de forma que os *switches* pertencentes a um determinado *slice* entregam seus eventos diretamente ao controlador do slice e vice-versa. Além disso, o controle do *slice* também não é mais necessário, de modo que a simplificação dessa atividade foi migrado para o plano de gerenciamento da arquitetura SDN.

Além disso, a proposta ainda define dois modelos de operação do *slice*: no primeiro modo, o controle é de responsabilidade do *tenant* ou usuário do *slice*, ou seja, o *slice* é controlado pelo sistema operacional de rede (NOS – *Networking Operating System*) atribuído pelo *tenant*, o qual define o controlador que *slice* será vinculado e que possui suas próprias regras de negócios e um conjunto de aplicações e protocolos. O outro modo, o controle do *slice* feito por uma aplicação de controle com um comportamento pré-definido da rede virtual e APIs com funções especiais de controle do *slice* são disponibilizadas externamente.

4.1.3 Arquitetura do *vSDNLight*

Na Figura 15 é possível observar uma visão de alto nível da arquitetura do *vSDNLight*, que está dividida em três camadas: camada física, camada de virtualização e camada de *slices*.

Figura 15 – Visão em alto nível do vSDNLight.



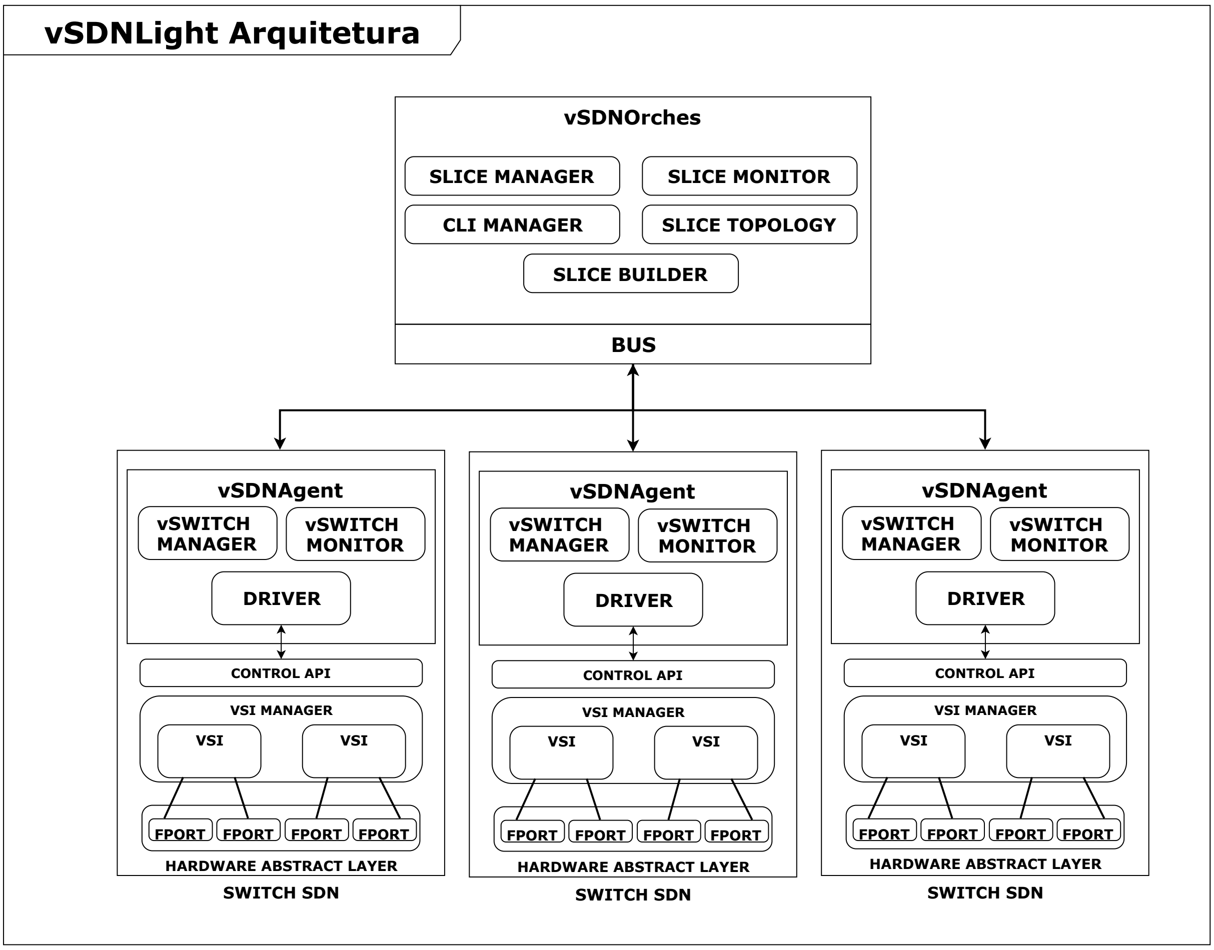
Fonte: Autor.

Na camada física fazem parte todos elementos responsáveis por encaminhar dados pela infraestrutura física, tais como, *switches* ou roteadores. No contexto da proposta essa camada é composta pelo componente *whitebox*, onde o *vSDNBox* representa o elemento de comutação na infraestrutura com a funcionalidade de virtualização de *switch*.

A camada de virtualização contém o conjunto de instâncias virtuais de *switches* gerenciadas por um componente chamado de *vSDNAgent*. Esse componente encontra-se inserido nos *whitebox* da infraestrutura, sendo responsável por negociar a alocação e deslocação de recursos de *switch* virtual junto a tecnologia de virtualização do *switch*.

A camada de *slice* é uma camada de software representado pelo *vSDNOrches* que mantém as informações do ciclo de vida dos *slices*. Ele é responsável por construir toda estrutura de alto nível do *slice*, definida pelas necessidades do usuário, e também solicitar instanciação do *slice* para as camadas inferiores.

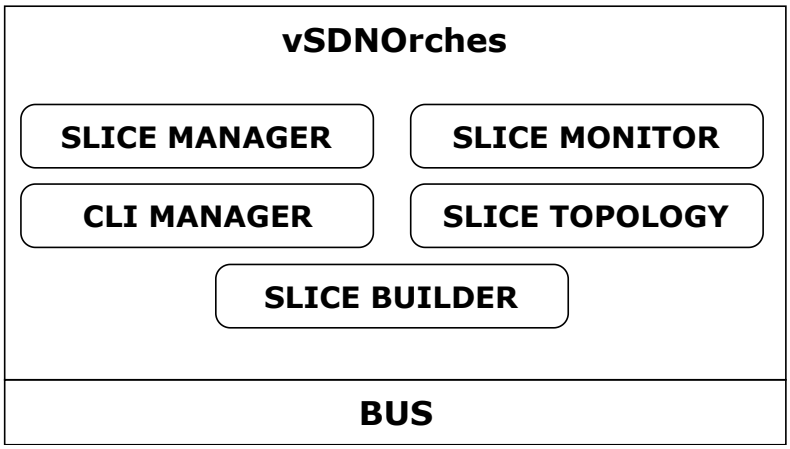
Figura 16 – Arquitetura do *vSDNLight* e seus componentes.



Fonte: Autor.

Na arquitetura do *vSDNLight*, ilustrada na Figura 16, é possível observar os componentes responsáveis pela gerência dos recursos de virtualização disponíveis na infraestrutura. Esta arquitetura contém dois elementos essenciais para funcionamento da orquestração: *vSDNOrches* e *vSDNAgent*.

Figura 17 – *vSDNOrches* e seus componentes.



Fonte: Autor.

O *vSDNOrches*, apresentado na Figura 17, é responsável por fazer toda a orquestração dos *slices* e mapeá-los em forma de VSIs. Além disso, ele também mapeia a interconexão entre os VSIs através de enlaces virtuais (ex., *VLAN* ou *VxLAN*) disponíveis nos *whiteboxes* da infraestrutura. A estrutura interna dele é composta pelos seguintes componentes: *Slice Manager*, *Slice Monitor*, *Topology Manager*, *CLI Manager*, *Bus* e *Slice Builder*.

Slice Manager é responsável administrar a criação, remoção e atualização dos *slices* e *tenants* solicitados pelos usuários. Neste componente, obtem-se as informações topológicas dos *slices* que estão criadas ou vão ser criados na infraestrutura. O slice é uma estrutura de dados (ex. *XML* ou *JSON*), armazenadas no componente *Slice Topology*, que contém a descrição e *status* dos mesmos. Essas estruturas contém informações do *switches* virtuais, por exemplo, onde os VSIs devem ser criados e em quais *switches* físicos eles serão criados, e também, informações dos enlaces virtuais aplicadas na criação de portas virtuais incluídas nesses VSIs.

Slice Monitor permite ao orquestrador a capacidade de monitorar elementos virtuais e físicos (i.e., *slices*, *switches* ou portas). Além disso, ele responsável por interpretar as mensagens dos eventos trocados entre esse componente e os agentes (*vSDNAgent*) a respeito de falhas, status (ex. falhas no *switch* virtual ou portas virtuais/físicas) ou consultas diretas de informações disponíveis nos agentes.

Topology Manager é responsável por fazer à persistência dos dados da infraestrutura física e das topologias virtuais. Ele pode baseado em banco de dados não relacional, seja em *JSON* ou Grafos. Como as informações a serem tratadas são praticamente topologias, então essas estruturas facilitaria a modelagem e otimizações. Ele também, oferece algumas ferramentas de manipulação dos dados da topologia, além, de oferecer métodos de descoberta de recursos a serem utilizados pelo demais componentes.

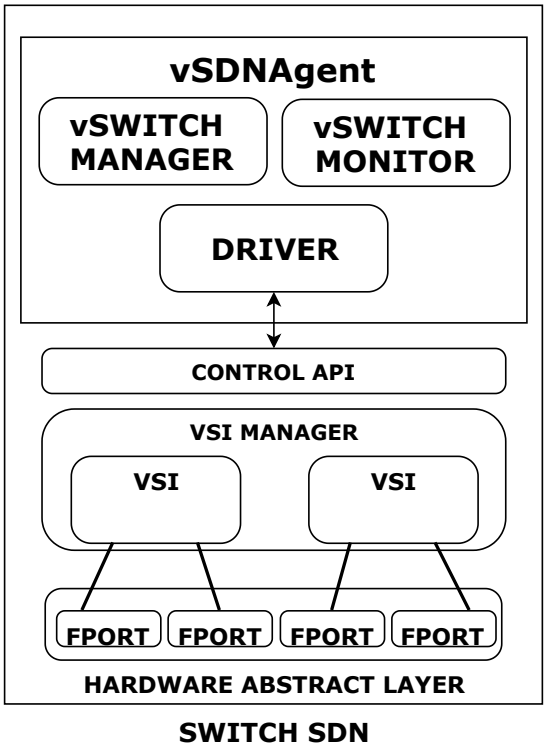
CLI Manager é a interface primária com usuário onde é possível desenhar a topologia da rede virtual através do *Slice Manager* para ser aplicada na infraestrutura física. Como também

é possível consultar e gerenciar um *slice* ou elemento físico, por exemplo, iniciar ou parar um *slice*, criar um enlace entre dois VSIs ou configurar a topologia física.

Slice Builder é responsável por coordenar *vSDNAgents* que se registram no orquestrador. A principal tarefa deste componente é negociar junto a infraestrutura os recursos a serem alocados para os *slices*. Essa negociação é iniciada pelo *Slice Manager* que repassa *Slice Builder* toda a descrição do *slice* que deve ser construído ou destruído. Neste caso, este componente analisa as informações enviadas e solicita aos agentes (*vSDNAgent*) a construção dos elementos virtuais. Além disso, este componente também pode negociar a construção de enlaces virtuais através da configuração de compartilhamentos do enlace físico podendo ser implementado pelos seguintes protocolos: 802.1q (*VLAN*), 802.1ad (*QinQ VLAN*) ou túneis virtuais (ex., *VxLAN* ou *GRE*).

Bus é um componente responsável por coordenar a mensagens síncronas e assíncronas entre o orquestrador e seus agentes. Esse componentes pode ser baseados em frameworks como: *RESTful*[2] , *WAMP*[3] ou *KAFKA*[4].

Figura 18 – *vSDNAgent* e seus componentes.



Fonte: Autor.

O *vSDNAgent*, ilustrado na Figura 18, é um software que administra o ciclo de vida das instâncias virtuais pertencentes aos *slices*, e também as suas configurações. A arquitetura do *vSDNLight* através do *vSDNAgent* migra a camada de virtualização para o plano de dados com

[2] Projeto REST: http://w3.org/2001/sw/wiki/REST
[3] Projeto WAMP: http://wamp-proto.org
[4] Projeto KAFKA: http://kafka.apache.org

objetivo de minimizar a sobrecarga dessas operações, antes, aplicada no plano de controle. Isto é feito através do provisionamento da instância virtual no *vSDNBox* e as configurações para filtrar os fluxos pertencentes ao slice da qual este VSI faz parte. Logo, o processamento da localização do *slice* ao qual corresponde o fluxo encontrado deixa de ser centralizado e passa a ser distribuído em pequenas partes nos *vSDNBoxes*.

Desta forma, quebra-se a necessidade do controle global desses elementos, que dependendo da escala da rede, acaba sobrecarregando o hipervisor e se tornando um elemento de falha. O resultado desta mudança é caracterizado na simplificação nas operações de controle tornando a arquitetura mais simples, por isso que *vSDNLight* é chamado de arquitetura leve, pois a visão agora é local e a escala é limitada pelos números de VSIs alocados, no entanto, o resultado do slice é global. O *vSDNAgent* é composto de três componentes: *vSwitch Manager*, *vSwitch Monitor* e o *Driver*.

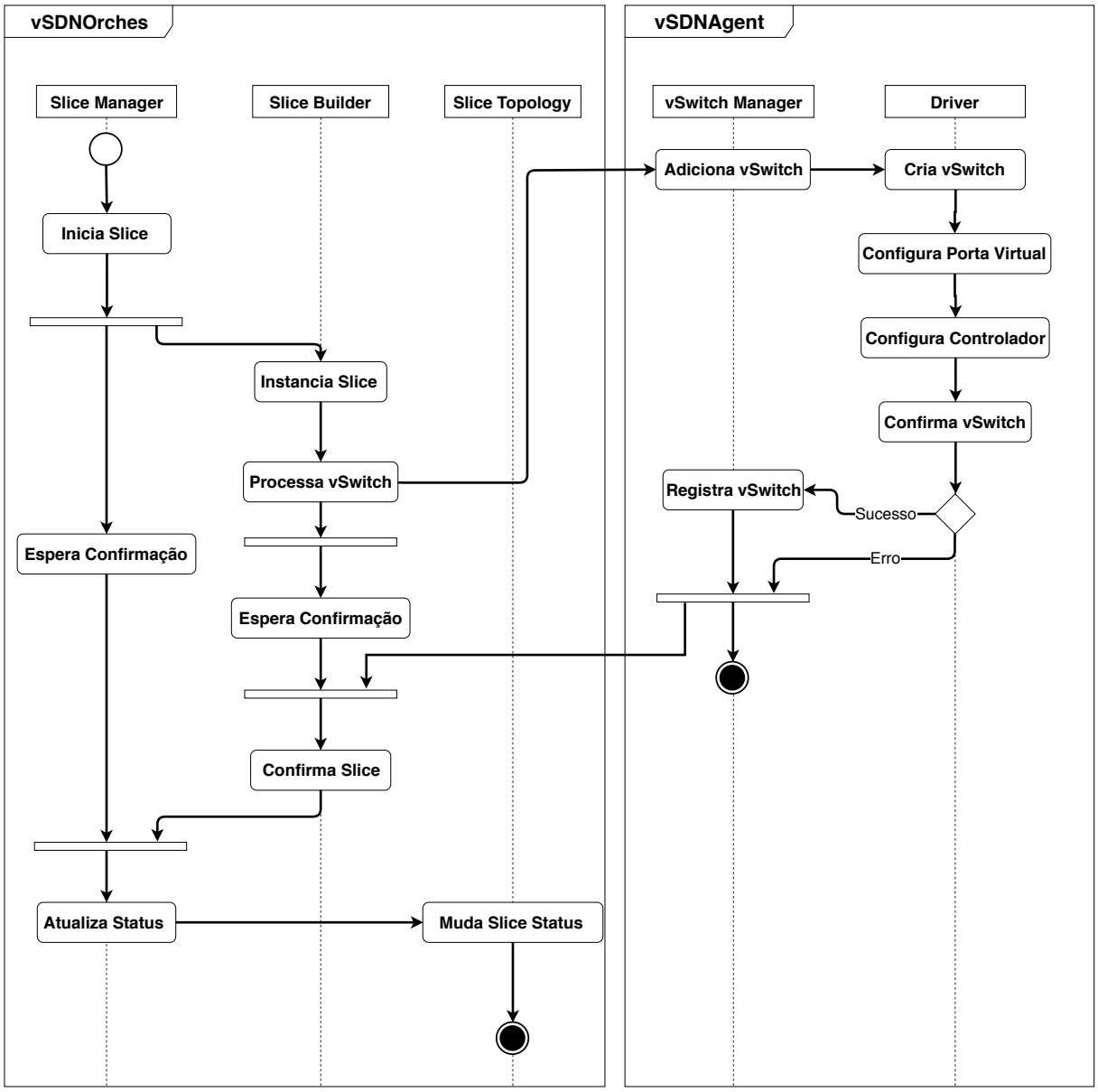
vSwitch Manager é componente responsável por solicitar à criação, remoção e atualização dos VSIs através do componente *Driver* que é encarregado negociar com *VSI Manager*, que prover a tecnologia de virtualização no *whitebox* (ex., *Open vSwitch*). Ao iniciar, este componente verifica se o *Driver* está disponível, após isso, ele se conecta ao orquestrador que registra-o como *switch* disponível para alocação. Ele também faz integração das portas virtuais com as portas físicas, configuração dos protocolos a serem utilizados no VSI e os filtros de fluxos administrados por esse VSI.

vSwitch Monitor monitora o comportamento dos VSIs e encaminha para o orquestrador, através de mensagens síncronas e assíncronas. As mensagens assíncronas, basicamente, são eventos ocorrentes no *switch* como: mudança de status de switches, mudança de status, inclusão ou remoção de portas ou falhas no *VSI Manager*. Por outro lado, as mensagens síncronas, são consultas vindas do orquestrador como estatísticas do VSI, fluxos ou portas.

Driver é responsável por fazer a comunicação com a tecnologia de virtualização. Ele tem uma implementação especifica para cada tipo de VSI Manager. O objetivo é permitir que *vSDNAgent* suporte qualquer tecnologia de virtualização de *switch* e com isso permitir que a proposta alcance a maior quantidade de *switches* possíveis ou fazer com que o fabricante possa desenvolver a sua proposta de *vSDNAgent*.

Quanto ao posicionamento da arquitetura de acordo com modelo SDN, apresentado no Capítulo 2, essencialmente estão localizado no plano de gerenciamento e plano de dados. No plano de dados estão os *vSDNAgent* interagindo diretamente com *whitebox*. Já o *vSDNOrches*, localiza-se no plano de gerenciamento, pois suas únicas atribuições são de configuração e monitoramento da infraestrutura SDN. De acordo com o que conceitua o documento de padronização de camadas SDN, definido por Haleplidis et al. (2015) através da RFC 7426, apesar do *vSDNAgent* executar pequenas ações de controle, ele não o faz com mesma intensidade de um controlador no plano de controle, por conta disso, pode-se localizar o *vSDNOrches* no plano de gerenciamento.

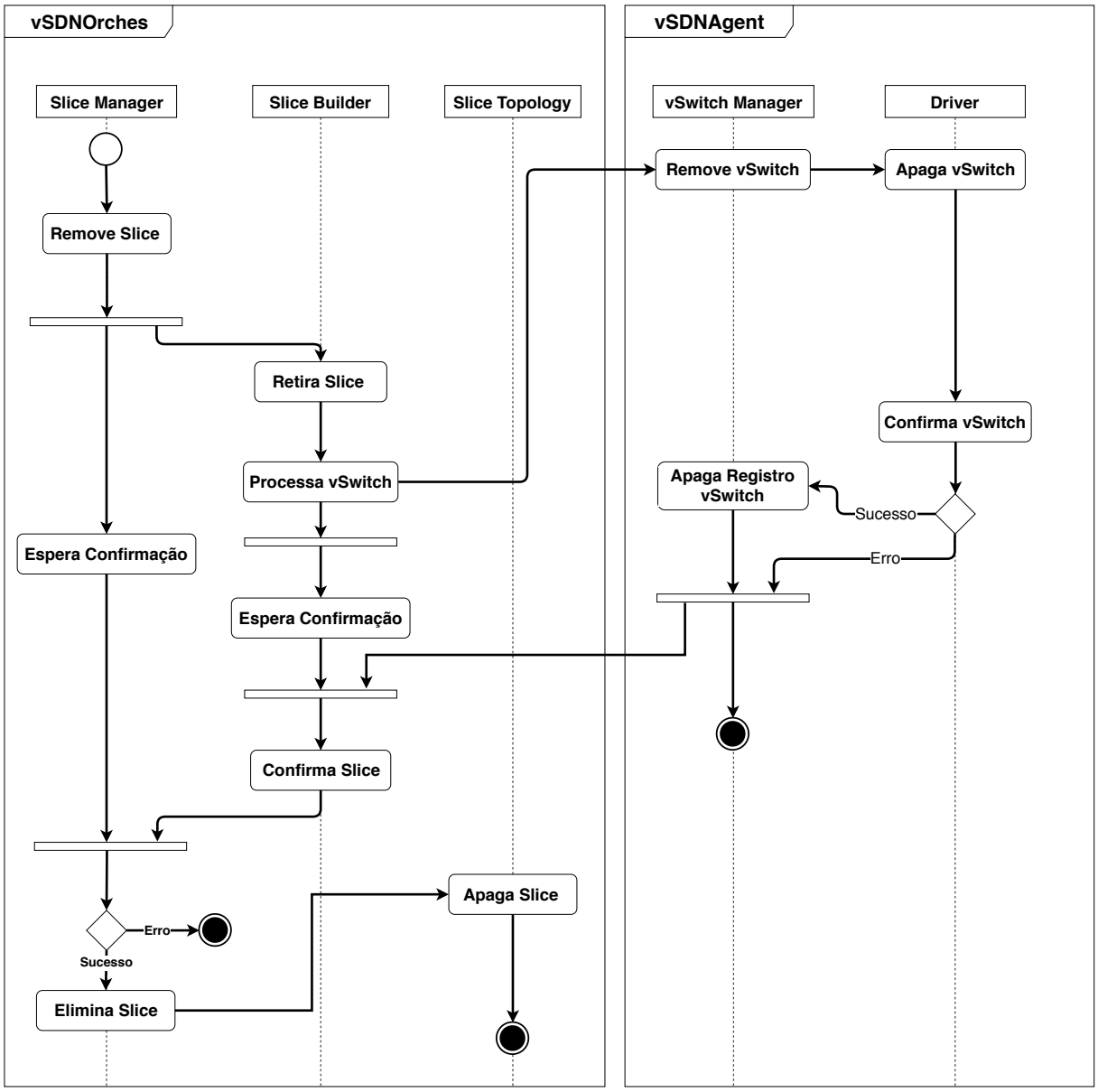
Figura 19 – Diagrama de atividades para criação do *slice*.



Fonte: Autor.

Na Figura 19 é apresentado o diagrama de atividades para a criação do slice na arquitetura *vSDNlight*. Neste caso, as ações são iniciadas no orquestrador pela a solicitação de inicialização do slice, este encaminha ao *Slice Builder* a criação do slice, nesse processo ele verifica os *switches* virtuais e onde devem ser criados. Após isso, o *Slice Builder* solicita aos *vSDNAgent* a criação e configuração dos *vSwitches* que repassa ao componente *Driver* para negociar com *VSI Manager*. Uma vez criado o *switch* virtual os *vSDNAgent* confirmam ao orquestrador a criação do VSI, que consequentemente, confirma ao *Slice Manager* a inicialização do *slice*, e por fim, atualiza o status do *slice* no *Slice Topology*.

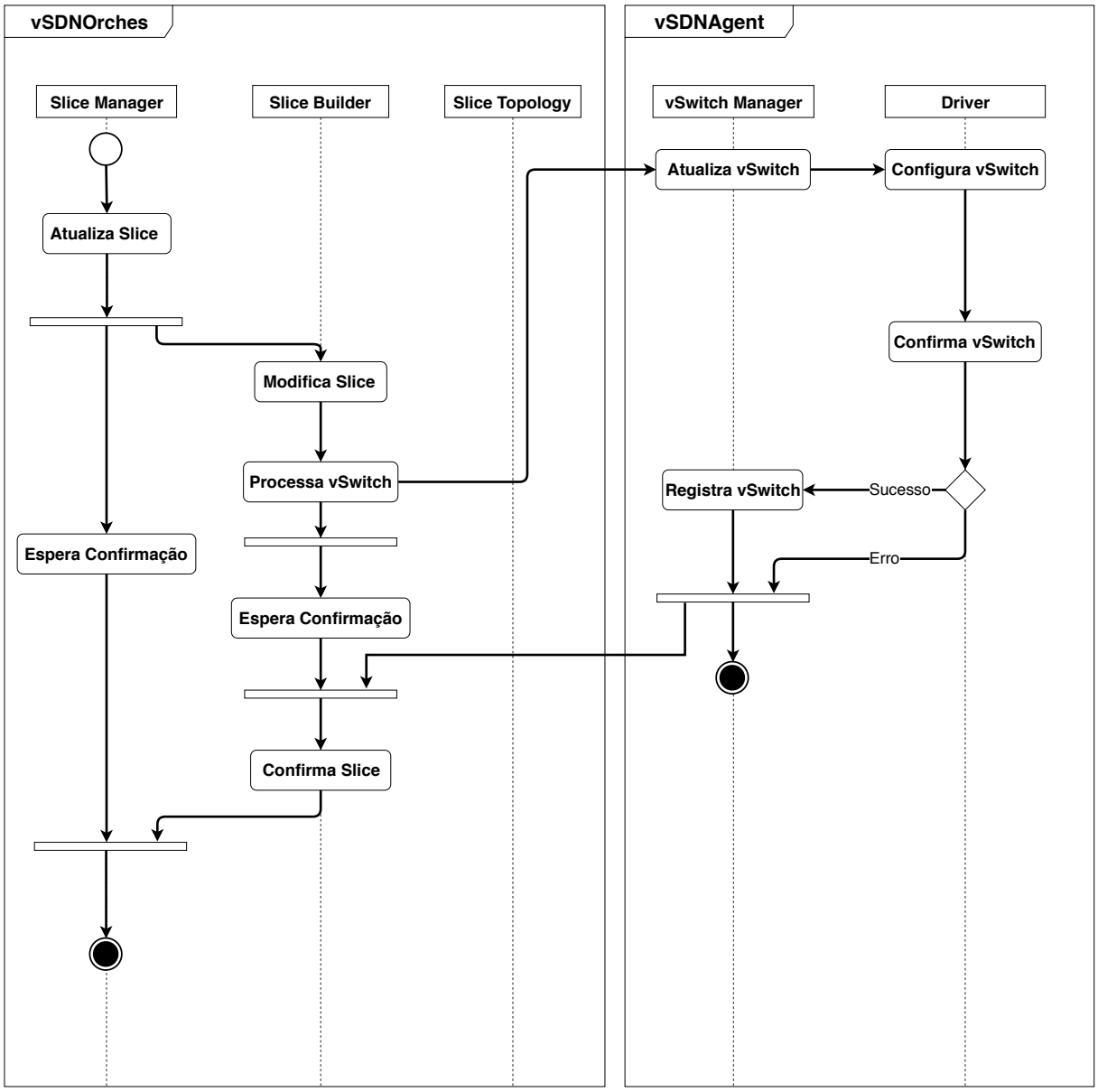
Figura 20 – Diagrama de atividades da remoção do *slice*.



Fonte: Autor.

A Figura 20 mostra o diagrama de atividade do processo de remoção do slice. Esta atividade inicia-se com a solicitação ao *Slice Manager* de remoção. Após isso, o componente envia a solicitação para o *Slice Builder* que processa os *vSwitches* que serão removidos da infraestrutura física, para a eliminação do slice. Uma vez processado, o *Slice Builder* solicita aos *vSDNAgent* a remoção dos switches virtuais. Que por sua vez confirma ao orquestrador a remoção dos mesmo. Então, o *Slice Builder* confirma a remoção do *slice* ao *Slice Manager* que solicita a remoção das informações topológicas da rede virtual no *Slice Topology* ou para o processamento em caso de erro durante o processo de remoção.

Figura 21 – Diagrama de atividades da atualização do *slice*.



Fonte: Autor.

Por fim, tem-se na Figura 21 o diagrama de atividade da atualização do slice. Novamente, o processo inicia-se com a solicitação de atualização ao *Slice Manager*, que pede ao *Slice Builder* para processar a atualização. Basicamente, ele verifica que *switches* virtuais foram modificados e solicita aos *vSDNAgent* específicos a atualização de configuração desses *switches* virtuais. Que por sua vez, confirma ao orquestrador a atualização dos elementos virtuais solicitados.

A seção apresentou o *vSDNLight*, uma arquitetura para o provisionamento de vSDN, caracterizado pela definição de um modelo de slices baseado em VSIs que elimina traduções intermediárias no plano de controle e migra a camada de virtualização para o plano de dados. Além disso, mostrou-se a proposta de orquestração deste novo modelo slice através de dois componentes *vSDNOrches* e *vSDNAgent*. Eles permitiram simplificar o processo de gerência das vSDNs, tornando-o uma arquitetura leve, pois "enxugou" as ações de orquestração e controle e

as distribuiu para mais próximo do dispositivos de redes da infraestrutura.

4.2 *vSDNEmul*

A resposta à questão de pesquisa suplementar: Como emular e avaliar essas soluções leves? É dado pelo *vSDNEmul*. Ele atua emulador de rede baseado em virtualização por contêiner. O objetivo do desenvolvimento deste emulador partiu das deficiências apresentadas pelo *Mininet* e outros emuladores de rede, apresentados no Capítulo 3, por possuírem uma arquitetura que limita tanto o escopo dos experimentos quando a fidelidade dos testes. Consequentemente, a serialização, contenção e carga do processos em segundo plano podem produzir atrasos que comprometam a operação de eventos, tais como: a transmissão do pacote ou a execução de um processamento, possibilitando até invalidar a avaliação de desempenho de um experimento.

Apesar de alguns avanços na emulação de redes SDN, ainda há desafios abertos como: a necessidade de aumentar a confiabilidade dos dados coletados em avaliações de desempenho, oferecer variados dispositivos emulados, utilizar vários protocolos SDN durante a emulação, utilizar softwares e protocolos mais atualizados, permitir múltiplos sistemas operacionais (SO) entre os nós, permitir isolamento real de recurso (ex. cpu, memória ou banda) para qualquer tipo nó e disponibilizá-lo de maneira pública e aberta (KREUTZ et al., 2015).

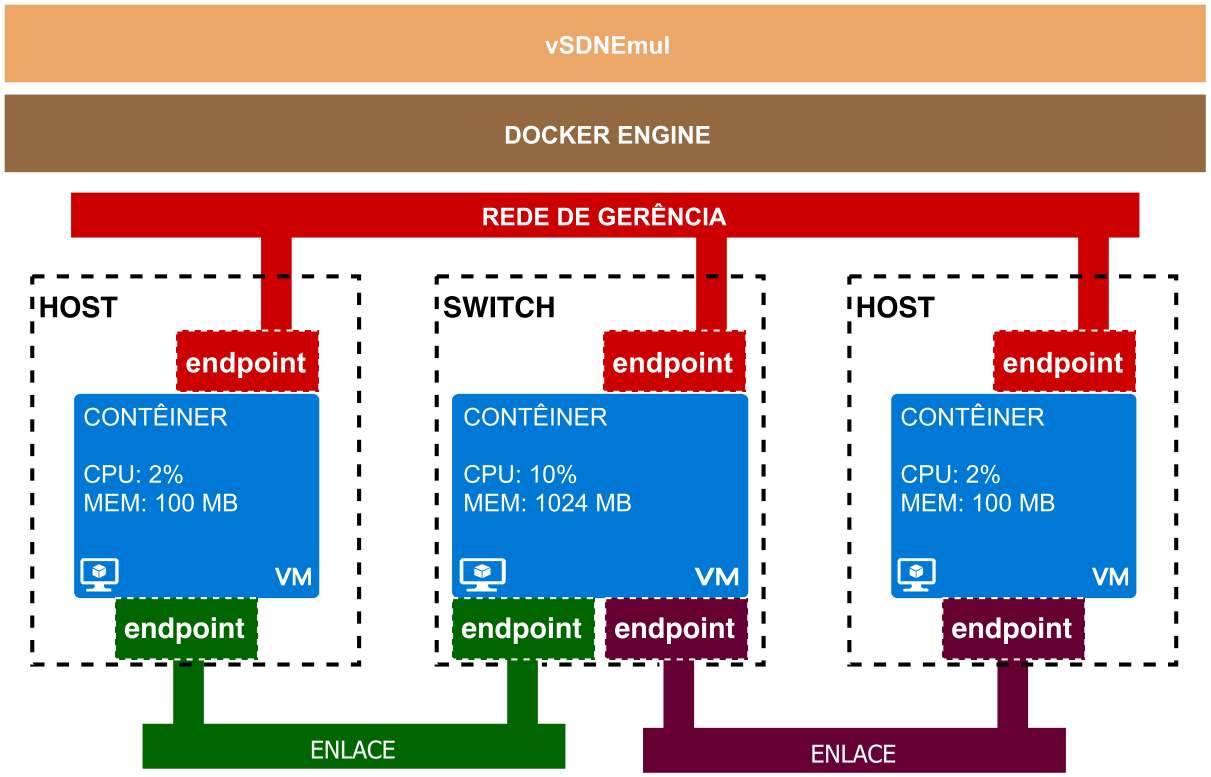
O *vSDNEmul* é um emulador de redes definidas por software que utiliza a conteinerização de sistema completos através do Docker[5], de forma, que ele isole os nós em contêiner e os interconecte através de enlace ou túneis virtuais. A adoção de contêiner oferece uma vantagem em relação a outros emuladores, que é o total isolamento de recursos de forma independente e flexível. Além disso, ele permite participar, do experimento emulado, uma diversidade de tipos de sistemas operacionais, aplicações de rede e sistemas distribuídos. Através de contêiner é possível emular uma infraestrutura completa de rede com hosts, servidores, switches ou roteadores, independentemente se são compatíveis à SDN ou outras arquiteturas.

A Figura 22 apresenta uma visão geral da proposta e ilustrando como os contêineres são executados para emular a topologia de rede. Basicamente a proposta pode ser independente da tecnologia de virtualização de contêiner, no entanto, como prova de conceito adotou-se o framework *Docker* pela possibilidade do aumento da escalabilidade do experimentos (usando *Swarm*[6] ou *Kubernetes*[7]) e a variedade de aplicações disponíveis em seu repositório. No entanto, a proposta não se limita apenas uma tecnologia de virtualização podendo se integrar outras soluções, por exemplo, o LXC[8].

Na Figura 22, cada contêiner é uma imagem baseada em um sistema operacional Linux contendo uma ou mais aplicações que são instanciadas durante o funcionamento do mesmo, por

[5] Projeto Docker: http://www.docker.com
[6] Projeto Docker Swarm: https://docs.docker.com/engine/swarm/
[7] Projeto Kubernetes: https://kubernetes.io/pt/
[8] Projeto LXC: http://www.linuxcontainers.org

Figura 22 – Visão geral da proposta vSDNEmul



Fonte: Autor

exemplo, o *Open vSwitch* para a criação de *switches* virtuais ou um servidor HTTP Apache. Além disso, esses contêineres podem possuir recursos isolados e configurados de maneira específica, como a necessidade de uma quantidade de memória e processamento.

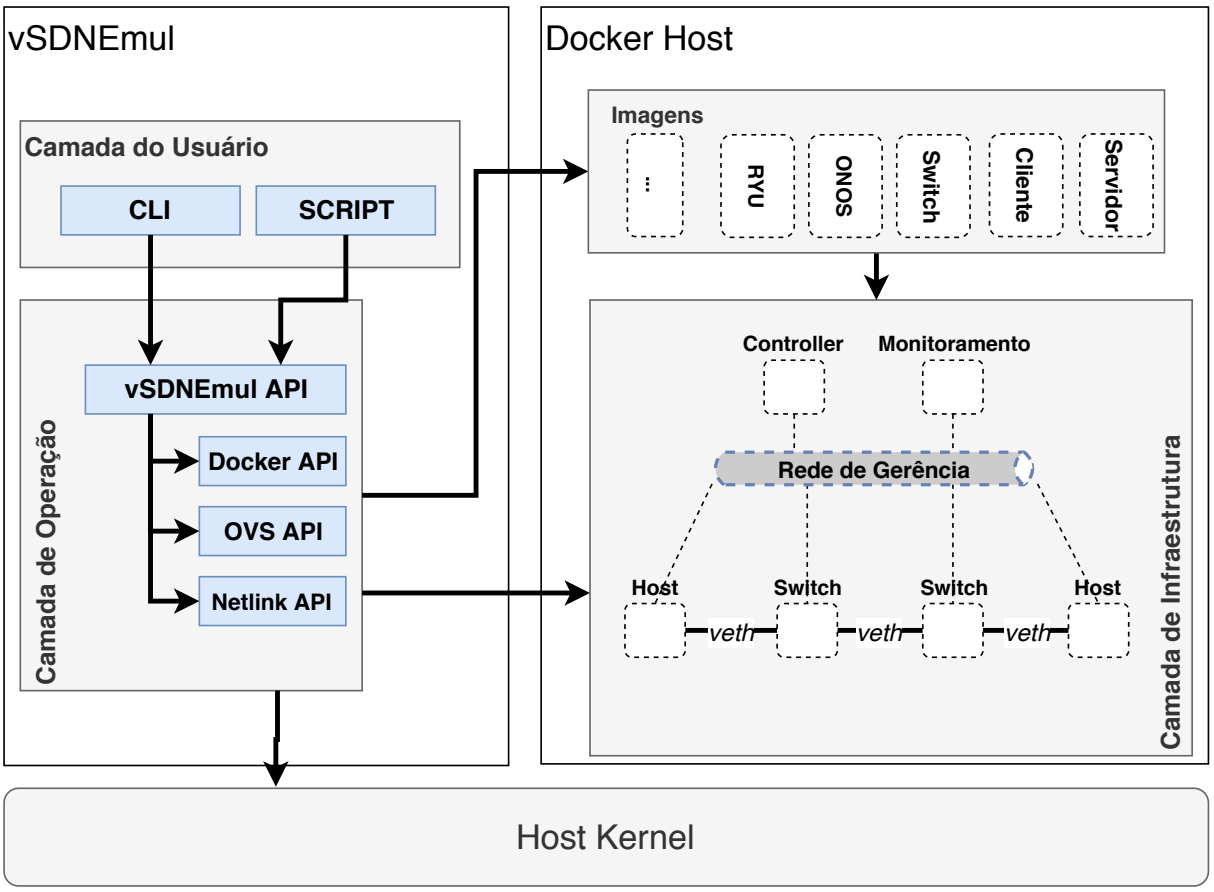
Ainda, na Figura 22, os enlaces que comunicam os nós entre si na topologia podem ser de dois tipos: pares de interfaces virtuais *ethernet* (*veth-pairs*) ou *bridges*. Sendo que a diferença entre eles é o desempenho na transmissão de dados ou vazão, simplesmente por estarem sendo executados na camada do usuário. Porém é possível melhorar esse desempenho através de uso de aceleradores de pacotes (ex., *DPDK*).

Por fim, tem-se a rede de gerência que é utilizada para enviar e receber os dados de controle, gerenciamento ou compartilhamento de internet. Nesta rede é integrada a comunicação de controle, tais como: nós de controladores e monitoramento. No entanto, esta rede já é provida pelo próprio *framework* do *Docker* para acesso a esses contêineres.

4.2.1 Arquitetura do *vSDNEmul*

O *vSDNEmul* é formado por uma arquitetura de três camadas que foi desenha para apoiar a emulação de experimentos através da instanciação de contêineres, enlace e portas virtuais. Conforme ilustrado na Figura 23 o *vSDNEmul* está dividido, respectivamente em: Camada de usuário, Camada de Operação e Camada de Infraestrutura.

Figura 23 – Arquitetura do vSDNEmul



Fonte: Autor

A Camada do usuário é responsável por interagir o usuário às funcionalidades do emulador. Atualmente, essa interação pode ser feita de duas maneiras: a primeira é através da sua CLI, que permite executar ações básica de controle e gerenciamento da topologia (ex., criar, listar e remover nós, enlace e controladores), essa opção e muito utiliza para simples experimentos; e na segunda opção, a construção do experimento é feito através de *scripts* utilizando API do *vSDNEmul*, neste caso, o emulador utiliza a linguagem *Python* para definir a construção e ações que ocorrerão durante a execução do mesmo, a vantagem desta opção é que o experimento fica mais rico de detalhes podendo acessar e configurar características mais complexas a respeito dos elementos utilizados.

Na Camada de Operação, encontra-se as bibliotecas necessárias para alocação de recurso no virtualizador, construção dos enlaces e acesso a outras funcionalidade do sistema operacional. Além disso, há a biblioteca do *vSDNEmul* que permite a manipulação de elementos do experimento, como também, a extensão de novas funcionalidades a modelos definido pelo emulador, por exemplo, a utilização de uma nova aplicação definida em uma imagem *Docker* ou um novo tipo de enlace disponível para topologia do experimento.

A Camada de Infraestrutura possui os recursos alocados pelo emulador durante a construção da topologia. Basicamente, ela é uma camada de software que contêm todos os elementos

lógicos de uma emulação, tais como: contêineres e enlaces construídos para representar o experimento. Além disso, por ser baseado em SDN, a camada de Infraestrutura possui os dois canais (controle e dados) separados para transmissão e recepção de dados. Seja para gerenciamento do nó ou envios de dados entre cliente e servidor.

4.2.2 Elementos de Redes

vSDNEmul usa contêineres para descrever nós em topologias emuladas. Ele podem incorporar elementos distintos durante o experimento, tal como: clientes, servidores ou dispositivos de redes, a partir da confecção de contêineres. Com *Docker* é possível personalizar a composição e execução de cada nó (ex., características do sistema operacional ou serviços disponíveis). O *vSDNEmul* implementa estas personalizações através de arquivo de descrição do *Docker*, chamado de *DockerFile*, que contém todas informações e configurações para construção da imagem que servirá de nó.

Esta estrutura permite ao *vSDNEmul* trabalhar com modelos de nós sempre atualizados ou customizados, pelo desenvolvedor. Isto permite ao emulador utilizar os mesmos softwares utilizados no ambiente de produção, tornando o cenários do emulador ainda mais realísticos que outra soluções. Além disso, o *Docker* tem uma extensa comunidade que compartilham imagens de aplicações, que podem facilitar a aplicação de novos modelos e a reprodução do experimento por outros usuários. Atualmente, o *vSDNEmul* possui um conjunto de modelos de elementos de rede disponíveis, dentre o quais destaca-se:

- **Whitebox**: o modelo *whitebox* é um contêiner que representa um switch SDN. Ele é baseado na aplicação *Open vSwitch* que habilita gerenciamento e controle através dos protocolos *OVSDB* e *OpenFlow*, respectivamente. Este nó permite a criação sob demanda de switches virtuais utilizando o *OVSDB*. Além disso, este modelo segue a proposta de *whitebox* descrito neste capítulo;

- ***Controller***: o modelo controller é um contêiner que representa um controlador SDN. Atualmente, há dois tipos de controladores disponíveis o Ryu[9] e ONOS[10]. O ONOS é um controlador desenvolvido em JAVA que dá suporte à diversos protocolos de *southbound* (ex., *OVSB*, *LISP*, *NETCONF* e *SNMP*). Além de várias versões do protocolo *OpenFlow*. O *Ryu* é um controlador menos robusto baseado em *Python* que dá suporte aos protocolos *OpenFlow* e *OVSDB*.

- ***Host***: o modelo host é um contêiner que pode representar aplicações tanto cliente quanto servidor. Este nó foi desenvolvido com várias ferramentas para análise de métricas de rede (ex., vazão, latência, atraso, largura de banda e variação do atraso). Como servidor, ele

[9] Projeto RYU: https://osrg.github.io/ryu/
[10] Projeto ONOS: https://onosproject.org/

pode se basear em qualquer aplicação real ou serviço de rede. O *Docker* oferece várias imagens de aplicações em seu repositório do *DockerHub*.

O *vSDNEmul* implementa também modelos de enlaces para emular conexões cabeadas entre as interfaces virtuais. Estas interfaces encaminham os pacotes de dados entre hosts e switches, e sua implementação pode variar de acordo com sistema operacional implantado. Atualmente, há dois tipos de enlaces disponíveis:

- ***Veth***: Esse modelo emula um enlace cabeado entre duas portas virtuais, permitindo a comunicação via protocolo Ethernet.
- ***Túnel***: O modelo túnel são enlaces virtuais para conectar dois contêiner locais ou remotos. Os tuneis podem ser implementados em *GRE* ou *VxLAN*.

A princípio, esta proposta aplica apenas enlaces cabeados. Entretanto, soluções de extensões podem ser desenvolvidas para o suporte a emulação de enlaces sem fio ou ópticos.

O *vSDNEmul* propõem uma maneira diferente de emular topologias de redes e seu elementos, permitindo à criação de um verdadeiros *testbed* portáteis, de sistemas compatíveis com SDN. Além disso, o objetivo central do emulador é criar experimentos em redes SDN através de virtualização baseado em contêineres, permitindo experimentos emulados ainda mais realistas. Adicionalmente, ao comparar as características do *Mininet* com o *vSDNEmul*, o *vSDNEmul* consegue ser mais flexível e realista. Além disso, sua API permite a rápida adição de novos elementos ao emulador.

4.3 Conclusão do Capítulo

Esta capítulo apresentou a proposta de arquitetura *vSDNLight* que responde a questão principal e fundamenta a hipótese desta tese. Para isso foram apresentados e descritos os principais alicerces da proposta de arquitetura: whitebox, modelo de slice via VSI e os componentes de orquestração (*vSDNOrches*) e provisionamento (*vSDNAgent*). Os whiteboxes permitiram a introdução de switches que provessem a instanciação de switches virtuais sob demanda. Já o modelo de slice via VSI capacitou à arquitetura a adoção de uma nova forma de slice formado por um conjunto de VSIs estabelecidos na infraestrutura permitindo, assim, remoção das traduções intermediária no plano de controle e distribuído esse processamento pelo plano de dados. Contudo, os componentes de orquestração e provisionamento habilitaram uma forma coordenada e leve de administrar os ciclo de vida tanto dos slices quanto dos switches virtuais.

Além disso, a resposta a questão suplementar foi dada pelo *vSDNEmul*, permitindo a utilização de contêineres para gerar emulações de experimentos de rede, mais realistas. No *vSDNEmul* cada nó executa de forma independente e isola umas das outras. Dando suporte a execução da proposta de arquitetura desta tese.

5 AVALIAÇÃO DA PROPOSTA

Neste capítulo é apresentada a análise do trabalho, que está dividido em duas seções. A primeira avalia o emulador proposto. Já a segunda, avalia a arquitetura proposta. Basicamente, cada seção possui uma descrição de implementação, a metodologia utilizada e a análise dos resultados obtidos.

5.1 *vSDNEmul*

O *vSDNEmul* foi desenvolvido para habilitar um ambiente que suportasse a implementação da arquitetura *vSDNLight*. Infelizmente, a principal ferramenta de emulação de redes SDN (i.e., *Mininet*) não suporta a utilização de VSI por switch emulado. Além disso, a questão da fidelidade de sua emulação é questionável (HANDIGOL et al., 2012; ISAIA; GUAN, 2016). Portanto, pretende-se realizar uma comparativo entre o *Mininet* e o *vSDNEmul*. Essa análise de desempenho visa observar dois aspectos: escalabilidade e fidelidade de vazão com ou sem tráfego concorrente.

5.1.1 Implementação do *vSDNEmul*

O *vSDNEmul* foi desenvolvido em *Python* utilizando à versão *3.7*, sendo que, para à implementação dos contêineres foi utilizada a solução *Docker* na versão *17.03*. Todavia, A versão empregada do emulador foi a *0.2* que está disponível em repositório no *Github*[1] do projeto.

A API do emulador disponibiliza alguns modelos de enlace e nós, que foram descritos no Capítulo 4. Além disso, são disponibilizados simples exemplos de topologias para facilitar o entendimento da API. Todas as bibliotecas externas e as suas respectivas versões estão disponíveis no repositório do gerenciador de bibliotecas do *Python* o *PyPi*[2], o objetivo foi diminuir ao máximo à incompatibilidade da API, quando instalado em outros sistemas operacionais Linux. A atual versão do emulador foi desenvolvida e testada utilizando o sistema operacional Fedora[3] versão 29.

As imagens dos nós utilizados pelos modelos, já estão pré-construídas e disponíveis no repositório de imagens do *Docker*, o *DockerHub*[4] . Isso diminui o tempo de instalação da emulador. No entanto, para o desenvolvimento de um novo modelo é necessário construir uma imagem com um sistema operacional *Linux*. Depois, é necessário a construção de uma classe que represente este novo modelo na API e assim inclui-la no emulador.

[1] Projeto vSDNEmul: http://github.com/fernnf/vsdnemul
[2] Projeto PyPi: http://pypi.org
[3] Projeto Fedora: http://getfedora.org
[4] Projeto DockerHub: http://hub.docker.com

5.1.2 Metodologia

Para a realização do estudo, foi estabelecido dois casos, onde o primeiro avalia à escalabilidade dos emuladores a partir de diferentes métricas relacionadas, principalmente, a utilização de recursos computacionais da máquina que hospeda os emuladores; e o segundo analisa da fidelidade do comportamento na reprodução de uma ambiente de rede criado pelos emuladores, a partir da aplicação de testes de vazão. O objetivo dos testes é avaliar qual dos emuladores é o mais escalável, que consequentemente poderá emular ambientes de rede mais robustos e qual deles apresenta maior fidelidade.

Ambos os casos foram reproduzidos em uma máquina virtual do *KVM* com 14*GB* de *RAM*, 80*GB* de disco rígido, 2 núcleos de processamento e sistema operacional *Ubuntu* 18.04, hospedada por um computador *Desktop Dell* com processador *intel core i7-4790* 3.6 *Ghz* de 4 núcleos e 16*GB* de memória *ram ddr3*. Entretanto, caso a reprodução dos testes sejam verificadas em ambientes computacionais com configurações muito distintas das que foram descritas, os resultados obtidos serão diferentes dos encontrados nesta tese, isto ocorrerá devido a disponibilidade de recursos computacionais da máquina de emulação afetar os resultados dos experimentos.

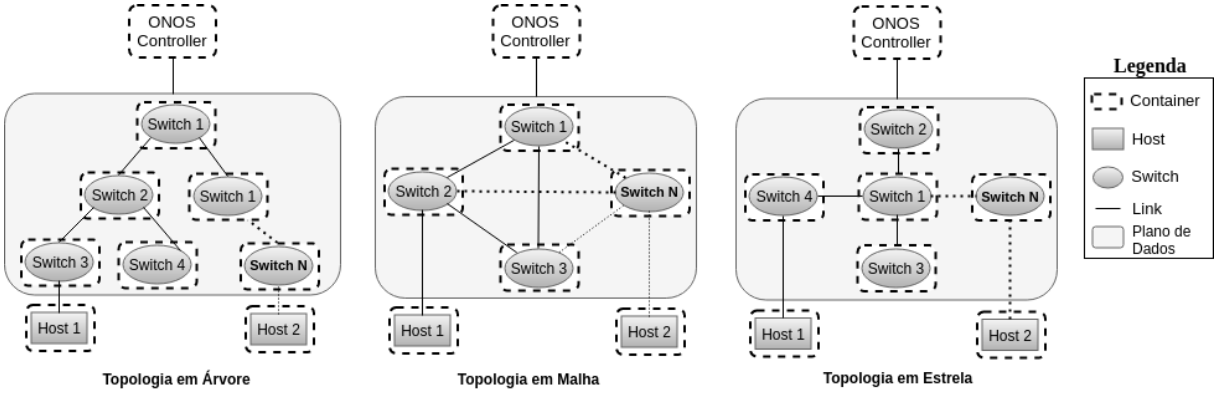
Em cada caso, os testes foram repetidos 15 vezes e coletados durante 50 segundos de execução dos experimento, técnica semelhante a adotada por Isaia e Guan (2016). Para cada repetição, foi calculado à média dos resultados de acordo com a variação do número de nós aplicados as topologias.

5.1.2.1 Análise de Escalabilidade

Para analisar a escalabilidade de cada emulador, foram definidos alguns ambientes virtuais de redes SDN, estes ambientes foram baseados em 3 diferentes tipos de configuração de topologias, tais como: árvore, estrela e malha. A Figura 24 representa as topologias criadas tanto para o *vSDNEmul* quanto para o *Mininet*. Em todas as topologias, foram analisados os desempenhos variando uma quantidade N de switches, onde N assume os valores iguais à 7, 15, 31, 63, 127, 255 e 511, os valores de N foram definidos de modo empírico, descartando qualquer relação de causa ou efeito na execução dos testes.

Em cada topologia, variou-se o número de switches na rede com o intuito de verificar a escalabilidade e o desempenho alcançado por cada emulador avaliado. Este modelo de análise foi baseado no trabalho de Isaia e Guan (2016), onde os autores avaliaram o desempenho do emulador *Mininet* utilizando topologias com características distintas, tais como o número de enlaces e tamanho da rota que o pacote enviado irá percorre na rede, de modo a verificar se estas características poderiam afetar o resultado da atuação da rede emulada.

Figura 24 – Topologias utilizadas para a análise da escalabilidade



Fonte: Autor

Para o controle das redes de teste foi utilizado o controlador ONOS em um contêiner do *Docker* para ambos os emuladores, porém, não incluído nos scripts dos experimentos. Pois, como *Mininet* não fornece controladores em sua emulação, diferente do *vSDNEmul,* fez-se isso para que a rede se torna-se funcional para o *Mininet* e que não houvessem vantagens por parte de um dos emuladores. Portanto, optou-se por utilizar o controlador desta forma devido ele ser o mesmo utilizado pelo *vSDNEmul* e ser facilmente configurado.

Para este primeiro caso, foram tomadas as seguintes métricas de desempenho aplicadas em cada topologia: i) tempo de inicialização e finalização; ii) média de uso de CPU e memória RAM; e iii) fidelidade de vazão (sem tráfego concorrente).

No tempo de inicialização e finalização, a verificação desta métrica buscou analisar o tempo que cada emulador levou para iniciar e finalizar cada topologia. Através dela foi possível verificar, quais dos emuladores tiveram melhor eficiência no uso dos recursos computacionais disponíveis, já que quanto maior o tempo gasto, maior o tempo de CPU utilizado. Para calcular estes intervalos de tempos foi utilizado a função *Time*[5] da linguagem *Python*, que nos permite calcular o tempo de execução de códigos ou funções desta linguagem.

A média de uso de CPU e memória RAM é importante para qualquer sistema computacional, seja para verificar o adequado funcionamento da aplicação ou a escalabilidade do sistema. Para esta análise, buscou-se verificar a quantidade de recursos computacionais necessários para habilitar cada topologia de acordo com número de nós. A verificação destas métricas se deu a partir da API do *Linux* (*Mininet*) e do *Docker* (*vSDNEmul*), o que permitiu realizar o monitoramento em tempo real da utilização de CPU, RAM, I/O de rede e I/O de blocos de cada contêiner ou *namespace* em execução na máquina virtual.

Neste contexto, o emulador *vSDNEmul*, devido em sua arquitetura, em que cada nó da rede é um contêiner, a coleta das amostras se deu a partir da soma do uso de CPU e Memória de todos os nós da rede em cada segundo de tempo de execução da emulação, por exemplo, o uso

[5] Função Time() Python: https://docs.python.org/3/library/time.html

de CPU da topologia em estrela no instante t=1seg, equivale a soma de uso de CPU de todos os nós da topologia neste mesmo instante de tempo. Por outro lado, no *Mininet*, devido todos os nós serem executados em um único *namespace*, o uso de CPU e memória foram coletados a partir de um único contêiner.

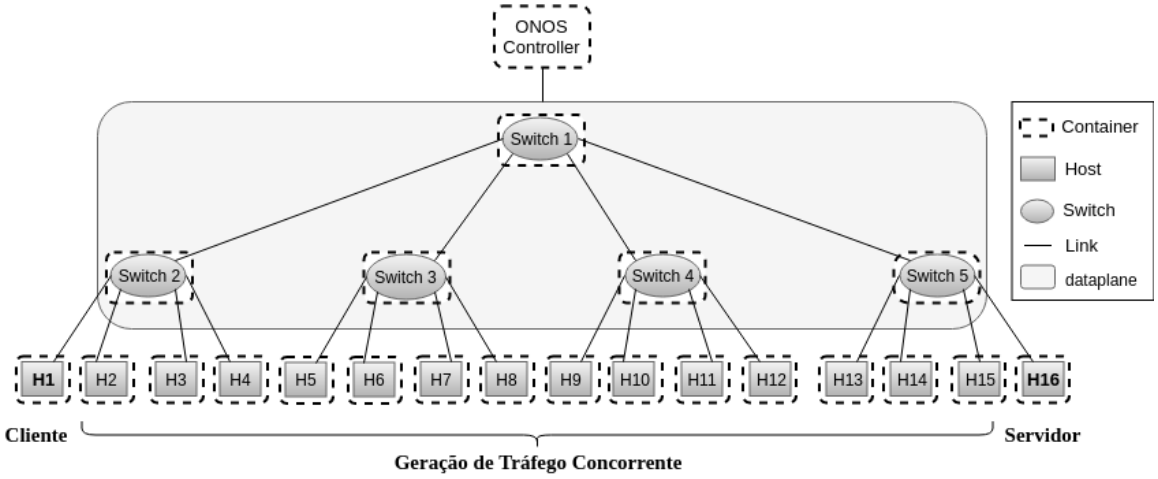
Na fidelidade de vazão, a utilização desta métrica foi baseada no trabalho de Yan e Jin (2017) que utilizou esta abordagem como avaliação de sua proposta fazendo uso do emulador *Mininet*. Para este caso, buscou-se avaliar o comportamento da vazão de cada emulador ao enviar pacotes a uma taxa constante de 1.5 *Gbps* a medida em que se aumenta o número de switches na rede e não há presença de tráfego em segundo plano, o que caracteriza um cenário de rede ideal.

Assim, de acordo com Yan e Jin (2017) a taxa de transferência dos dados na rede deve-se manter inalterada ou apresentar valores próximos do desejado devido não existir concorrência no enlace. Para a investigação desta métrica, uma conexão cliente-servidor do tipo UDP entre o *host1* e o *host2* foi criada utilizando o *Iperf3*[6] , onde o *host1* (cliente) enviou pacotes ao h*ost2* (servidor) por cerca de 50 segundos, e durante esta transmissão se mediu a taxa de transferência para os diferentes tamanhos das topologias da Figura 24.

5.1.2.2 Análise de Vazão Utilizando Tráfego Concorrente

A aplicação desta métrica foi baseada no trabalho de Roy et al. (2014) que comparou a fidelidade de vazão do *Mininet* utilizando uma topologia em árvore com 5 *switches* e 16 *hosts*, conforme Figura 5x3 foi estabelecida uma conexão em primeiro plano do tipo cliente-servidor entre o host1 e o host16 usando o *Iperf3* com o protocolo UDP, disparando pacotes entre eles a uma taxa de transferência constante de 1000 Mbps. Ao mesmo tempo em que é estabelecida esta conexão, foi criado 7 pares de conexão cliente/servidor para os 14 *hosts* restantes da topologia, utilizando o *iperf3* com protocolo UDP.

Figura 25 – Topologia em árvore utilizada na avaliação da vazão com tráfego concorrente.



Fonte: Autor

[6] Projeto Iperf: http://iperf.fr

Isto foi feito em conjunto da geração do tráfego em segundo plano com taxas de transmissão iguais a 200, 400, 600, 800 e 1000 Mbps com o intuito de verificar o comportamento da rede emulada pelo *Mininet*.

Diante disto, o mesmo procedimento adotado pelos autores foi aplicado utilizando os emuladores *Mininet* e *vSDNEmul*, com a diferença de que utilizou-se, diferentes valores no tráfego principal, mais especificamente, valores iguais a 1000, 1500, 2000, 2500 e 3000 Mbps e no tráfego em segundo plano, valores iguais a 400, 600, 800, 1000 e 1200Mbps.

5.1.3 Resultados

5.1.3.1 Análise da Escalabilidade

O tempo de inicialização e finalização por topologia foi uma característica importante para testar a escalabilidade dos emuladores, pois, conforme o tempo de inicialização aumenta, maior será o tempo de uso de CPU, podendo gerar instabilidade na rede emulada, devido a sobrecarga dos recursos computacionais.

As tabelas 3 e 4 demonstram a comparação entre o *Mininet* e o *vSDNEmul* referente ao tempo de inicialização e finalização, em minutos, de cada uma das topologias descritas na Figura 24. Através das tabelas é possível verificar que o *vSDNEmul* apresenta resultados superiores ao *Mininet*, isto ocorre devido a sua diferença de arquitetura.

Tabela 3 – Tempo de inicialização (em minutos) por topologia

Nº Switches	**Topologia em Árvore**		**Topologia em Estrela**		**Topologia em Malha**	
	Mininet	vSDNEmul	Mininet	vSDNEmul	Mininet	vSDNEmul
7	0.32	0.55	0.35	0.51	0.67	1.10
15	0.57	0.82	0.55	0.80	1.12	1.53
31	1.12	2.11	1.10	2.13	1.89	4.05
63	1.89	4.18	1.78	4.10	2.33	7.18
127	2.88	7.42	2.97	7.31	3.80	13.23
255	4.34	15.57	6.68	15.1	7.77	19.55
511	7.98	30.4	7.16	30.25	9.78	38.27

Ao inicializar uma topologia, o *vSDNEmul* aloca as imagens (contêiner) correspondentes a cada modelo de nó, essas imagens irão inicializar seu sistema operacional e posteriormente executar um processo contendo a aplicação do modelo, por exemplo, no caso dos switches, o *Open vSwitch* (OVS) e, em seguida, criará todos os enlaces da topologia. Diferente do *Mininet*, que aloca um contêiner contento o processo do OVS já inicializado no sistema operacional hospedeiro e todos os demais nós criados, são incorporados a este processo e incluindo as portas e enlaces, o que acaba por diminuir consideravelmente o tempo de inicialização das topologias. Por outro lado, o fato do *vSDNEmul* inicializar vários contêineres, agrega um tempo maior tanto para criar como para finalizar uma topologia. Principalmente, acima dos 31 nós onde

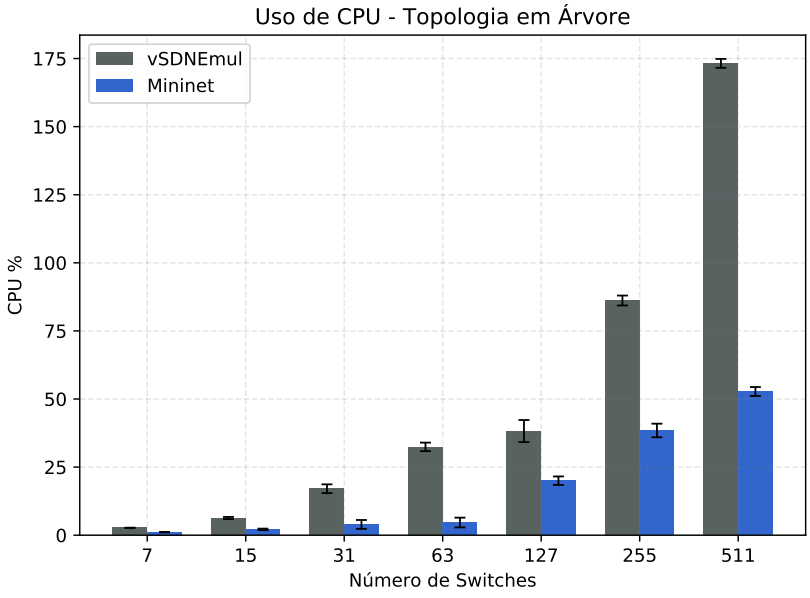
esses tempos começam a se distanciar entre Mininet e vSDNEmul. Por exemplo, se pegarmos a topologia em malha com 511 switches, que é a topologia que mais leva tempo para inicializar e finalizar, a diferença entre os emuladores *Mininet* e *vSDNEmul*, respectivamente, é 74,45% na inicialização (Tabela 3) e 74.67% na finalização (Tabela 4).

Tabela 4 – Tempo de finalização (em minutos) por topologia

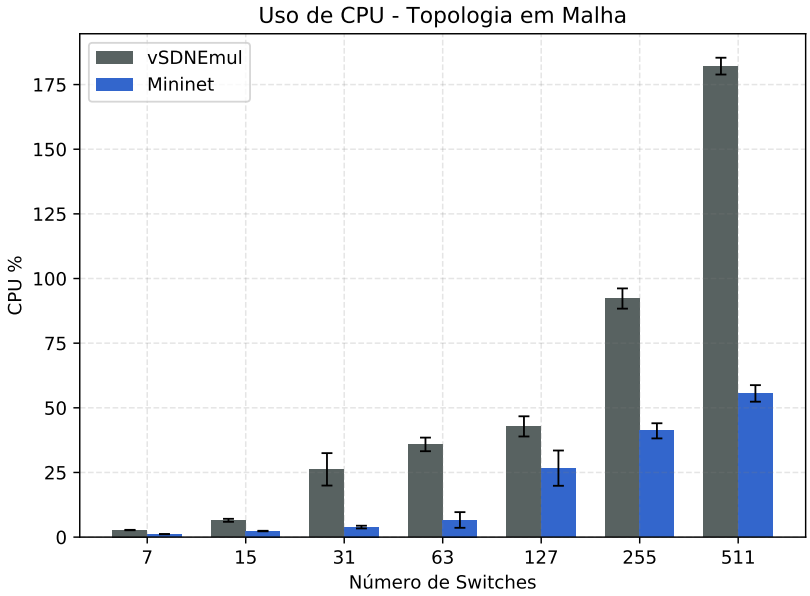
Nº Switches	Topologia em Árvore		Topologia em Estrela		Topologia em Malha	
	Mininet	vSDNEmul	Mininet	vSDNEmul	Mininet	vSDNEmul
7	0.34	0.83	0.31	0.80	0.85	1.18
15	0.89	1.13	0.78	1.16	1.45	1.88
31	1.39	2.23	1.42	2.02	2.02	4.45
63	2.11	5.10	2.14	5.15	2.64	7.79
127	3.18	9.18	3.08	9.19	4.07	14.78
255	4.93	19.18	4.78	19.11	8.12	23.40
511	8.27	42.5	8.11	41.27	11.15	44.03

O uso de CPU é um quesito importante para a manutenção da confiabilidade dos resultados obtidos pelo emulador, pois este influenciará diretamente no seu desempenho à medida que o recurso de processamento vai ficando escasso. Os resultados descritos na Figura 26, demonstram o percentual médio de uso de CPU para cada topologia mediante o aumento do número de switches. Os valores de CPU foram obtidos a partir da verificação da utilização do processador. Assim, por estar se utilizado 2 núcleos o valor máximo de processamento a ser obtido é 200%, o que significa, que se este valor chegar a 200% todos os 2 núcleos do processador da máquina foram utilizados.

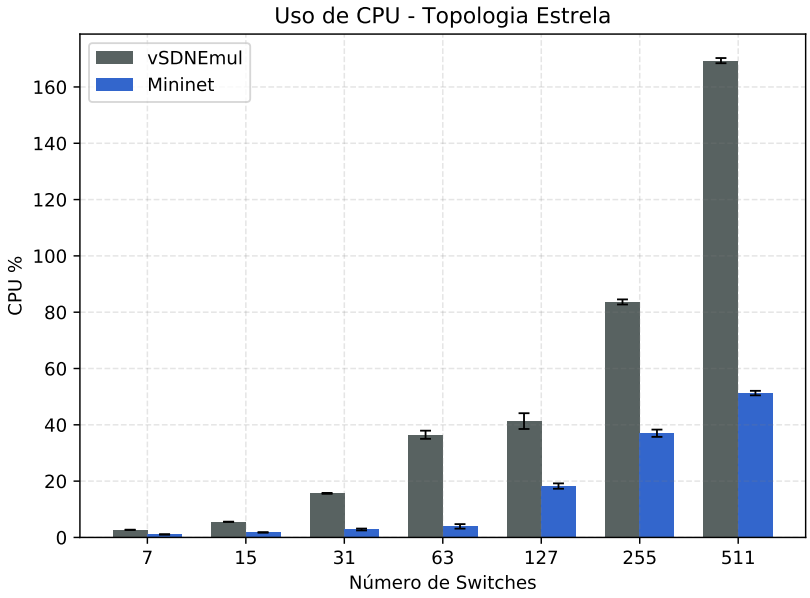
Figura 26 – Uso de CPU (em %) por topologia



(a)



(b)

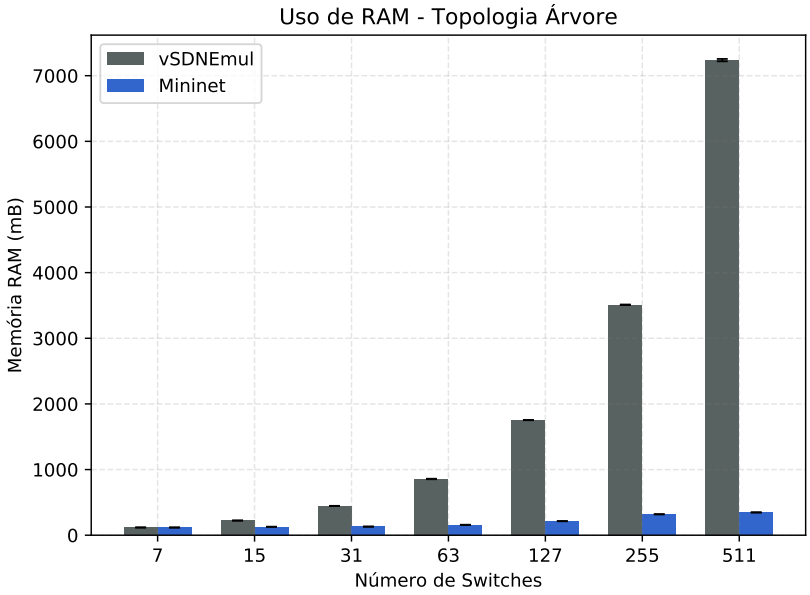


(c)
Fonte: Autor

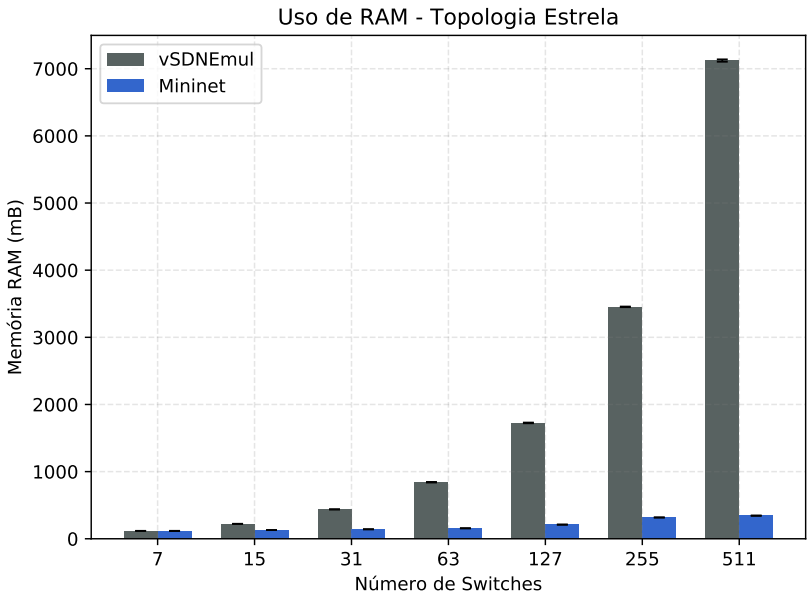
A partir dos gráficos da Figura 26, percebe-se que o *vSDNEmul* apresenta maior utilização de CPU em relação ao *Mininet* nas três topologias, sendo à topologia em malha (Figura 26c) a que obteve maior média de utilização da CPU, em torno de 175% para o *vSDNEmul* e 55% para o *Mininet*, uma diferença de 120%. Isto ocorreu por que na arquitetura do *Docker* há uma pré-alocação de recursos tanto de processamento quanto de memória para o funcionamento mínimo do isolamento do contêiner, tanto que o valor máximo de *switches* possível para alocação no *Docker* foi de 511 nós. Além disso, quanto mais enlaces, há mais interface e, consequentemente, mais recursos a serem alocar. Por isso, há a diferença de CPU entre as topologias, tanto para *vSDNEmul* e *Mininet*.

Os gráficos da Figura 27 mostram os resultados obtidos pela utilização média que cada emulador consumiu de memória durante a execução das diferentes topologias. Nas três situações, o *vSDNEmul* apresentou uso bem superior ao *Mininet*, que utilizou em torno de 340 MB, no pior caso, contra 7 GB do *vSDNEmul*, uma diferença de 66,6%. Este fato pode ser explicado devido o *vSDNEmul* utilizar em sua arquitetura, a virtualização de contêineres, que embora seja considerada uma virtualização com baixa utilização de recursos computacionais (CPU, RAM), quando utilizada em larga escala, agrega maior utilização destes recursos do que um processo do sistema Linux, já que o todos os comutadores criados pelo *Mininet* rodam em um mesmo processo do Linux.

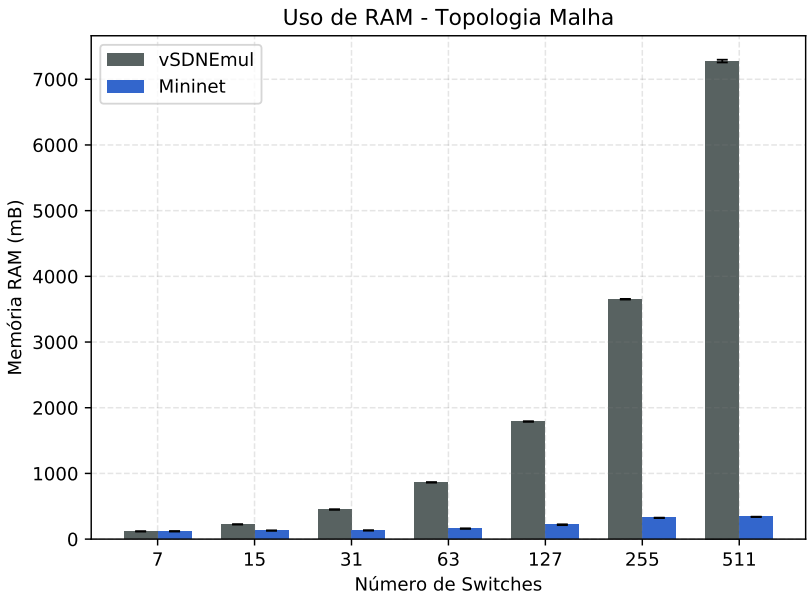
Figura 27 – Uso de memória (em MB) por topologia



(a)



(b)



(c)
Fonte: Autor

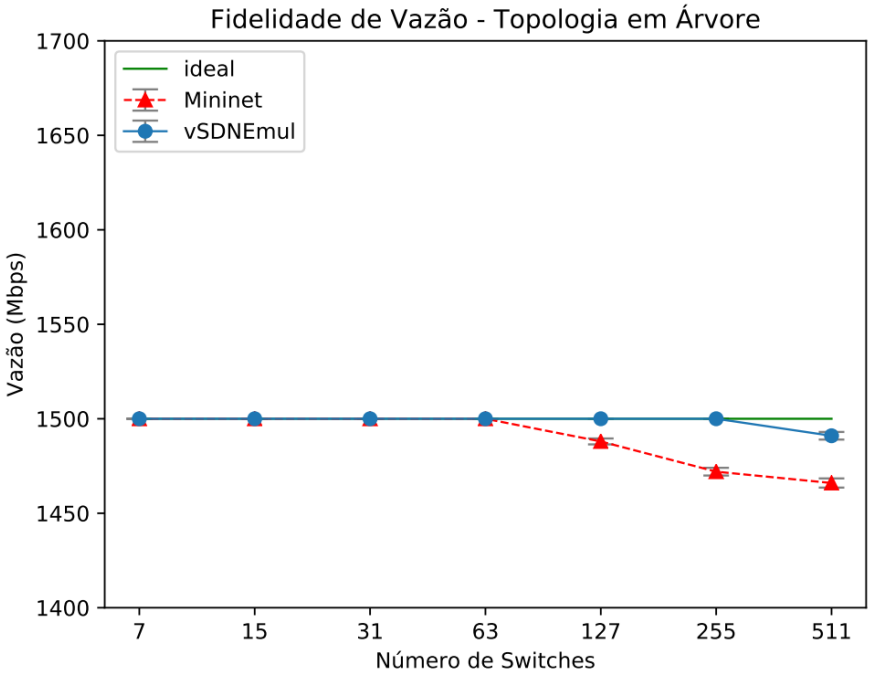
Apesar destas métricas (CPU, memória) serem muito mais elevadas no *vSDNEmul*, há um compensação em relação a isso, pois no caso do *Docker* esses recursos ficam pré-alocados para uso do contêiner, diferentemente do *namespace* que irá atrás do recurso quando o contêiner estiver em uso, sendo assim, dependendo da demanda o recurso pode não está disponível ou demorar a ser disponibilizado ocasionando atraso no processamento. Logo o que poderia se torna uma limitação, no caso do *vSDNEmul*, acaba sendo um vantagem, pois cada contêiner possuirá recursos mínimos de CPU e memória para sua execução. Além disso, diferente do *Mininet*, com *vSDNEmul* é possível ampliar os recursos através da utilização de *clusters* de repositórios de contêineres com soluções como *Swarm* ou *Kubernetes*, permitindo a execução de topologias em diferentes infraestruturas computacionais ou de nuvem computacionais. Assim, ampliando os ganhos do emulador quanto a escalabilidade.

Os gráficos da Figura 28 mostram os resultados da fidelidade da vazão, observou-se que para *vSDNEmul* a vazão de envio de pacotes se manteve quase constante, o que não ocorreu com *Mininet* devido ao fato comentado no parágrafo anterior, que apesar do emulador usar menos CPU e memória quando há necessidade da demanda ela pode não esta disponível, isso que fez com que fidelidade do *Mininet* começasse a decair, a partir da utilização de 31 nós (Figura 29c na topologia em malha). Entretanto, observa-se perda de fidelidade no *vSDNEmul*, quando a emulação está próximo dos limites disponíveis no computador hospedeiro, ou seja, quando há um alto consumo de CPU para topologias com grande número de switches. Isso, fez com que a capacidade de comutação das interfaces diminuíssem devido a utilização de CPU estar próxima do seu limite e não por limitações do emulador.

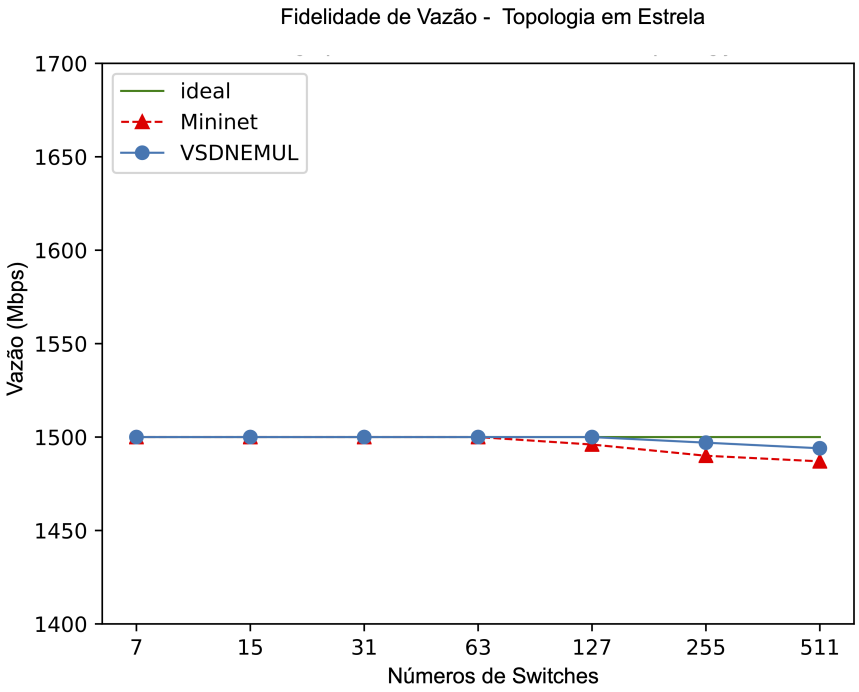
Desse modo, a topologia em malha ilustrado na Figura 29c, por possuir maior utilização de *CPU* e maior número de enlace, obteve os piores resultados em relação a fidelidade da vazão

em um tráfego de 1500Mbps, principalmente, em topologias com 127 *switches* para a malha e estrela e 255 *switches* para árvore. Apesar do Mininet apresentar uso de CPU bem inferior ao *vSDNEmul*, o mesmo apresentou resultados bem abaixo do esperado para esta métrica, o que pode ser considerado um problema característico da arquitetura do emulador Mininet.

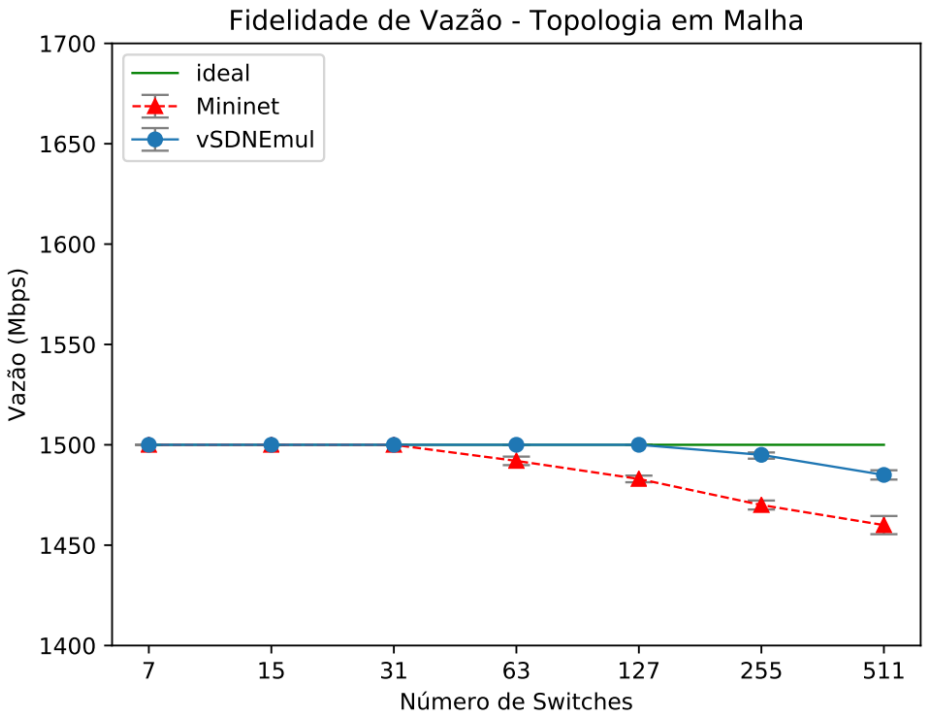
Figura 28 – Fidelidade de Vazão (Mbps) por Topologia



(a)



(b)



(c)
Fonte: Autor

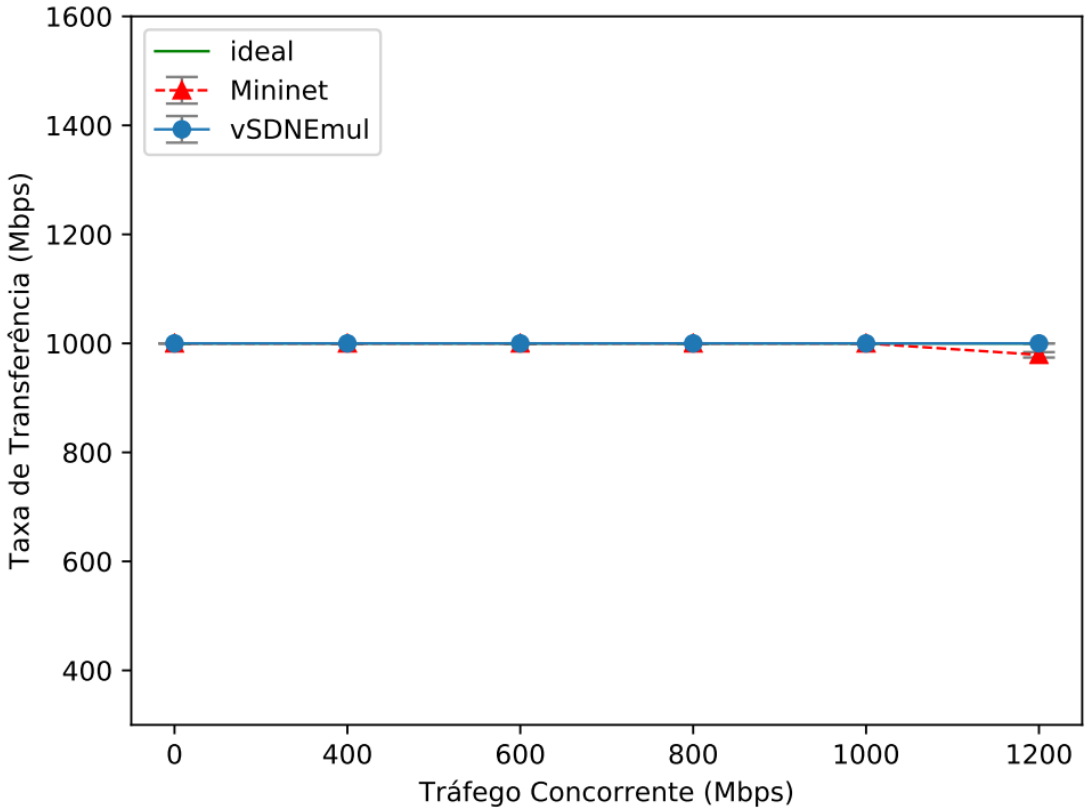
Portanto, embora o *Mininet* suporte a emulação de um grande número dispositivos devido possuir menor utilização de recurso computacionais (CPU e RAM), um grande número de enlaces prejudica a fidelidade da taxa de transferência na topologia emulada e compromete a precisão dos resultados, isso demonstra que a ineficiência de sua arquitetura prejudica em até 4% na coleta de resultados, enquanto o *vSDNEmul* é de 0,6%, para topologia em malha e árvore, porém por falta de recursos na máquina hospedeira.

Em contraste, o *vSDNEmul* pode suportar o mesmo número de nós, mantendo o rendimento mais próximo do ideal, mesmo diante de sobrecarga dos recursos computacionais. Ao usar às topologias de malha, estrela e árvore, o *vSDNEmul* chega a suportar 96 (Figura 28b e Figura 29c) e 192 (Figura 28a) nós a mais que o *Mininet*, mantendo a taxa de transferência inalterada.

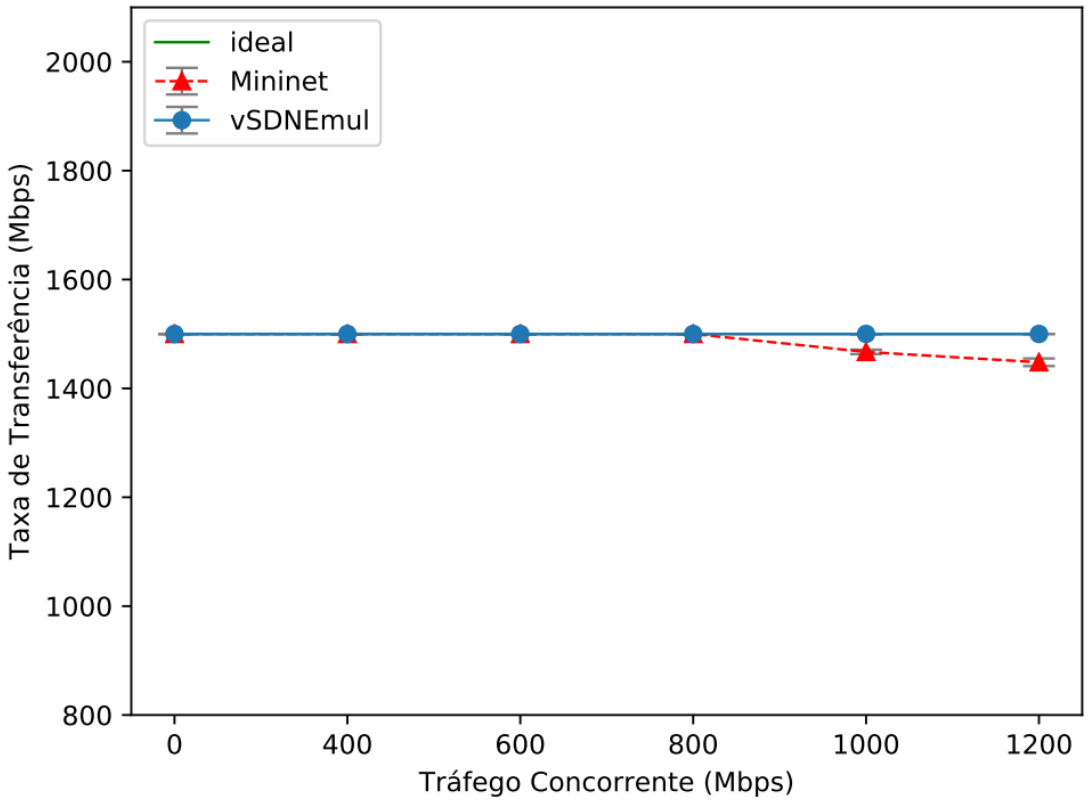
5.1.3.2 Análise da Vazão Utilizando Tráfego Concorrente

Através dos resultados descritos na Figura 29 é possível verificar que o tráfego em primeiro plano inicialmente permanece no valor desejado mesmo com o aumento do tráfego em segundo plano. No entanto, percebe-se que mesmo diante destas taxas de transmissão inferiores, o *Mininet* ainda obtém falhas que se distanciam do cenário ideal e que pioram ainda mais conforme a taxa de envio de pacotes no tráfego principal aumenta.

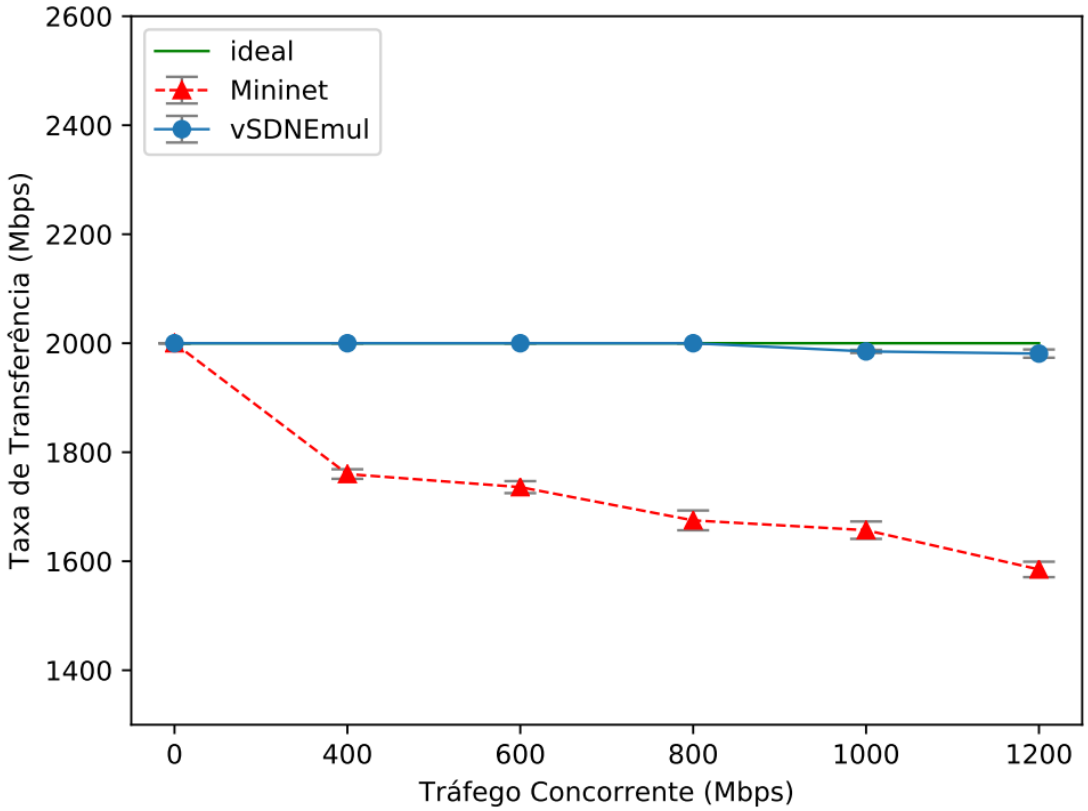
Figura 29 – Fidelidade da Vazão com Diferentes Taxa de Transferências



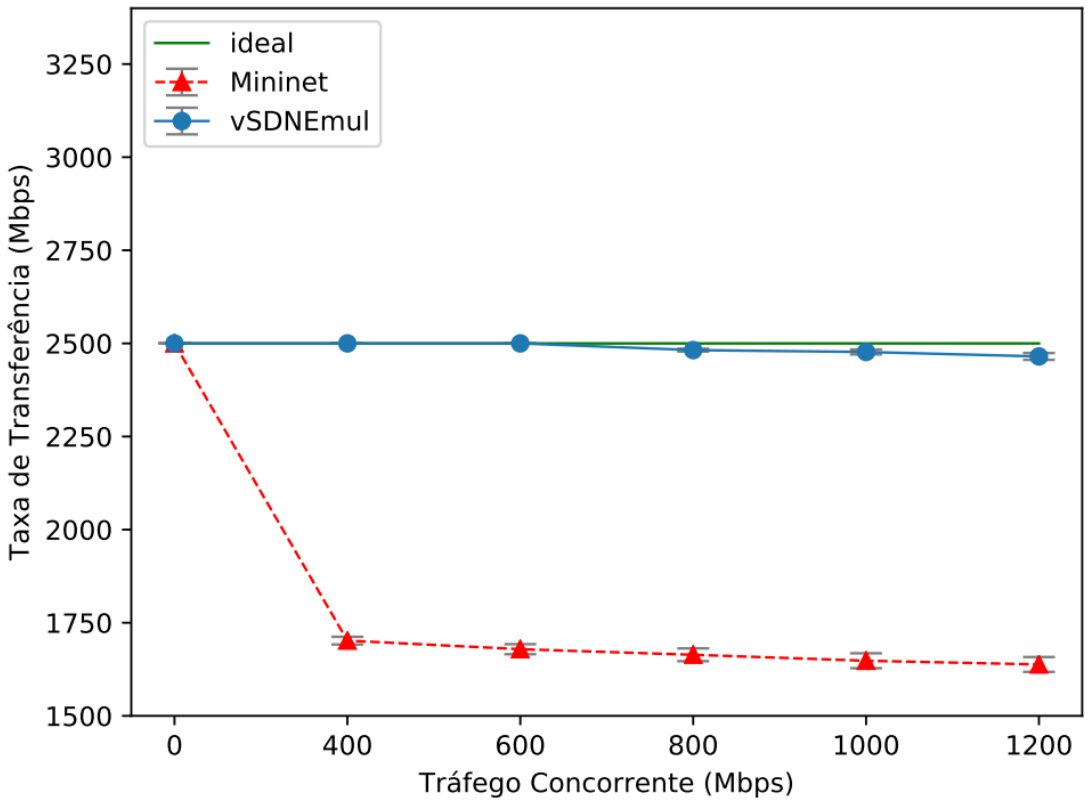
(a)



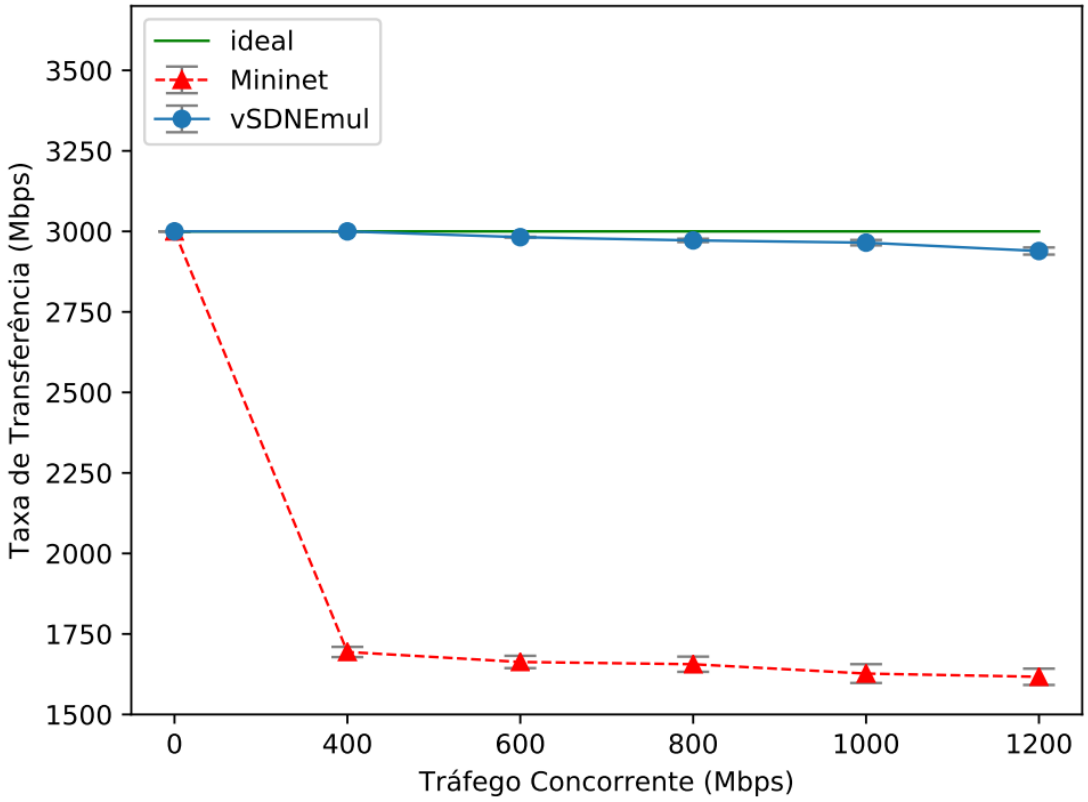
(b)


ideal
Mininet
vSDNEmul
Taxa de Transferência (Mbps)
2600
2400
2200
2000
1800
1600
1400
0
400
600
800
1000
1200
Tráfego Concorrente (Mbps)

(c)


ideal
Mininet
vSDNEmul
Taxa de Transferência (Mbps)
3250
3000
2750
2500
2250
2000
1750
1500
0
400
600
800
1000
1200
Tráfego Concorrente (Mbps)

(d)



(e)
Fonte: Autor

Segundo Roy et al. (2014), o tráfego pode ser reduzido para superar a limitação mencionada. No entanto, o encolhimento arbitrário de uma rede e seu tráfego para que ela possa caber em recursos disponíveis são problemas próprios da arquitetura do emulador *Mininet*. Mais especificamente, essa abordagem pode mostrar um comportamento de rede insatisfatório que é manifestado apenas durante a emulação.

A diferença obtida nos resultados pode ser compreendida, também, pela diferença de arquitetura entre os emuladores. Como os nós da rede emulada pelo *Mininet* estão compreendidas em um único contêiner *namespace* e não há isolamento das interfaces de rede, assim, todas elas estão associadas a um único processo integrado a este contêiner. Desse modo, ao se utilizar cenários com altas taxas de tráfego, a concorrência no processamento das filas de pacotes nas interfaces aumenta, e consequentemente, a taxa de transferência diminui. Pois, há apenas um processo para da vazão no processamento das filas de pacotes contidas no contêiner, além disso, por ser uma ambiente compartilhado este processo ainda tem que compartilhar o tempo de processamento com outros processos do sistema operacional. Portanto, a vazão diminui por este processo se sobrecarregar de interrupções de processamento de pacotes.

Apesar do *vSDNEmul* não conseguir manter a taxa de envio constante para valores acima de 2000 Mbps, os resultados obtidos apresentam perdas na fidelidade bem inferiores, entorno de 0.9%, já o *Mininet* tem perdas próximas de 42% da fidelidade em seu pior caso, Figura 29e. Com isso, conclui-se que o *vSDNEmul* consegue ter uma emulação mais fiel e

mantém uma taxa de transmissão bem próxima da taxa desejada. Isto demonstra que o isolamento criado pelos contêineres *Docker* a cada um dos nós da topologia de rede, minimiza o gargalo no processamento de filas das interfaces, produzindo resultados mais consistentes durante a emulação, e consequentemente, criando experimento ainda mais fieis a redes reais quando comparado ao emulador *Mininet*.

Esta seção apresentou os resultados obtidos durante a aplicação dos dois caso de avaliação do comportamento dos emuladores testados, computando os limiares de recursos computacionais necessários para cada emulador criar topologias com diferentes números de nós na rede, o tempo que cada um levou para iniciar ou finalizar as respectivas topologias, o consumo de CPU e Memória e a fidelidade de vazão com e sem tráfego concorrente com o intuito de verificar o comportamento da rede emulada por ambos as ferramentas.

A partir da análise destas métricas, foi possível concluir que o Mininet consegue suportar a construção de mais nós e utilizar menos recursos de CPU e memória, no entanto, os testes de vazão mostraram a fragilidade da sua arquitetura que limita o emulador criar emulações fieis. Além disso, essa limitação ainda prejudica a fidelidade de seus resultados.

Em contraponto a este resultado, mesmo diante de sobrecarga da CPU, o *vSDNEmul* mantém um padrão de consistência superior para esta métrica se comparado ao *Mininet*, tanto com tráfego concorrente, quanto sem ele. Um dos motivos desta diferença é devido todas as interfaces de rede criadas nos nós pelo *vSDNEmul* estarem isoladas e com recursos computacionais próprios a cada uma delas, ao contrário do *Mininet* que estão dispostas dentro de um único namespace na máquina host e compartilham os mesmos recursos computacionais entre elas.

Já, em relação a escalabilidade o *vSDNEmul* vai depender dos recursos disponíveis no sistema hospedeiro, No entanto, percebeu-se que no *Mininet* quanto mais nós participam da geração de tráfego a fidelidade diminui drasticamente durante o experimento, diferente do *vSDNEmul* que mantém sua fidelidade mais próximo possível do esperado.

5.2 vSDNLight

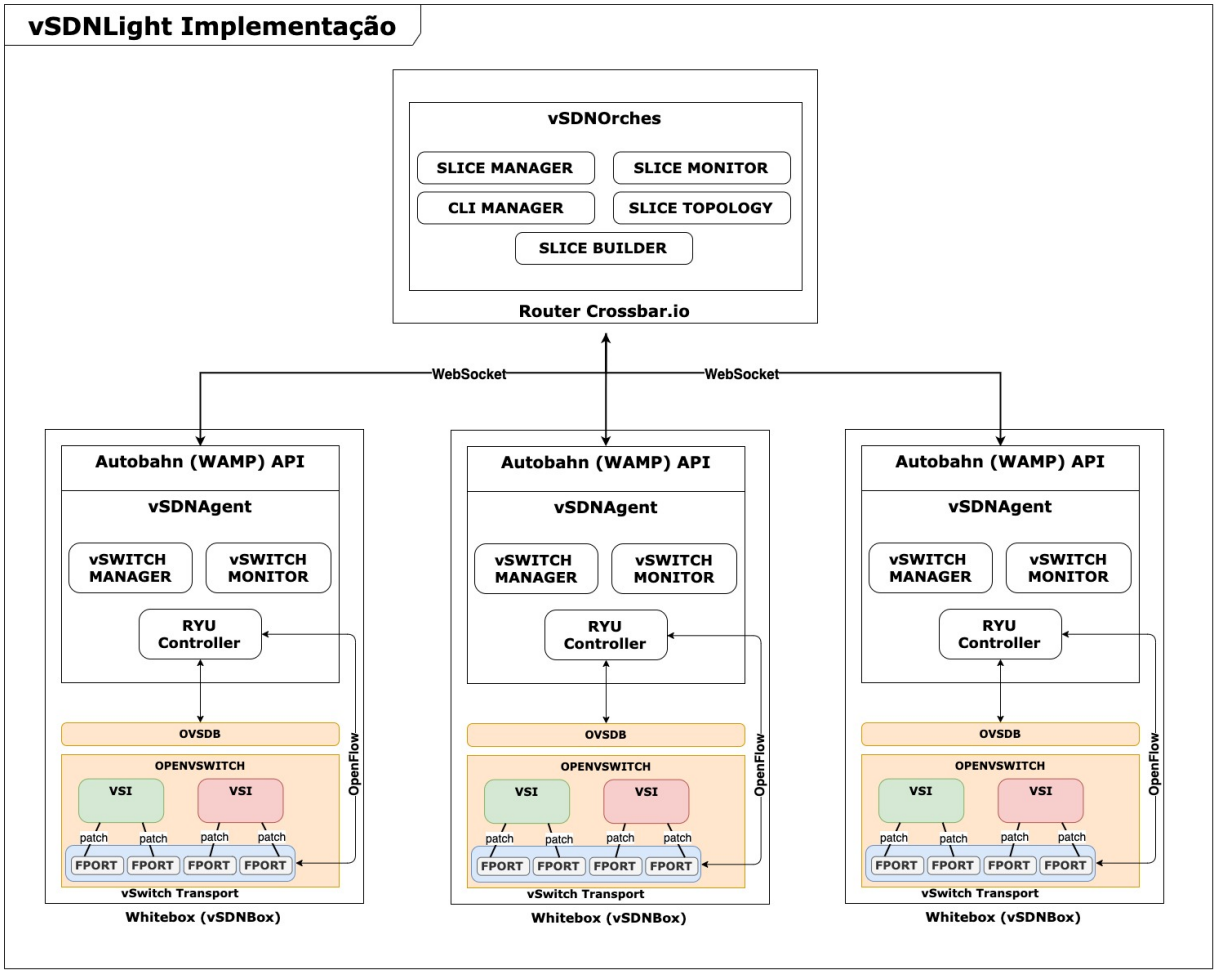
O *vSDNLight* apresenta uma proposta de provisionar redes virtuais definidas por softwares através de uma arquitetura leve. Sua arquitetura utiliza um novo modelo de slice baseado em instâncias virtuais de *switches* (VSI) que são criados em *Whiteboxes*, que constituem a infraestrutura SDN, através dos VSIs foi possível remove o modelo de *proxy* aplicados pelas soluções atuais. Sendo assim uma prova de conceito do *vSDNLight* foi desenvolvida e seu desempenho foi comparado com outras tecnologias de hipervisores SDN, como: *FlowVisor* e *OpenVirtex*. Esta comparação levou em consideração as métricas como: uso de CPU e Memória, latência e vazão.

5.2.1 Implementação do *vSDNLight*

O *vSDNLight* foi desenvolvido com base na plataforma *Crossbar.io*[7], um *framework* para aplicações distribuídas e micro serviços. Este *framework* implementa o protocolo WAMP[8] (*Web Application Message Protocol*) que permite suas aplicações trabalharem tanto com chamadas de procedimento remotos (RPC – *Remote Procedure Call*) quanto Pub/Sub (*Publish and Subscribe*). Para isso é necessário a utilização de um *Router Crossbar.io* que encaminha as mensagens entre as aplicações, e também, registra todos os tópicos, procedimentos e aplicações remotas.

A implementação da arquitetura, ilustrada na Figura 30, foi construída baseada com os módulos definidos na arquitetura genérica do Capítulo 4, o desenvolvimento foi feito em *Python* na versão 3.7 utilizando a biblioteca *Autobahn*[9] que permitiu a criação de módulos tanto remotos quanto embarcados ao *Router Crossbar.io*.

Figura 30 – Implementação do *vSDNLight*



Fonte: Autor

A arquitetura foi desenvolvida como prova de conceito nesta tese e está disponível em

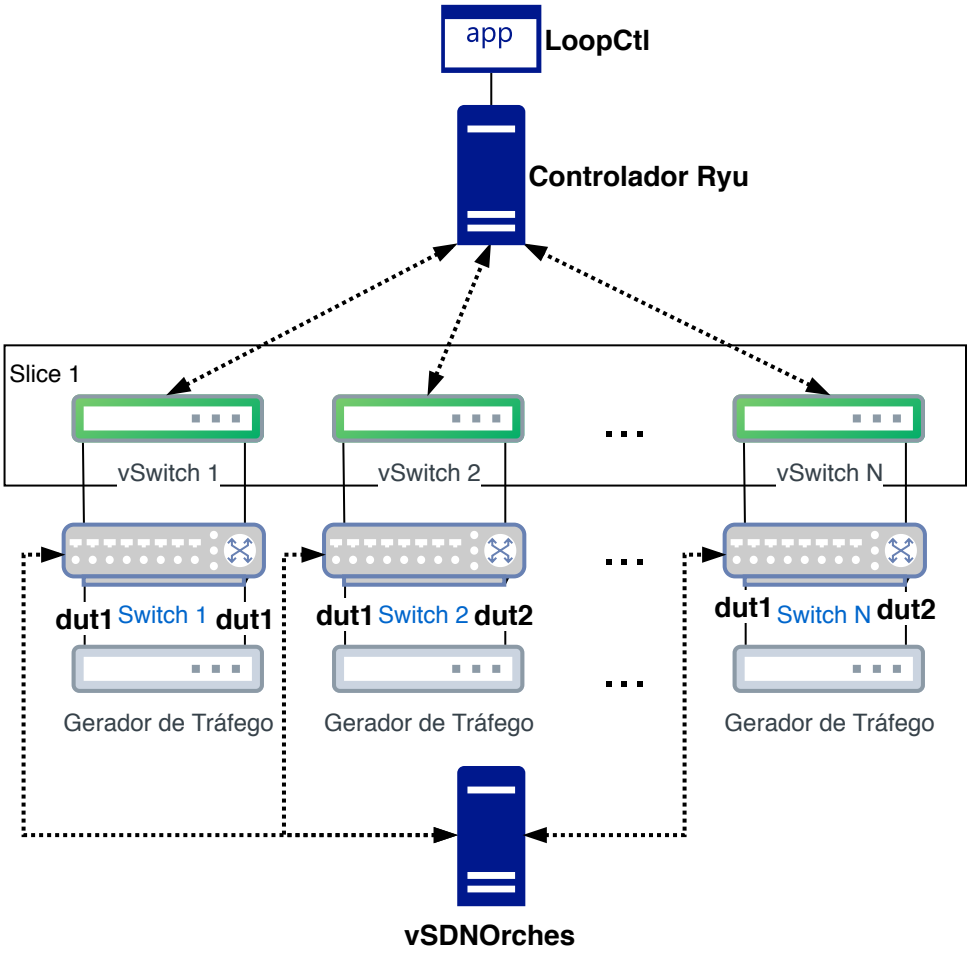
[7] Projeto Crossbar.io: https://crossbar.io/
[8] Projeto WAMP: https://wamp-proto.org/
[9] Projeto Autobahn: http://github.com/crossbario/autobahn-python

repositório do *GitHub*[10]. Atualmente, ela estão na versão 0.2. Além disso, a arquitetura está disponível também como um dos modelos do emulador *vSDNEmul*.

5.2.2 Metodologia

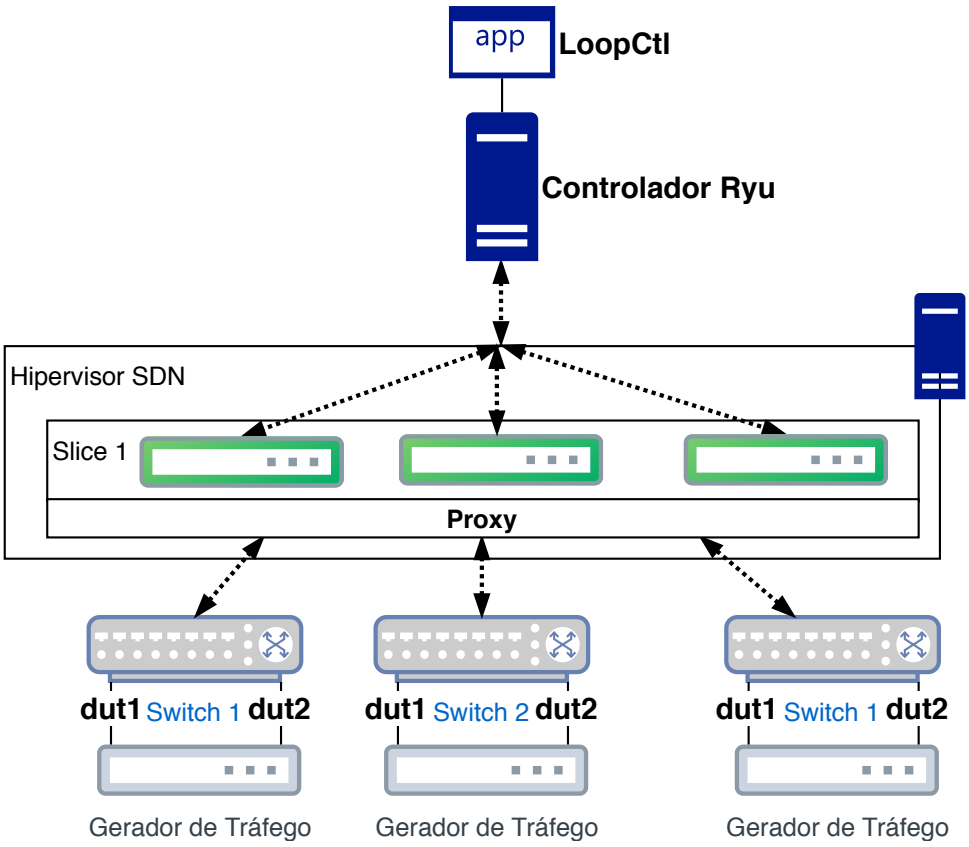
A metodologia utilizada para realizar à avaliação e validação da arquitetura *vSDNLight* foi organizada e desenvolvida no emulador *vSDNEmul*. O objetivo da avaliação é medir o comportamento da arquitetura, à medida que a escala de switches vai aumentando. Para à validação da proposta os resultados são comparados com outras duas soluções de hipervisores SDN: *OpenVirteX* e *Flowvisor*. A decisão de escolha dessas soluções se deu devido a sua disponibilização aberta e por serem as principais ferramentas de virtualização atualmente. A Figura 31 mostra a visão geral dos cenários criados para a realização dos experimentos.

Figura 31 – Visão dos cenários utilizados durante avaliação de desempenho



(a)

[10] Projeto vSDNAgent e vSDNOrches: http://github.com/fernnf/vsdnagent e http://github.com/fernnf/vsdnorches



(b)
Fonte: Autor

No primeiro cenário, representado pela Figura 31a, tem-se a avaliação da arquitetura proposta nesta tese, basicamente, o plano de dados é composto por *N* switches, onde *N* corresponde ao número de switches que foram variados durante os experimentos, neste caso, usou-se os valores de 1, 5, 10, 20, 30, 40 e 50 switches. Em cada experimento possui um slice contendo a mesma variação da quantidade de switches na infraestrutura, ou seja, a topologia do slice é uma cópia da topologia da infraestrutura. No segundo cenário ilustrado na Figura 31b, tem-se o cenário criado para avaliar os hipervisores SDN, e da mesma forma, usa-se os mesmo valores de switches por experimentos e utilizando-se um slice criado no hipervisor composto pelo elementos da infraestrutura.

Os experimentos são repetidos 15 vezes para cada valor de *N* switches, cada switches possui duas portas virtuais (DUT1 e DUT2), as quais há um gerador de pacotes vinculado, que as utilizam para enviar e receber pacotes durante as medições. Toda a metodologia aplicada na medição e coleta de resultados seguiram as informações estabelecida na RFC 8456 (BHUVANESWARAN et al., 2018) e no artigo de Jawaharan et al. (2018). Nos experimentos foram tomadas as seguintes métricas de desempenho aplicadas em cada quantidade de switches: i) CPU e Memória; e ii) Vazão e iii) Latência.

Para uso de CPU e memória, buscou-se verificar a quantidade de recursos necessários para funcionamento do slice durante os testes de vazão e latência de acordo com número de nós, basicamente, foram medidos os valores utilizados pela proposta desta tese em comparação com

os valores obtidos pelos hipervisores SDN (*OpenVirtex* e *Flowvisor*).

A latência é o tempo em que o controlador da rede virtual leva para processar os pacotes gerados no gerador de pacote sobre condições carga de geração baixa. Portanto, quando esse pacote chega na porta do DUT1 é gerado um PACKET-IN[11] que vai até o controlador e volta como PACKET-OUT[12], que encaminha o pacote para porta DUT2, Logo a latência é o tempo que gasto para que esse pacote que chega na porta DUT1 é encaminhado para DUT2.

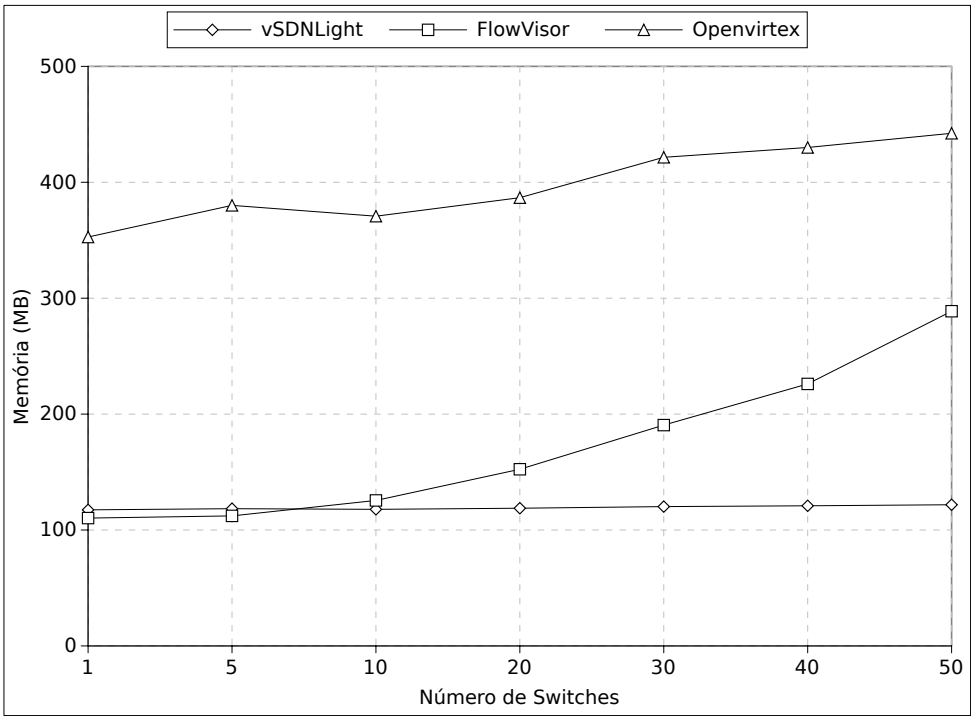
A vazão é a taxa máxima que um controlador consegue encaminhar pacotes da porta DUT1 para porta DUT2, ou seja, quantos PACKET-IN cada controlador consegue processar por segundo. Para isso, o gerador de pacotes bombardeia a porta DUT1 com uma quantidade de 10000 pacotes (JAWAHARAN et al., 2018), que representa a entrada de 10000 clientes no switch, e verifica a taxa de máxima de encaminhamentos que os pacotes chegam na porta DUT2.

Os experimentos foram realizados em um servidor IBM Blade com processador Intel Xeon de 2.50Ghz com 24 núcleos, memória RAM de 45 GB e sistema operacional Fedora Server 29.

5.2.3 Análise dos Resultados

Nas Figuras 32 e 33, tem-se os resultados de memória e CPU, e respectivamente medidos em valor de RAM em MB (*megabytes*) e percentual de uso.

Figura 32 – Uso de memória por número de switches

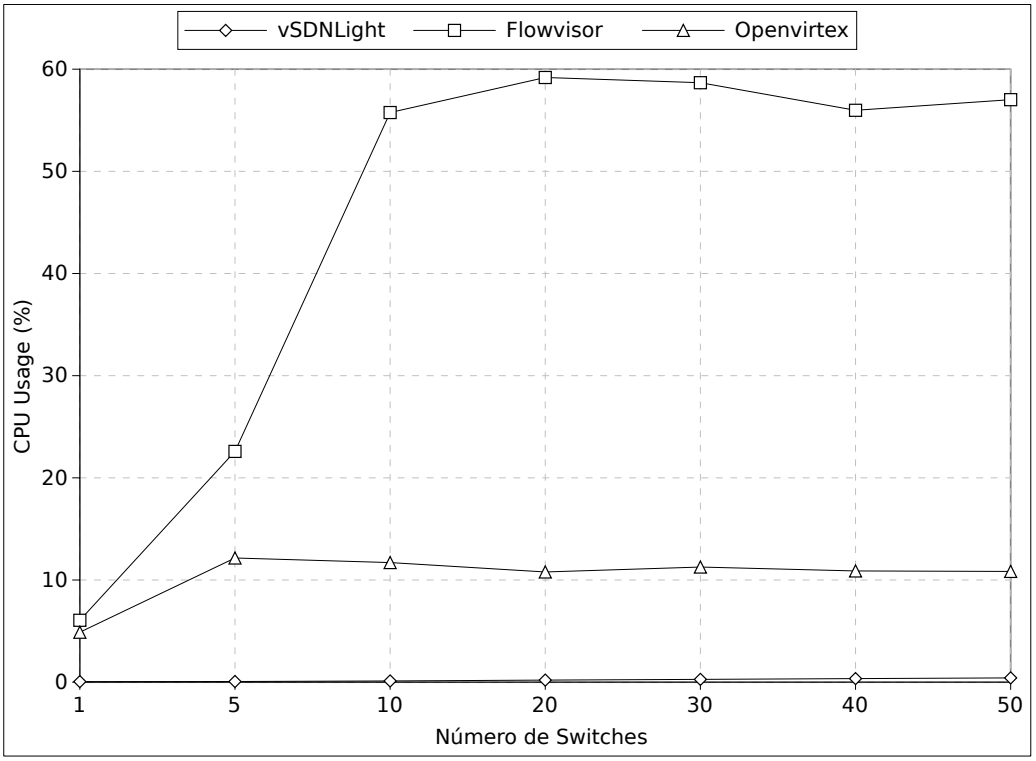


Fonte: Autor

[11] Mensagem Packet-in: http://flowgrammable.org/sdn/openflow/message-layer/packetin/
[12] Mensagem Packet-out: http://flowgrammable.org/sdn/openflow/message-layer/packetout/

A Figura 32 apresenta o consumo de memória do *vSDNLight* (i.e. *vSDNOrches*), *OpenVirtex* e *FlowVisor*. Durante os experimentos foram coletados o consumo de memória física utilizada pelo contêineres que executam cada solução. Nesta métrica, o *vSDNLight* manteve seu consumo de memória praticamente constante em torno de 121MB. Diferente das outras soluções, que conforme o número de switches ia aumentando o consumo de memória também vai aumentando. Por exemplo, para 50 switches o *Flowvisor* necessitou de 145MB a mais de memória que o *vSDNLight* para suportar a quantidade de switches, por outro lado, o OpenVirtex necessitou 339 MB para atender os mesmos 50 switches. Isto demonstra que a *vSDNLight* consegue consumir menos recursos a medida em que a escala de switches vai aumentando, validado a leveza da arquitetura.

Figura 33 – Uso de CPU por número de switches

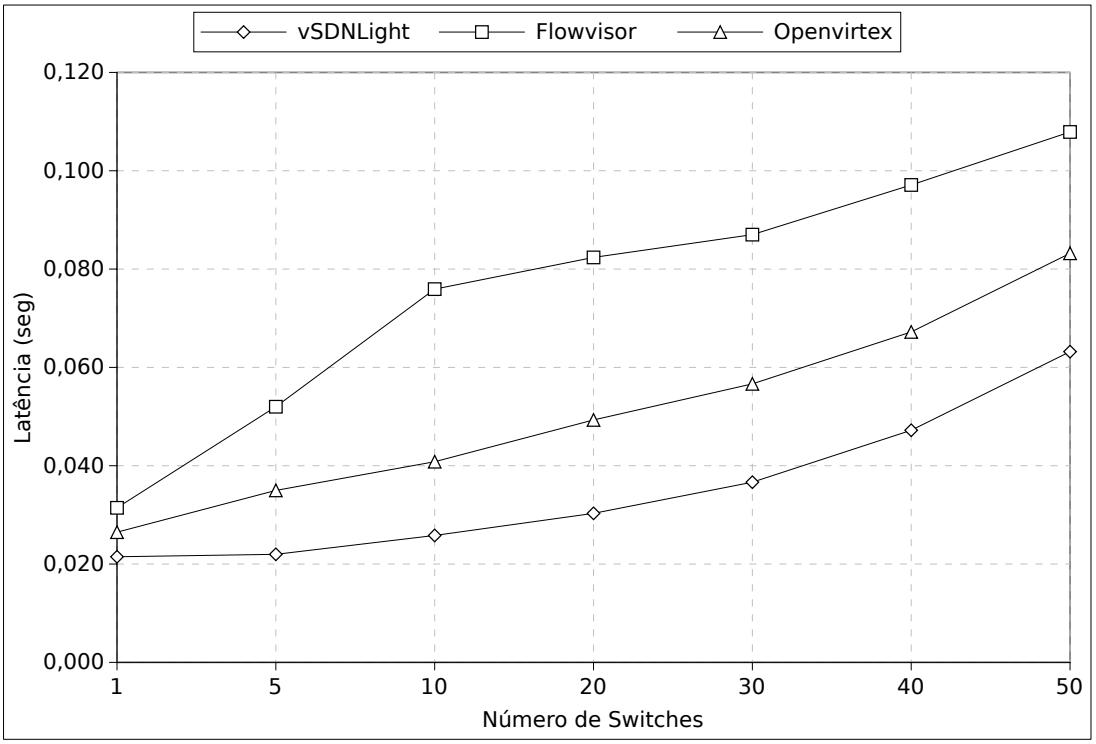


Fonte: Autor

A Figura 33 ilustra o uso de CPU entre as soluções avaliadas mediante a variação da escala de switches na infraestrutura. Novamente, o *vSDNLight* demonstrou ser mais leve que as outra soluções, mesmo para 50 switches, onde o seu consumo de CPU ficou em torno de 0,2%, que ao compara-lo com as demais soluções percebe-se que o *FlowVisor* precisou de 57,8% e o *OpenVirteX* necessitou de 9,6%, a mais de CPU, para suportarem mesma escala de 50 switches.

As métricas de CPU e memória demonstram que a arquitetura proposta consegue ser mais leve que as demais soluções avaliadas, mesmo com a variação do aumento da escala da infraestrutura. Além disso, ele também demonstra o quanto de recurso é desperdiçado pelo modelo de *proxy* em ações que, necessariamente, não precisão estar no plano de controle.

Figura 34 – A latência por número de switches

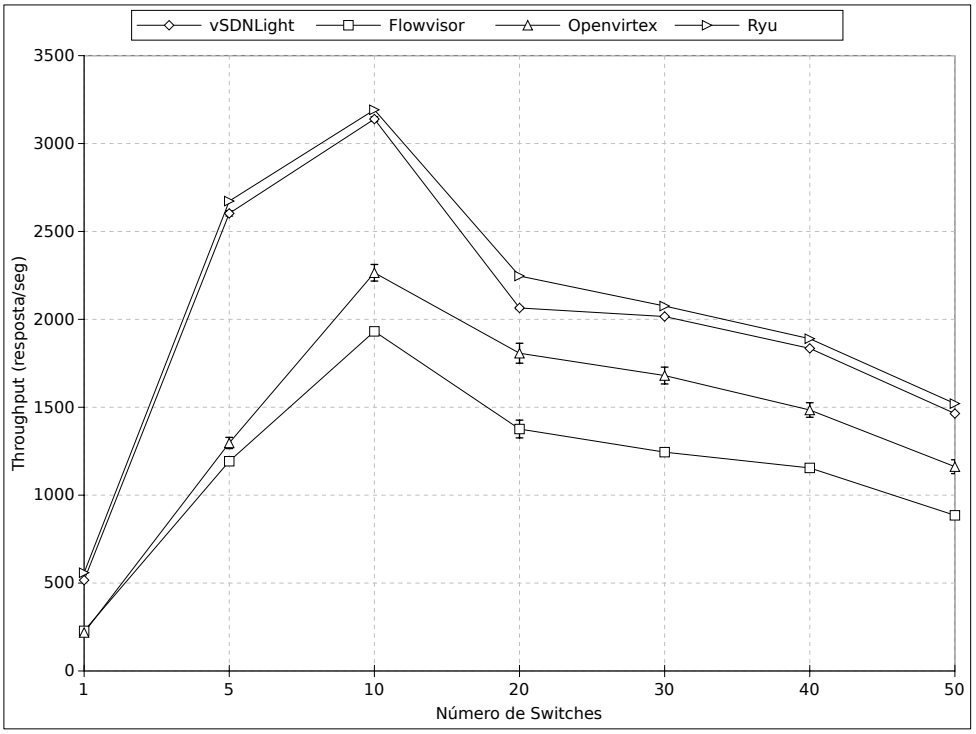


Fonte: Autor

A Figura 34 mostra os resultados de latência das soluções avaliadas. Ela apresenta que a arquitetura conseguiu diminuir 53% (em torno de 0,06 segundos) o tempo de latência quando comparado ao solução *FlowVisor*, e de 29,7% (em torno de 0,022 segundos) em relação ao *OpenVirtex*, a uma escala de 50 switches. Demonstrando que remoção do modelo de proxy do plano de controle trouxe vantagens para latência do slice, pois os switches virtuais interagem diretamente com o controlador da rede virtual, sem a interferência do hipervisor.

A Figura 35 mostra o número de respostas que o controlador da rede virtual consegue responder por segundo. No gráfico percebe-se que o limite do controlador ryu é atingindo próximo de 10 switches algo em torno de 3100 respostas por segundo atingido pelo arquitetura *vSDNLight*, a partir deste ponto, observa-se uma queda de desempenho do controlador de rede virtual. Especificamente, neste experimento foi realizado a análise de desempenho do controlador interagindo diretamente com o switches para observar se esse comportamento das soluções ou do controlador em si. Os resultados demonstram que o *vSDNLight* contribui com ganhos para vazão diminuindo o *overhead* aplicado pelo modelo de proxy. Onde, para 50 switches, a proposta *vSDNLight* conseguiu aumentar em 24,4% o ganho em vazão quando comparado à solução *OpenVirtex* e 45,7% de ganho quando comparado à solução *FlowVisor*.

Figura 35 – A vazão por número de switches



Fonte: Autor

Os resultados apresentados nesta seção permitem concluir que proposta *vSDNLight* consegue se tornar uma arquitetura leve e escalável quando comparada a outras soluções que promovem a virtualização em redes SDN. Eles também demonstram que é possível implementar uma solução de virtualização em redes SDN sem precisar usar o modelo slice baseado em proxy, implementado pelas soluções atuais, e ter vantagens de desempenho com isso. Além disso, o modelo de slice baseado em VSI traz mais escalabilidade para o tamanho do slice, sem comprometer a orquestração em termos de recurso utilizados ou sobrecargas para gerenciar o tamanho do mesmo. Adicionalmente, a partir do uso do modelo baseado em VSI, a arquitetura ajudou a diminuir o *overhead* aplicado pelo modelo de proxy, porém, o desempenho deste slice vai depender também do controlador a ser utilizado, pois as instâncias virtuais vinculadas ao *slice* estão diretamente ligadas ao seu controlador.

5.3 Conclusão do Capítulo

Neste capítulo, apresentou-se a validação e análise de proposta principal e da proposta suplementar, ambas definidas no Capítulo 4, que correspondem às respostas das questões apresentadas no Capítulo 1. Para o vSDNEmul foi demonstrado que emulador consegue ter desempenho e escalabilidade igual a principal solução de emulador de redes SDN (i.e., *Mininet*) com a qual foi comparada, no entanto, é possível concluir que a fidelidade das emulações do *vSDNEmul* é maior que as proporcionadas pelo *Mininet*. Já para o *vSDNLight* foi exposto uma prova de conceito e comparada com outras soluções que promovem virtualização em redes SDN. Os

resultados demostraram que a arquitetura proposta nesta tese consegue ser leve e escalável através de comparações medindo seu uso de CPU e memória e seu desempenho através da avaliação da latência e vazão. Também foi possível concluir que o *vSDNLight* remove o *overhead* aplicado pelo modelo de proxy e o converte em vantagens de desempenho em relação as métrica avaliadas.

6 CONSIDERAÇÕES FINAIS E TRABALHOS FUTUROS

Neste capítulo são apresentados as reflexões e conclusões sobre o que foi desenvolvido nesta tese, bem como, as respostas as questões de pesquisas apresentadas no Capítulo 1, sugestões de trabalhos futuros a serem desenvolvidos e a lista de trabalhos publicados.

6.1 Conclusões Gerais

Durante o desenvolvimento desta tese a seguinte questão principal foi levantada: ***Como oferecer uma arquitetura de virtualização em SDN de forma leve e escalável?*** Para responder esta questão, elaborou-se a seguinte hipótese: ***As limitações encontradas nas vSDNs podem ser reduzidas através de uma arquitetura leve e distribuída, contribuído com o aumento na vazão de fluxos nas vSDNs e o baixo consumo de CPU e memória para gerência das mesmas***. Para validar a da hipótese definiu-se arquitetura leve e distribuída é aquela que remove a necessidade de proxy da arquitetura através da redução das atribuições do hipervisor e a distribuiu para os elementos de redes no plano de dados. Além disso, a arquitetura prove um modelo de *slice* baseado VSI que transfere a camada de virtualização para plano de dados, providos através *switches whiteboxes* que fornecem instâncias, de forma, que a união dessas instâncias criam o novo modelo de *slice*.

A principal contribuição da tese consistiu na proposta de arquitetura para provisionamento de vSDNs, que rompe com o modelo de *proxy* da soluções atuais e aplica um modelo mais leve e distribuído, trazendo como vantagem da proposta a melhora no desempenho das vSDNs. A proposta foi validada através da implementação e realização de experimentos via emulador *vSDNEmul*. A principal vantagem percebida, está na diminuição do *overhead* empregado pelo modelo de *proxy* e refletindo em ganhos de desempenho pela arquitetura. Dessa forma, as limitações da soluções atuais, que eram altas taxa de uso memória e CPU e barreiras nos desempenho de vazão e latência foram superadas.

Como outra vantagem da arquitetura, pode-se destacar que mesmo com aumento da escala de equipamentos na infraestrutura, não houve necessidade de obtenção de mais recurso computacionais para atender a demanda de switches. Além disso, a aplicação de instâncias virtuais não provocou grandes perdas de desempenho, contribuindo muito pouco para degradação das métricas avaliadas (vazão e latência).

O principal obstáculo para adoção da arquitetura é a necessidade de suporte da infraestrutura a equipamentos que permitam a virtualização de switches, sendo que atualmente, a arquitetura não suporta uma solução hibrida que aceite a utilização de equipamentos com e sem suporte à virtualização de switches. No entanto, com o uso cada vez maior de *whiteboxes* pela principais infraestruturas, esse problema passa a ser superado.

As duas soluções desenvolvidas nesta tese: *vSDNEmul* e *vSDNLight* suportam a hipótese

deste trabalho ao demonstrar um ambiente que foi capaz de avaliar a proposta de forma escalável e fiel ao comportamento real de uma infraestrutura de rede, esse comportamento foi validado através de experimentos realizado com emulador, respondendo assim a questão suplementar. Além disso, os resultado obtidos através de experimentos na arquitetura demonstram que o *vSDNLight* é capaz de oferecer uma solução leve e escalável ao remover o hipervisor do plano de controle, que é o principal elemento de sobrecarga das soluções atuais, respondendo assim à questão principal de pesquisa.

6.2 Trabalhos Futuros

A proposta *vSDNLight* foi desenvolvida para oferece uma nova forma de provisionamento de vSDNs, o qual o foco principal do trabalho. No entanto, com a disponibilidade desta arquitetura pode-se alavancar novas possibilidades de estudos em áreas não exploradas nesta tese:

Migração de slice: No vSDNLight a alocação do slice pode ser on-demand, ou seja, arquitetura decide em quais pontos da infraestrutura será feita a alocação da rede virtual. No entanto, por motivos de escassez de recurso disponíveis nos switches, pode haver a necessidade de realocar a rede virtual em outros pontos da infraestrutura promovendo a migração de redes virtuais.

Elasticidade de slice: É uma característica muito necessária na arquitetura permitindo o aumento e a redução de slice em tempo de execução. Mediante a demanda do administrador para adoção e remoção de um ou mais nós do slice.

Inteligência artificial: O uso de inteligência artificial para a alocação e realocação de slice de forma inteligente levando em consideração métricas de rede disponíveis na infraestrutura. Podendo explorar o aprendizado de máquina (*Machine Learning*) e aprendizado profundo (*Deep Learning*).

6.3 Contribuições em Produção Bibliográfica

- ✓ Artigo em desenvolvimento para submissão em periódico com os últimos resultados da avaliação da arquitetura.
- ✓ Artigo publicado no *International Journal of Simulation Systems, Science and Technology* (2019) - *vSDNEmul: A Software-Defined Network Emulator Based on Container Virtualization* (FARIAS et al., 2019)
- ✓ Artigo publicado e apresentado nos anais do Workshop de Pesquisa Experimental da Internet do Futuro – WPEIF – SBRC (2019): vSDNLight: Uma Proposta de Arquitetura

Leve para Provisionamento de Redes Virtuais Definidas por Software (FARIAS; ABELÉM, 2019)

- ✓ Artigo publicado e apresentado nos anais do Workshop de Pesquisa Experimental da Internet do Futuro – WPEIF – SBRC (2018): vSDNEmul: Emulador de Redes Definidas Por Software Usando Contêineres (FARIAS; SALVADOR; ABELÉM, 2018)

- ✓ Artigo publicado e apresentado nos anais do Workshop de Gerência e Operação de Redes e Serviços – WGRS – SBRC (2018): vSDNBox: Um Hardware Especializado de Baixo Custo Gerenciado via SDN (FARIAS et al., 2018)

6.4 Contribuições Tecnológicas

Atualmente, alguns componentes da arquitetura vSDNLight fazem parte do projeto IDS[1] (Infraestruturas Definidas por Softwares) da RNP (Redes Nacional de Pesquisa). E estão em fase implantação na infraestrutura de produção do FIBRE para suporte a nova versão da rede de experimentação FIBRENET.

No Backbone da FIBRENET estão sendo incorporados os modelos de *whiteboxes* baseados em *software-switched* e descrito nesta tese. Atualmente, esses *whiteboxes*, já estão em sua terceira geração com processadores *Xeon* e suporte a interfaces de 10 *Gbps ethernet*. Além disso, o modelo de *slice* baseado em VSIs está sendo implantado também neste *backbone*.

[1] Projeto IDS: https://wrnp.rnp.br/sites/wrnp2019/files/IDS.pdf

REFERÊNCIAS

AHMED, M. F.; TALHI, C.; CHERIET, M. Towards flexible, scalable and autonomic virtual tenant slices. In: **Proceedings of the 2015 IFIP/IEEE International Symposium on Integrated Network Management, IM 2015**. [S.l.: s.n.], 2015. p. 720–726. ISBN 9783901882760.

AL-SHABIBI, A. et al. OpenVirteX: Make Your Virtual SDNs Programmable. In: **Proceedings of the Third Workshop on Hot Topics in Software Defined Networking**. [s.n.], 2014. p. 25–30. ISBN 978-1-4503-2989-7. Disponível em: <http://doi.acm.org/10.1145/2620728.2620741>.

Anish Babu, S. et al. System Performance Evaluation of Para Virtualization, Container Virtualization, and Full Virtualization Using Xen, OpenVZ, and XenServer. In: **2014 Fourth International Conference on Advances in Computing and Communications**. IEEE, 2014. p. 247–250. ISBN 978-1-4799-4363-0. Disponível em: <http://ieeexplore.ieee.org/document/6906035/>.

BANNIZA, T.-R. et al. A European approach to a clean slate design for the future internet. **Bell Labs Technical Journal**, v. 14, n. 2, p. 5–22, aug 2009. ISSN 10897089. Disponível em: <http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6781205>.

BENSON, T.; AKELLA, A.; MALTZ, D. Unraveling the complexity of network management. In: **Proceedings of the 6th USENIX Symposium on Networked Systems Design and Implementation**. Berkeley, CA, USA: NSDI'09, 2009. p. 335—-348. Disponível em: <https://dl.acm.org/citation.cfm?id=1559000>.

BERMAN, M.; ELLIOTT, C.; LANDWEBER, L. GENI: Large-scale distributed infrastructure for networking and distributed systems research. In: **2014 IEEE 5th International Conference on Communications and Electronics, IEEE ICCE 2014**. [S.l.: s.n.], 2014. p. 156–161. ISBN 9781479950515.

BHUVANESWARAN, V. et al. **Benchmarking Methodology for Software-Defined Networking (SDN) Controller Performance**. [S.l.], 2018. Disponível em: <https://datatracker.ietf.org/doc/rfc8456https://www.rfc-editor.org/info/rfc8456>.

BLENK, A.; BASTA, A.; KELLERER, W. HyperFlex: An SDN virtualization architecture with flexible hypervisor function allocation. In: **Proceedings of the 2015 IFIP/IEEE International Symposium on Integrated Network Management, IM 2015**. [S.l.: s.n.], 2015. p. 397–405. ISBN 9783901882760. ISSN 1573-0077.

BLENK, A. et al. Survey on network virtualization hypervisors for software defined networking. **IEEE Communications Surveys and Tutorials**, v. 18, n. 1, p. 655–685, 2016. ISSN 1553877X.

BLENK, A. et al. Control Plane Latency with SDN Network Hypervisors: The Cost of Virtualization. **IEEE Transactions on Network and Service Management**, v. 13, n. 3, p. 366–380, 2016. ISSN 19324537.

BOURAS, C.; KOLLIA, A.; PAPAZOIS, A. Teaching 5G networks using the ONOS SDN controller. In: **2017 Ninth International Conference on Ubiquitous and Future Networks (ICUFN)**. IEEE, 2017. p. 312–317. ISBN 978-1-5090-4749-9. Disponível em: <http://ieeexplore.ieee.org/document/7993800/>.

CASADO, M.; FOSTER, N.; GUHA, A. Abstractions for software-defined networks. **Communications of the ACM**, ACM, v. 57, n. 10, p. 86–95, sep 2014. ISSN 00010782. Disponível em: <http://dl.acm.org/citation.cfm?doid=2661061.2661063>.

CHEN, H.; BENSON, T. Switch-visor: towards infrastructure-level virtualization of SDN switches. In: **Proceedings of the 2nd Workshop on Cloud-Assisted Networking - CAN '17**. New York, New York, USA: ACM Press, 2017. p. 25–30. ISBN 9781450354233. Disponível em: <http://dl.acm.org/citation.cfm?doid=3155921.3158431>.

CHOWDHURY, N. M. K.; BOUTABA, R. A survey of network virtualization. **Computer Networks**, Elsevier B.V., v. 54, n. 5, p. 862–876, 2010. ISSN 13891286. Disponível em: <http://dx.doi.org/10.1016/j.comnet.2009.10.017>.

CORIN, R. D. et al. VeRTIGO: Network virtualization and beyond. In: **European Workshop on Software Defined Networks, EWSDN 2012**. [S.l.: s.n.], 2012. p. 24–29. ISBN 9780769548708. ISSN 2379-0350.

DRUSTSKOY, D.; KELLER, E.; REXFORD, J. Scalable Network Virtualization in Software-Defined Networks. p. 13, 2013.

FARIAS, F. N. N.; ABELÉM, A. J. G. vSDNLight: Uma Proposta de Arquitetura Leve para Provisionamento de Redes Virtuais Definidas por Softwares. **Gramado| RS**, SBC, p. 20, sep 2019. ISSN 2595-2692.

FARIAS, F. N. N. et al. vSDNEmul: A Software-Defined Network Emulator Based on Container Virtualization. **International Journal of Simulation Systems, Science & TechnologySimulation**, v. 20, n. 4, p. 7.1, 2019. ISSN 1473-8031.

FARIAS, F. N. N. et al. Pesquisa Experimental para a Internet do Futuro: Uma Proposta Utilizando Virtualização e o Framework Openflow. In: **Anais do Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos**. [s.n.], 2011. p. 1–62. Disponível em: <http://sbrc2011.facom.ufms.br/files/mc/mc1.pdf>.

FARIAS, F. N. N.; SALVADOR, P. M.; ABELÉM, A. J. G. vSDNEmul: Emulador de Redes Definidas Por Software Usando Contêineres. In: **Anais do IX Workshop de Pesquisa Experimental da Internet do Futuro (WPEIF - SBRC 2018)**. SBC, 2018. v. 9. Disponível em: <https://portaldeconteudo.sbc.org.br/index.php/wpeif/article/view/2320>.

FARIAS, F. N. N. et al. vSDNBox: Um Hardware Especializado de Baixo Custo Gerenciado via SDN. In: **Anais do XXIII Workshop de Gerência e Operação de Redes e Serviços (WGRS - SBRC 2018)**. SBC, 2018. v. 23. Disponível em: <http://portaldeconteudo.sbc.org.br/index.php/wgrs/article/view/2687>.

FIBRE. **Future Internet Brazilian Environment For Experimentation**. 2019. Disponível em: <https://fibre.org.br/>.

Galan Jimenez, J.; Gazo Cervero, A. Overview and Challenges of Overlay Networks: A Survey. **International Journal of Computer Science & Engineering Survey**, Academy and Industry Research Collaboration Center (AIRCC), v. 2, n. 1, p. 19–37, feb 2011. ISSN 09763252.

HALEPLIDIS, E. et al. **Software-Defined Networking (SDN): Layers and Architecture Terminology**. 2015. Disponível em: <https://rfc-editor.org/rfc/rfc7426.txt>.

HAN, Y. et al. ONVisor: Towards a scalable and flexible SDN-based network virtualization platform on ONOS. **International Journal of Network Management**, v. 28, n. 2, p. 1–20, 2018. ISSN 10991190.

HANDIGOL, N. et al. Reproducible network experiments using container-based emulation. In: **Proceedings of the 8th international conference on Emerging networking experiments and technologies - CoNEXT '12**. [s.n.], 2012. p. 253. ISBN 9781450317757. Disponível em: <http://dl.acm.org/citation.cfm?doid=2413176.2413206>.

HIBLER, M. et al. Large-scale Virtualization in the Emulab Network Testbed. **Proceedings of the 2008 USENIX Annual Technical Conference**, p. 113—-128, 2008.

HU, F.; HAO, Q.; BAO, K. A survey on software-defined network and OpenFlow: From concept to implementation. **IEEE Communications Surveys and Tutorials**, v. 16, n. 4, p. 2181–2206, 2014. ISSN 1553877X. Disponível em: <http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6819788>.

IEEE 802.1Q. **IEEE standard for local and metropolitan area networks– bridges and bridged networks. Amendment 23, Application Virtual Local Area Network (VLAN) Type, Length, Value (TLV)**. [s.n.], 2015. 89 p. ISBN 9780738195650. Disponível em: <https://ieeexplore.ieee.org/document/7062078>.

ISAIA, P.; GUAN, L. Performance benchmarking of SDN experimental platforms. In: **IEEE NETSOFT 2016 - 2016 IEEE NetSoft Conference and Workshops: Software-Defined Infrastructure for Networks, Clouds, IoT and Services**. [S.l.]: Institute of Electrical and Electronics Engineers Inc., 2016. p. 116–120. ISBN 9781467394864.

JAWAHARAN, R. et al. Empirical evaluation of SDN controllers using mininet/wireshark and comparison with cbench. In: **Proceedings - International Conference on Computer Communications and Networks, ICCCN**. [S.l.]: Institute of Electrical and Electronics Engineers Inc., 2018. v. 2018-July. ISBN 9781538651568. ISSN 10952055.

KREUTZ, D. et al. Software-Defined Networking : A Comprehensive Survey. **Proceedings of the IEEE**, v. 103, n. 1, p. 14 – 76, 2015. ISSN 0018-9219. Disponível em: <http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6994333>.

KUMAR, K.; KURHEKAR, M. Economically Efficient Virtualization over Cloud Using Docker Containers. In: **2016 IEEE International Conference on Cloud Computing in Emerging Markets (CCEM)**. IEEE, 2016. p. 95–100. ISBN 978-1-5090-4573-0. Disponível em: <http://ieeexplore.ieee.org/document/7819678/>.

LANTZ, B.; HELLER, B.; MCKEOWN, N. A network in a laptop. **Proceedings of the Ninth ACM SIGCOMM Workshop on Hot Topics in Networks - Hotnets '10**, p. 1–6, 2010. ISSN 1450304095. Disponível em: <http://portal.acm.org/citation.cfm?doid=1868447.1868466>.

LARA, A.; KOLASANI, A.; RAMAMURTHY, B. Network Innovation using OpenFlow: A Survey. **IEEE Communications Surveys & Tutorials**, PP, n. 99, p. 1–20, 2013. ISSN 1553-877X. Disponível em: <http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6587999>.

MCKEOWN, N. et al. OpenFlow. **ACM SIGCOMM Computer Communication Review**, ACM, v. 38, n. 2, p. 69, mar 2008. ISSN 01464833. Disponível em: <http://portal.acm.org/citation.cfm?doid=1355734.1355746>.

MOREIRA, M. D. D. et al. Internet do futuro: Um novo horizonte. In: **Minicursos do Simpósio Brasileiro de Redes de Computdores e Sistemas Distribuídos**. [s.n.], 2009. p. 1–59. Disponível em: <http://www.lbd.dcc.ufmg.br/colecoes/sbrc/2009/080.pdf>.

Nawej, C. M.; Du, S. Virtual private network's impact on network performance. In: **2018 International Conference on Intelligent and Innovative Computing Applications (ICONIC)**. [S.l.: s.n.], 2018. p. 1–6.

NUNES, B. A. A. et al. A survey of software-defined networking: Past, present, and future of programmable networks. **IEEE Communications Surveys and Tutorials**, v. 16, n. 3, p. 1617–1634, 2014. ISSN 1553877X.

PEUSTER, M.; KAMPMEYER, J.; KARL, H. Containernet 2.0: A Rapid Prototyping Platform for Hybrid Service Function Chains. In: **2018 4th IEEE Conference on Network Softwarization and Workshops (NetSoft)**. IEEE, 2018. p. 335–337. ISBN 978-1-5386-4633-5. Disponível em: <https://ieeexplore.ieee.org/document/8459905/>.

PFAFF, B.; DAVIE, B. **The Open vSwitch Database Management Protocol**. RFC Editor, 2013. RFC 7047. (Request for Comments, 7047). Disponível em: <https://rfc-editor.org/rfc/rfc7047.txt>.

RANA, D. S.; DHONDIYAL, S. A.; CHAMOLI, S. K. Software Defined Networking (SDN) Challenges, issues and Solution. **International Journal of Computer Sciences and Engineering**, ISROSET: International Scientific Research Organization for Science, Engineering and Technology, v. 7, n. 1, p. 884–889, jan 2019.

REXFORD, J.; DOVROLIS, C. Future Internet architecture: Clean Slate Versus Evolutionary Research. **Communications of the ACM**, v. 53, n. 9, p. 36, 2010. ISSN 00010782. Disponível em: <http://portal.acm.org/citation.cfm?doid=1810891.1810906>.

ROY, A. R. et al. Design and management of DOT: A Distributed OpenFlow Testbed. In: **2014 IEEE Network Operations and Management Symposium (NOMS)**. IEEE, 2014. p. 1–9. ISBN 978-1-4799-0913-1. Disponível em: <http://ieeexplore.ieee.org/document/6838241/>.

SALVADORI, E.; CORIN, R. D. Generalizing Virtual Network Topologies in OpenFlow- - Based Networks Roteiro. In: **2011 IEEE Global Telecommunications Conference (GLOBECOM 2011),**. [S.l.: s.n.], 2012. ISBN 9781424492688.

SHERWOOD, R. et al. FlowVisor: A Network Virtualization Layer. In: **Network**. [s.n.], 2009. p. 15. ISBN 9781450307970. ISSN 0717-6163. Disponível em: <http://www.techrepublic.com/whitepapers/flowvisor-a-network-virtualization-layer/2382721>.

SHIN, M. K.; NAM, K. H.; KIM, H. J. Software-defined networking (SDN): A reference architecture and open APIs. **International Conference on ICT Convergence**, p. 360–361, 2012. ISSN 21621233.

Sindhura, B.; Shobha, K. R. Implementation and testing of openflow switch using fpga. In: **2017 8th International Conference on Computing, Communication and Networking Technologies (ICCCNT)**. [S.l.: s.n.], 2017. p. 1–5.

STUCKMANN, P.; ZIMMERMANN, R. European research on future Internet design. **IEEE Wireless Communications**, v. 16, n. 5, p. 14–22, oct 2009. ISSN 1536-1284. Disponível em: <http://ieeexplore.ieee.org/document/5300298/>.

WAN, M.; YIN, S. Future internet architecture and cloud ecosystem: A survey. In: **AIP Conference Proceedings**. [S.l.]: American Institute of Physics Inc., 2018. v. 1955. ISBN 9780735416543. ISSN 15517616.

WANG, S. Y.; CHOU, C. L.; YANG, C. M. EstiNet openflow network simulator and emulator. **IEEE Communications Magazine**, IEEE, v. 51, n. 9, p. 110–117, 2013. ISSN 01636804.

WETTE, P.; DRÄXLER, M.; SCHWABE, A. MaxiNet: Distributed emulation of software-defined networks. In: **2014 IFIP Networking Conference, IFIP Networking 2014**. [S.l.: s.n.], 2014. ISBN 9783901882586.

YAN, J.; JIN, D. A lightweight container-based virtual time system for software-defined network emulation. **Journal of Simulation**, Palgrave Macmillan Ltd., v. 11, n. 3, p. 253–266, aug 2017. ISSN 17477786.

ZHENG, P.; NI, L. EMPOWER: a network emulator for wireline and wireless networks. In: **IEEE INFOCOM 2003. Twenty-second Annual Joint Conference of the IEEE Computer and Communications Societies (IEEE Cat. No.03CH37428)**. IEEE, 2003. v. 3, p. 1933–1942. ISBN 0-7803-7752-4. Disponível em: <http://ieeexplore.ieee.org/document/1209215/>.