UNIVERSIDADE FEDERAL DO PARÁ
INSTITUTO DE TECNOLOGIA
PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA

UMA ABORDAGEM SDN PARA VIRTUALIZAÇÃO DE REDES

BILLY ANDERSON PINHEIRO

TD 19/2016

UFPA / ITEC / PPGEE
Campus Universitário do Guamá
Belém-Pará-Brasil
2016

UNIVERSIDADE FEDERAL DO PARÁ
INSTITUTO DE TECNOLOGIA
PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA

BILLY ANDERSON PINHEIRO

UMA ABORDAGEM SDN PARA VIRTUALIZAÇÃO DE REDES

Tese submetida à Banca Examinadora do Programa de Pós-Graduação em Engenharia Elétrica da UFPA para a obtenção do Grau de Doutor em Engenharia Elétrica na área de Computação Aplicada

Orientador: Dr. Antônio J. G. Abelém

UFPA / ITEC / PPGEE
Campus Universitário do Guamá
Belém-Pará-Brasil
2016

Agradecimentos

Imaginem, no início da década de 90, em uma cidade do interior, de um estado que já é uma zona periférica do Brasil, professoras trabalhando em uma escola pública têm a sensibilidade de dar atenção a um garoto e viabilizar provas e documentações para fazer com que ele entrasse um ano antes na primeira série do fundamental, pois perceberam que ele perdia facilmente a atenção e começava a arrumar problemas.

Imaginem um garoto usar as férias para frequentar aulas particulares, pois ele, aos 10 anos, iria fazer um prova e concorrer com outras 300 crianças por vagas, naquela que era a escola pública boa da cidade.

Imaginem o garoto estudar, agora em uma escola privada, de 13h30min as 20h15min todos os dias, preparando-se pra outra prova, muito pior que aquela que ele fez 7 anos atrás. Tendo em média 3 professores para cada disciplina e, dessa vez, todo o material de apoio necessário.

Imaginem o garoto entrar na universidade e ter que viajar 2 horas para ir e 2 horas para voltar, todos os dias, de sua cidade até a capital... isso por 4 anos. Contando com o apoio financeiro do governo para as passagens e com o dinheiro dos pais para todo o resto, para que assim ele não precisasse se preocupar e pudesse se concentrar apenas nos estudos.

Imaginem esse garoto largando o curso técnico que fazia concomitantemente à graduação para poder fazer estágio e ter seu próprio dinheiro... ele não precisava... mas ao mesmo tempo ele "precisava".

Imaginem o garoto conseguindo um orientador que, talvez por ainda ser bastante novo, entenda bem as diferenças, deixa o garoto livre. Deixa-o crescer como queria... para que talvez um dia ele floresça.

Imaginem a quantidade de professores que passaram pela vida desse garoto. Alguns ele encontra pelos bares da vida, outro ele já encontrou em posições invertidas de aluno e professor, outros ele nunca reencontrou... mas cada um tem uma parcela de

culpa no que o garoto se tornou.

Imaginem o número de privilégios que esse garoto teve, pais com dinheiro no momento certo para pagar aula particular, para pagar escola privada, para pagar a alimentação...

Imaginem a quantidade de sorte que o garoto teve ao encontrar inúmeras pessoas que o ajudaram, desde as professoras quando ele tinha 6 anos... até os amigos que revisam seus textos e discutem suas ideias até hoje.

Imaginem se todos pudessem ter o mesmo que este garoto, se todos tivessem as oportunidades, se todos tivessem o tempo livre para estudar, se todos tivessem a melhor escola, se todos tivessem os melhores materiais, se todos tivessem a sorte de nascer em uma família que usava seus poucos recursos para a educação do filho, imaginem...

São tantas coisas para imaginar que, hoje, o garoto, apesar do esforço que fez... tem mais dívidas que méritos. Ele deve ao governo, àquela entidade que deveria redistribuir e prover a mesma oportunidade que ele teve, a todos. Ele deve a inúmeras pessoas que nunca verão seus filhos na universidade, mas que pagaram para esse garoto ter acesso.

Hoje, o título de doutor é, acima de tudo, um lembrete de que todo o conhecimento adquirido foi conseguido com suor e lágrimas de muitas pessoas e o mínimo que este doutor pode fazer agora é lutar para que outros possam ter acesso aos mesmos privilégios que tive.

Sumário

Lista de Abreviaturas

IP	Internet Protocol
IoT	Internet of Things
SDN	Software-Defined Networking
NV	Network Virtualization
vSDNs	virtual SDN networks
IaaS	Infrastructure as a Service
GNV	Global Network View
API	Application Programming Interface
SSL	Secure Socket Layer
OFP	OpenFlow Protocol
VR	Virtualização de Redes
VLAN	Virtual Local Area Network
VPNs	Virtual Private Network
MAC	Media Access Control
CAPEX	Capital Expenditures
OPEX	Operating Expenditures
CPU	Central Processing Unit
RAM	Random Access Memory
UML	Unified Modeling Language
L2PNV	Layer 2 Prefix-based Network Virtualization
QoS	Quality of Service
CIM-SDN	Common Information Model for Software-Defined Networking
NVP	Network Virtualization Proxy
FPS	Flow Proxy Switch
FPM	Flow Proxy Manager
NVPP	Network Virtualization Proxy Protocol

MVC	Model View Controller
JSON	JavaScript Object Notation
GNC	Graph Network Controller
LGV	Local Graph Virtualization
GGV	Global Graph Virtualization

Lista de Figuras

Lista de Tabelas

Resumo

Resumo da Tese apresentada à banca julgadora como parte dos requisitos necessários para obtenção do título de Doutor no programa de pós-graduação em engenharia elétrica.

Uma Abordagem SDN para Virtualização de Redes

Orientador: Dr. Antônio J. G. Abelém
Palavras-chave: Virtualização; Redes Definidas por Software; Openflow; escalabilidade

As Redes Virtuais Definidas por Software (virtual SDN networks - vSDNs) surgiram da associação de virtualização e Redes Definidas por Software (Software-Defined Networking - SDN), proporcionando maior controle e melhor utilização dos recursos de rede. Vários trabalhos já mostraram a viabilidade e benefícios dessa abordagem. No entanto, o tema ainda carece de soluções que possam virtualizar uma rede de forma escalável, intuitiva e simplificada. Desta forma, esta Tese propõe uma abordagem SDN para a virtualização de redes com o objetivo de reduzir as limitações em vSDNs. Sendo assim, sugere-se adotar na virtualização de rede a separação entre planos de dados (distribuído) e controle (centralizado), visão global da rede e uso de abstração de fluxo para gerir a comunicação entre os diferentes pontos. Para sustentar nossa proposição, três soluções foram desenvolvidas: o CIM-SDN (Common Information Model for Software-Defined Networking), para viabilizar o uso de representação formal dos novos elementos das vSDNs; o NVP (Network Virtualization Proxy), para prover maior escalabilidade através da separação do plano de controle em partes centralizadas e descentralizadas; e o GVL (Graph Virtualization Layer), para fornecer maior uso de abstrações entre o hypervisor e os controladores simplificando, assim, o entendimento e uso da rede. Foram realizadas provas de conceitos para as três soluções propostas, demonstrando a viabilidade da abordagem.

Abstract

Abstract of the thesis presented to the jury as the fulfillment of the requirements to obtain the PhD. in the electrical engineering program.

Title

Advisor: Dr. Antônio J. G. Abelém
Keywords: Virtualization; Software-Defined Networking; Openflow; Scalability.

The virtual SDN networks (vSDNs) have emerged from the association of virtualization and Software-Defined Networking (SDN), providing greater control and better use of network resources. Several studies have already shown the feasibility and benefits of this approach. However, the issue still lacks solutions that can virtualize a network in a scalable, intuitive and simplified manner. Thus, this thesis proposes an SDN approach to network virtualization with the aim of reducing the limitations on vSDNs. Therefore, it is suggested to adopt in network virtualization the separation between data plans (distributed) and control (centralized), the global network view and use of flow abstraction to manage the communication between the different points. To support our proposition, three solutions were developed: the CIM-SDN (Common Information Model for Software-Defined Networking), to enable the use of formal representation of the new elements of vSDNs; The NVP (Network Virtualization Proxy), to provide greater scalability by separating the control plane into centralized and decentralized parts; And Graph Virtualization Layer (GVL), to provide greater use of abstractions between the hypervisor and controllers, thereby simplifying the understanding and use of the network. Proof of concept tests was carried out for the three proposed solutions, demonstrating the feasibility of the approach.

CAPÍTULO 1

Introdução

1.1 Visão Geral

A Internet, originalmente concebida para prover serviço de rede para uma comunidade fechada, é hoje um inegável sucesso mundial, com usuários dos mais diversos tipos de serviços em todos os lugares do planeta.

A tecnologia básica IP (Internet Protocol - IP), cuja flexibilidade e simplicidade são uma das causas do sucesso da Internet, é também a causa das limitações atuais de sua arquitetura, que se tornam cada vez mais evidentes em diversos aspectos, como: a incapacidade de identificar usuários em toda a extensão da Internet, que dificulta o combate à proliferação de mensagens de spam; o esgotamento dos endereços de rede (IPv4), que inibe o desenvolvimento da chamada Internet das coisas (*Internet of Things* - IoT); a falta de suporte à mobilidade, que dificulta a entrega de conteúdo sensível à localização dos usuários; entre outros [6].

Adaptações pontuais têm sido historicamente propostas e implementadas conforme o surgimento de novas demandas. Esta abordagem, apesar de ter atendido a necessidades momentâneas, tem gerado aumento de complexidade e custo de manutenção da Internet. Além disso, quanto maior o número destas adaptações, maior é a complexidade da arquitetura resultante, tornando mais difícil a superação de desafios futuros, uma situação comumente referenciada como o "engessamento" da Internet, cada vez mais resistente a alterações estruturais.

Por conta deste cenário, há um entendimento crescente entre os pesquisadores em redes de computadores que as soluções para a maioria destes problemas dependerão de um "redesenho" da atual arquitetura da Internet, de maneira a solucionar, de forma mais abrangente e de longo prazo, as dificuldades apresentadas pela rede atual. A partir

disto abriu-se espaço para o estudo e proposição de modelos alternativos por meio de um campo de pesquisa denominado Internet do Futuro [6].

Nesse contexto, as Redes Definidas por Software (S*oftware-Defined Networking* - SDN)[7] são hoje uma das soluções mais comentadas. A virtualização de redes (*Network Virtualization* - NV), com importante papel complementar no desenvolvimento de SDN, vem crescendo de maneira a permitir que recursos possam ser compartilhados e que topologias dinâmicas sejam geradas [8]. Atualmente, esta junção é denominada Redes Virtuais Definidas por Software (virtual SDN networks - vSDNs) [9].

A principal ideia para trabalhar com virtualização de rede é a divisão desta em fatias ou redes virtuais (slices), para otimizar a utilização dos seus recursos e para separá-la em diferentes instâncias lógicas, sendo que os recursos compartilhados podem ser nós ou enlaces [10]. Esta abordagem é baseada em infraestrutura como serviço (Infrastructure as a Service - IaaS) [11], [12].

Visando ao uso desse paradigma, várias redes experimentais têm surgido ao redor do mundo [13], [14], [15], com o objetivo de possibilitar a experimentação de protocolos em larga escala. Assim, essas redes devem possuir roteadores e comutadores, espalhados por um país ou pelo mundo, que podem ser programados para executar protocolos experimentais.

O protocolo OpenFlow foi um dos trabalhos recentes que mais causou impacto no âmbito de SDN. Ele teve como objetivo inicial possibilitar a separação do tráfego experimental do tráfego de produção, permitindo que pesquisadores possam testar novos protocolos de rede sem causar impacto na rede de produção [16]. Outro objetivo dessa proposta é possibilitar que os fabricantes possam adicionar as funcionalidades do OpenFlow aos seus comutadores sem necessitarem expor o projeto desses equipamentos. Para auxiliar nesta tarefa, o conceito de controlador de rede foi usado, para concentrar a inteligência e inserir as regras de encaminhamento no comutadores [17].

1.2 Motivação e Desafios

As arquiteturas de redes tradicionais não conseguem suportar completamente os requisitos dos serviços e usuários atuais. Além disso, elas têm se tornado uma arquitetura "engessada", altamente dependente dos fabricantes de equipamentos. Como alternativa para resolver esses problemas e promover o controle de equipamentos de diferentes fabricantes, SDN tem avançando como um novo paradigma para redes de computadores. Suas características de dinamicidade e flexibilidade permitem que as redes estáticas atuais possam se transformar em uma plataforma de serviço capaz de atender rapidamente às necessidades de mudanças de negócios e de usuários [18].

Para habilitar o compartilhamento da rede por diferentes controladores uma camada de abstração chamada hypervisor foi criada [10]. Ela monitora e abstrai os recursos de rede e provê uma rede virtual para as camadas superiores, chamada de slice, assim,

cada controlador pensa que pode controlar o recurso de rede completamente, quando na verdade ele controla apenas a parte do recurso que o hypervisor lhe concedeu.

O hypervisor atua como um tradutor (proxy): ele recebe mensagens dos *switches* em linguagem de rede (OpenFlow) e traduz as informações para um formato de grafo usado internamente para criar os slices; então, ele encaminha as mensagens aos controladores usando novamente linguagem de rede (OpenFlow), estes irão traduzir a mensagem novamente para uma linguagem de grafo usada para manipular a lógica da rede [3].

Todas essas traduções, além do desperdício de processamento, inibem o uso da abstração de grafo entre o virtualizador e o controlador, dificultando a aplicação direta da teoria de grafos no planejamento da rede para melhorar seu desempenho e facilitar o gerenciamento, o controle e o uso de uma visão global da rede.

1.2.1 Desafios Científicos

Comumente, o estudo de redes está focado em protocolos e tecnologia de comunicação. Estes são divididos em camadas, definidas tanto pelo modelo OSI como pelo TCP/IP [19]. Tais modelos organizam uma série de protocolos delimitando seus escopos e dentro do estudo dos protocolos ocorre um estudo de algumas abstrações, como a abstração de fluxo provida pelo TCP, a atuação autônoma e distribuída do CSMA-CA e a própria topologia da rede através de grafos [20].

A manipulação de fluxo, a separação de planos de dados e controle e a visão global da rede são as principais contribuições científicas que o paradigma de SDN traz para o universo de redes [21]. Até o advento destas, poucas abstrações foram realmente estudadas em redes, diferentemente de outras disciplinas, onde o foco são as abstrações e não as tecnologias, evidenciando um problema científico no universo de redes, a pouca exploração de abstrações.

Pontos positivos e negativos sobre sistemas distribuídos e centralizados são discutidos na literatura há muito tempo [22], sendo que o universo SDN apresenta um caso particular de mistura dos dois paradigmas, apresentando o plano de dados distribuído e o plano de controle centralizado. No entanto, a virtualização acaba por tornar o plano de controle centralizado um problema para a escalabilidade, uma vez que todo o tráfego de controle, incluindo eventuais pacotes vindos do plano de dados, passará pelo plano de controle e, por conseguinte, pelo hypervisor da rede.

A visão global da rede utilizada em SDN traz um problema de ordem científico referente à escalabilidade da solução, uma vez que um hypervisor teria que concentrar toda a topologia da rede e depois distribuir o seu controle entre os diferentes slices. Tal problema já começa a receber atenção da academia, existindo uma boa base para a discussão do problema de escalabilidade no universo de virtualização em SDN.

Ainda que SDN apresente uma visão global da rede e o controle separado do plano de dados com a possibilidade de manipular fluxos, nossa maneira de pensar os problemas

de rede ainda considera uma rede de pacotes, distribuída e com camadas hierárquicas. Problemas como spanning tree, que são originários de um ambiente onde não existem fluxos, ainda são tratados em SDN. Desta forma, recriamos problemas que não deveriam existir nesse paradigma, trazendo à tona outro problema científico, a dificuldade de adotar completamente um paradigma como SDN e não apenas uma adoção da tecnologia [23].

A virtualização, ainda que não seja um conceito de redes, é fortemente utilizada para prover compartilhamento de recurso e abstração [24]. Normalmente, junto destas abstrações, novas funcionalidades são adicionadas, uma vez que ao fornecer novas abstrações é possível visualizar problemas de uma forma distinta e consequentemente novas funcionalidades aparecem.

Quando avaliamos especificamente o uso de virtualização no universo de rede ele serve basicamente para compartilhar recurso. As abstrações que utilizamos são poucas, uma vez que a complexidade das camadas de baixo são repassadas para as de cima, ou seja, o hypervisor não abstrai a linguagem de rede utilizada e expõe para as camadas de cima exatamente os mesmo tipos de elementos existentes nas camadas de baixo, apenas adicionando algumas funcionalidades, como enlaces virtuais.

A junção dos dois problemas, inicialmente relatados (pouco estudo de abstrações e dificuldade de adotar um paradigma e não apenas tecnologias), acaba resultando em um terceiro problema que é a necessidade de gerenciar complexidade. Este problema é evidenciado ao perceber que os hypervisors não abstraem as complexidades através do uso de novas abstrações, fazendo apenas o repasse de complexidade para os controladores SDN.

Nesse contexto, podemos destacar os seguintes desafios científicos:

- Pouca exploração do conceito de abstrações em redes;
- A dificuldade de adotar efetivamente um paradigma e não apenas as tecnologias associadas a ele;
- Como prover escalabilidade usando um hypervisor que concentra toda a topologia da rede e depois distribui o seu controle entre os diferentes slices;
- A dificuldade dos hypervisors em ocultar a complexidade existente no contexto para as camadas superiores.

1.2.2 Desafios Tecnológicos

Uma vez que SDN apresenta novos elementos de rede, não mais restritos a *hardware*, estes precisam de uma representação, porém, sem a perda de informação e escopo. Neste contexto, o slice é um elemento da rede, uma abstração passa a ser tão importante quanto um *switch*, sendo que a primeira está no plano de controle, enquanto o segundo no plano de dados [25]. Tais elementos precisam ser representados nos diagramas de

rede, porém, o uso de diagramas de redes com figuras de equipamentos e linhas representando os enlaces não mais é suficiente para representá-los em conjunto com aplicações, controladores e hypervisors. Este problema de ordem tecnológica, a dificuldade de representar os novos elementos SDN, ainda pode ser acrescido a dificuldade de manter uma documentação atualizada da rede, uma vez que as mudanças da rede e a atualização das documentações acontece em momentos distintos.

É possível condensar os desafios tecnológicos levantados em:

- Como representar os novos elementos de vSDNs;
- Como manter uma documentação atualizada da rede.

1.3 Hipótese central e Questões de Pesquisa

Dadas as limitações nas abordagens atuais de vSDNs, o principal objetivo desta tese é responder a seguinte questão:

Questão Principal: **Como oferecer virtualização em SDN de forma escalável, intuitiva e simplificada?**

Para responder a questão levantada esta tese apresenta a seguinte hipótese:

Hipótese: **As limitações em vSDNs podem ser reduzidas através de uma abordagem SDN para a virtualização de redes.**

Na nossa hipótese assumimos abordagem SDN como aquela que apresenta a separação entre planos de dados (distribuído) e controle (centralizado), visão global da rede e uso de abstração de fluxo para gerir a comunicação entre os diferentes pontos da rede.

Para guiar a investigação desta tese, questões de pesquisa adicionais relacionadas com a hipótese são definidas e apresentadas a seguir.

Questão de pesquisa I: Como representar elementos virtualizados e os novos elementos SDN?

Questão de pesquisa II: Como implementar soluções escaláveis em SDN?

Questão de pesquisa III: Como simplificar o entendimento de virtualização de forma a minimizar as questões técnicas?

1.4 Contribuições

Entre as principais contribuições cientificas e técnicas desta tese, podemos destacar: a criação de um Formalismo para Modelagem de vSDNs, onde foram criadas classes para representar os novos elementos SDN assim como o elementos virtualizados; a criação de um Framework escalável de virtualização para SDN que permita que grandes redes

sejam virtualizadas, provendo uma melhor utilização dos recursos e permitindo a gerência desta seguindo os preceitos de SDN; e a criação de uma camada de virtualização que provê maior abstração para a rede e permita o uso do paradigma SDN e não apenas sua tecnologia.

Além destas contribuições gerais, outras contribuições específicas foram geradas:

- Realizar um survey na área de modelagem de redes;
- Realizar um survey dos hypervisor SDN;
- Definir os requisitos de virtualização para uma SDN;
- Definir procedimento para validar uma rede antes de sua implementação;
- Implementar um hypervisor como proxy voltado à escalabilidade;
- Implementar um hypervisor usando grafos para se comunicar com os controladores.

1.5 Organização do Texto

O restante do documento está dividido em 5 capítulos seguindo o ordenamento descrito abaixo:

- Capitulo 2: Apresenta um estudo sobre todos os importantes aspectos relacionados à Virtualização, mais especificamente sobre virtualização no ambiente OpenFlow. Além disso, os conceitos e arquitetura de Redes Definidas por *Software* são apresentados.
- Capítulo 3: Apresenta as propostas de modelagem e virtualização de rede. Para todas as soluções apresentadas serão informadas as principais vantagens e desvantagens, incluindo os problemas relacionados à modelagem, escalabilidade e abstração de redes;
- Capítulo 4: Apresenta os detalhes das soluções de modelagem e virtualização propostas nesta tese.
- Capítulo 5: Descreve as implementações, os possíveis usos, os experimentos realizados e os resultados obtidos.
- Capítulo 6: Descreve as considerações finais deste trabalho, as contribuições e levanta as questões que podem ser discutidas futuramente.

CAPÍTULO 2

Referencial Teórico

Neste capítulo é apresentada uma visão geral sobre SDN, um dos principais paradigmas usados para o que conhecemos hoje como Internet do Futuro. Ainda em estágio de maturação, SDN apresentam novos desafios comparados ao paradigma tradicional de redes. No restante do capítulo serão apresentadas as possiveis soluções de virtualização de redes em especial a solução de virtualização em SDN.

2.1 Redes Definidas por Software

SDN está chamando muita atenção da indústria como uma solução não apenas acadêmica, mas realmente aplicável a redes de produção. Todos os elementos da indústria de redes, incluindo fornecedores de equipamentos, provedores de serviços de Internet, provedores de serviços na nuvem e usuários, estão trabalhando e/ou atentos aos avanços de SDN. Desta forma, o desenvolvimento de soluções usando SDN não é apenas uma questão acadêmica, mas uma questão de mercado [26].

2.1.1 Visão Geral

O desenvolvimento das Redes Definidas por Software tem gerado um grande interesse em repensar as abordagens clássicas de arquitetura e projetos de rede. Com SDN é possível separar o plano lógico do hardware de encaminhamento, e mover o controle lógico e de estado para um componente de software programável.

Uma das principais características habilitadas através da separação entre o plano de dados e de controle, é a habilidade de projetar e pensar o plano de controle como uma aplicação controladora centralizada operando com uma visão global da rede (*Global*

Network View - GNV). Basicamente, SDN proporciona aos projetistas de rede a liberdade para refazer o plano de controle da rede, permitindo que este atue como uma aplicação centralizada em vez de sistema distribuído pensado de maneira centralizada [27].

2.1.2 Arquitetura

Para a implementação de uma SDN é necessário que alguns novos elementos estejam presentes na rede. A Figura 1 apresenta o componentes de hardware e a Figura 2 os componentes de software.

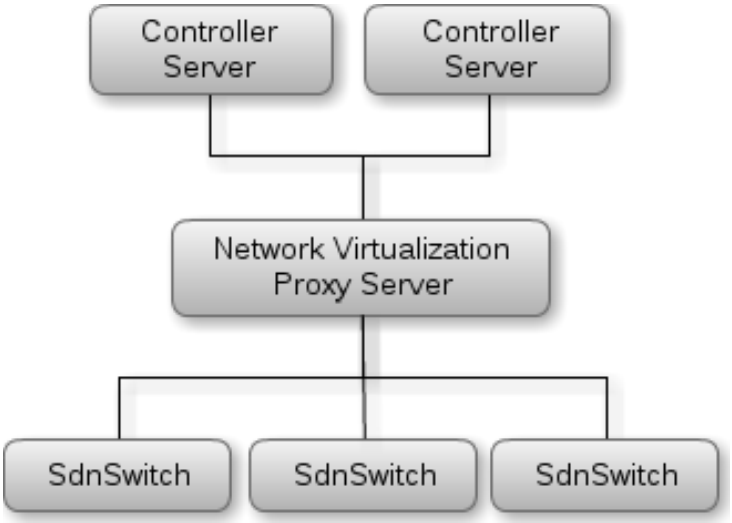


Figura 1: Arquitetura de Hardware SDN [1].

Os *Controller Servers* e *Virtualization Proxy Servers* são servidores responsáveis por rodar os programas que irão prover o controle e a virtualização da rede respectivamente. É importante que estes equipamentos sejam vistos de maneira diferentes pelas ferramentas de gerenciamento pois eles são a base para a criação da rede. Se qualquer problema ocorrer nesses equipamentos, toda a rede será comprometida.

Os SDN *Switches* são equipamentos com suporte à comutação por fluxo. Diferentes dos *switches* convencionais eles não possuem o mesmo nível de inteligência, então seu gerenciamento precisa ser pensado levando em conta o controle que esta sendo executado fora de cada um desses equipamentos.

É importante ressaltar que na arquitetura tradicional a rede não era tão dependente de um serviço fornecido por um servidor de rede como ela é com SDN.

Os elementos de software apresentados na Figura 2 são responsáveis por todo o comportamento da rede. Eles são descritos a seguir:

- **API de Fluxos:** Os comutadores irão suportar uma API única, que irá trabalhar com fluxos e terá seu controle em um elemento externo. O OpenFlow é a API de fluxo mais conhecida atualmente. No entanto, nada impede que outras APIs possam

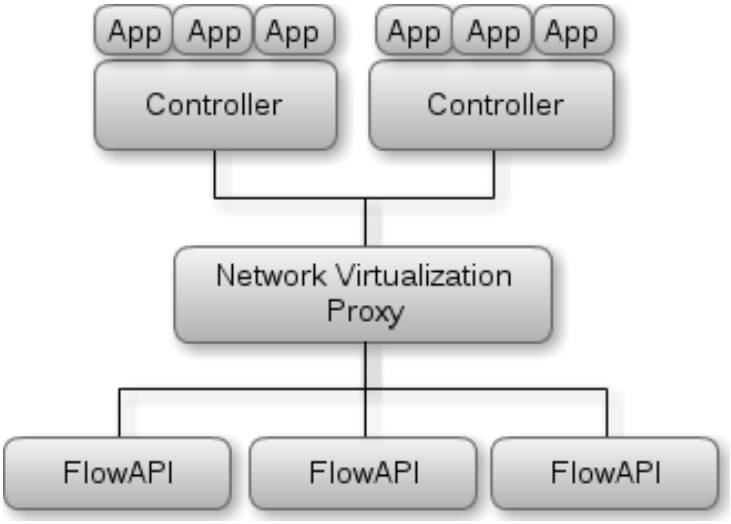


Figura 2: Arquitetura de Software SDN [1].

ser utilizadas pela rede desde que os demais componentes de rede tenham suporte a elas;

- **Camada de Virtualização:** Este elemento é responsável por integrar todos os comutadores da rede e possibilitar a virtualização de rede através da distribuição dos recursos por diferentes controladores de rede. Atualmente a única implementação conhecida é o FlowVisor, desta forma uma melhor descrição sobre ele será realizada posteriormente. Para que outras implementações SDN possam ser utilizadas, é necessário que estas possuam suporte a Flow API usada na rede.

- **Controlador:** Ele provê uma interface de programação de alto nível. que pode ser usado para implementar as aplicações que irão definir o gerenciamento da rede. A inteligência que foi tirada dos comutadores está centralizada no Controllers. Existem várias implementações de Controllers como NOX, Becon e POX;

- **Apps:** São aplicações que serão executadas sobre os Controlles e que irão definir o comportamento da rede.

As SDN ainda estão em foco atualmente, e muitos trabalhos vem sendo desenvolvidos. Desta forma, ainda não existe um padrão bem definido para nomear os seus componentes. Adotamos esta por acreditarmos representar bem cada um dos elementos envolvidos.

2.1.3 OpenFlow

O arcabouço (*framework*) OpenFlow [16] é uma das possíveis implementações de SDN que vem oferecendo aos pesquisadores a possibilidade de testar seus protocolos experimentais em redes de produção como: redes de ensino e pesquisa e em redes metropolitanas. Ele, além de oferecer o protocolo de controle, chamado de OpenFlow protocol

para manipular a tabela de encaminhamento dos roteadores e *switches*, também oferece uma (*Application Programming Interface* - API) simples e extensível para programar o comportamento dos fluxos de pacotes. É desta forma que, através desta API, pesquisadores podem rapidamente construir novos protocolos e aplicá-los em um ambiente de produção sem interferir nos demais fluxos.

No OpenFlow é proposto um mecanismo que é executado em todos os comutadores ou roteadores, de forma que possa haver isolamento entre os tráfegos. Assim, o OpenFlow possibilita que os pesquisadores reprogramem os comutadores sem provocar interferência nas configurações da rede de produção. Além de permitir que os fabricantes possam incluir as funcionalidades do OpenFlow nos seus comutadores sem necessitarem expor o projeto desses equipamentos.

Outro requisito desejado é que os equipamentos OpenFlow devam possuir um custo baixo e desempenho semelhante aos já utilizados nas redes de produção, de forma que os administradores destas redes aceitem a substituição dos equipamentos já existentes. Desta forma, podemos sumarizar os seguintes requisitos do OpenFlow: possibilidade de uso em implementação de baixo custo e de alto desempenho; capacidade de suportar uma ampla gama de pesquisas científicas; garantia de isolamento entre o tráfego experimental e o tráfego de produção; consistência com a necessidade dos fabricantes não exporem o projeto de suas plataformas.

O OpenFlow explora a tabela de fluxo que já existe nos comutadores atuais, e normalmente é utilizada para implementar serviços como NAT, firewall e VLANs. Sendo assim, o comutador OpenFlow é constituído de uma tabela de fluxos e um evento associado a cada entrada na tabela. Sua arquitetura pode ser decompostas em três partes básicas:

- **Tabela de Fluxos:** Cada entrada na tabela de fluxos contém uma ação associada, e consiste em Campos do cabeçalho (utilizado para definir um fluxo), ações (define como os pacotes devem ser processados e para onde devem ser encaminhados) e contadores (utilizados para estatísticas ou remoção de fluxos inativos);
- **Canal Seguro:** Para que a rede não sofra ataques, o *Secure Channel* assegura 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). Dependendo do controlador, ele pode suporta outras interfaces (passivas ou ativas), TCP e PCAP. Essas são bem úteis em ambientes virtuais, pela simplicidade de utilização, pois não necessitam de chaves criptográficas;
- **Protocolo OpenFlow:** É um protocolo binário 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 OpenFlow (*OpenFlow Protocol* - OFP) permite que o comutador se torne programável.

2.2 Virtualização

A virtualização é uma técnica que permite que um sistema execute processos oferecendo a cada um deles a ilusão de executar sobre recursos dedicados. Ela representa uma maneira de isolar as diferentes linhas de execução permitindo a utilização eficiente da crescente capacidade computacional disponível, além de integrar arquiteturas onde elementos comuns a um conjunto de processos virtualizados possuem apenas uma cópia em execução acessada de forma compartilhada [28].

2.2.1 Virtualização de Rede

A virtualização de computadores já é usada há muito tempo e hoje está amplamente disponível em várias plataformas. Essencialmente ela é realizada por meio do compartilhamento de processadores e dispositivos de E/S, utilizando técnicas de fatiamento de tempo e memória virtual. Assim como a virtualização provê o compartilhamento de recursos de um nó computacional por múltiplos sistemas, a Virtualização de Redes (VR) é um método para que múltiplas arquiteturas de rede heterogêneas compartilhem o mesmo substrato físico, neste caso, componentes de uma rede como roteadores, comutadores, multiplexação de enlaces, etc [24].

Existem três abordagens bem conhecidas para a implementação de VR: as Redes Locais Virtuais (Virtual Local Area Network - VLAN), as Redes Privadas Virtuais (Virtual Private Network - VPNs) e as redes de sobreposição (Overlay) [24]. Adicionalmente, uma quarta abordagem vem ganhando força a partir do conceito de redes programáveis [29].

2.2.1.1 Redes Locais Virtuais (VLANs)

Uma VLAN 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.

Comumente, em uma rede local, a comunicação entre as diferentes máquinas é governada pela arquitetura física. Graças às redes virtuais (VLANs), é possível livrar-se das limitações da arquitetura física (constrangimentos geográficos, restrições de endereçamento, etc), definindo uma segmentação lógica (software), baseada num agrupamento de máquinas com critérios como endereços (Media Access Control - MAC), números de porta ou protocolo.

Foram definidos vários tipos de VLAN, de acordo com o nível em que se efetua a virtualização:

- **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:** Distinguem-se vários tipos de VLAN de nível 3: VLAN por sub-rede (Network Address-Based VLAN), que associa sub-rede de acordo com o endereço IP fonte dos datagramas e VLAN por protocolo (em inglês 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 acima da rede física, e a esse respeito oferece mais flexibilidade para a administração e modificações da rede, porque qualquer arquitetura pode ser alterada por simples parametrização dos comutadores. Adicionalmente, existe um ganho em segurança, já que as informações são encapsuladas em um nível suplementar e eventualmente analisadas [30].

2.2.2 Redes Privadas Virtuais (VPN's)

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. As VPNs são túneis seguros entre pontos autorizados com proteção criptográfica, entre redes corporativas ou usuários remotos. A segurança é a primeira e mais importante função da VPN. Uma vez que dados privados serão transmitidos pela Internet, eles devem ser protegidos de forma a não permitir que sejam modificados ou interceptados.

Uma das grandes vantagens decorrentes do uso das VPNs é a redução de custos com comunicações corporativas, pois elimina a necessidade de enlaces dedicados de longa distância que podem ser substituídos pela Internet. Sendo assim, as LANs podem, através de enlaces dedicados ou discados, conectarem-se a algum provedor de acesso local e interligarem-se às outras LANs, possibilitando o fluxo de dados através da Internet. Esta solução pode ser bastante interessante sob o ponto de vista econômico, sobretudo nos casos em que enlaces internacionais ou nacionais de longa distância estão envolvidos.

2.2.3 Redes de Sobreposição (Overlay)

Uma rede de sobreposição é uma rede virtual que cria uma topologia virtual sobre a topologia física da rede. Os nós em uma rede de sobreposição são unidos por meios de ligações virtuais, que correspondem a caminhos na rede subjacente. Na Figura 3 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 existam várias 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. Assim, as redes sobrepostas têm sido usadas para implantar novos recursos e extensões na Internet.

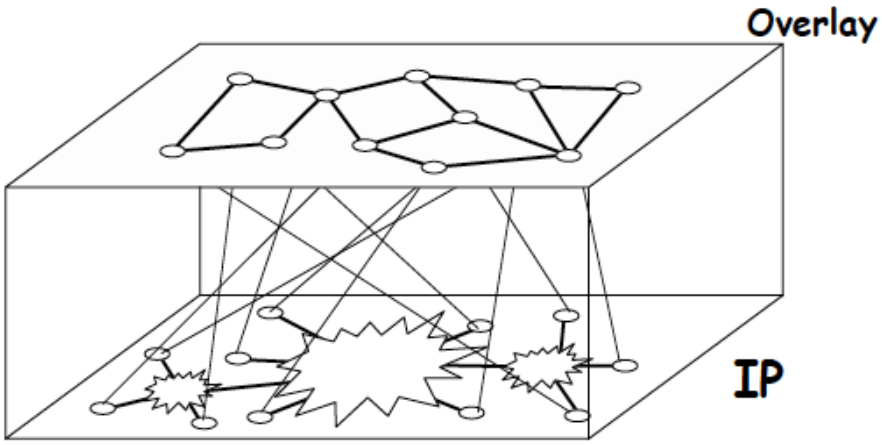


Figura 3: Modelo de Rede de Sobreposição.

Muitos modelos de sobreposição têm sido propostos nos últimos anos para resolver problemas que incluem: garantias de qualidade de serviço [31], ataque de negação de serviços [32], distribuição de conteúdo [33], compartilhamento de arquivos [34] e até sistemas de armazenamento [35]. Redes de sobreposição também estão sendo usada como ambientes de teste, por exemplo PlanetLab [36], Federica [37] e o GENI [38], para desenvolvimento e avaliação de novas arquiteturas.

2.2.3.1 Virtualização de Rede com OpenFlow

Assim como a 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. Portanto, a rede em si deve ter uma camada de abstração de hardware, similar ao que acontece na virtualização de computadores.

Esta camada deve ser facilmente "fatiável", para que múltiplas redes independentes possam ser executadas simultaneamente em cima, sem interferir entre si, sobre uma variedade de *hardwares* diferentes, incluindo comutadores, roteadores, pontos de acesso, etc.

De acordo com Sherwood [29], a camada de abstração de hardware é provida pelo OpenFlow e como camada de virtualização se tem FlowVisor. Similar à virtualização de computadores, o FlowVisor é uma camada de abstração que reside entre o controlador e comutador de fluxos, conforme ilustrado na Figura 4. Portanto, a integração FlowVisor

e OpenFlow permite que em uma rede OpenFlow possam ser criadas várias fatias de recursos de redes, rodando simultaneamente e isoladamente.

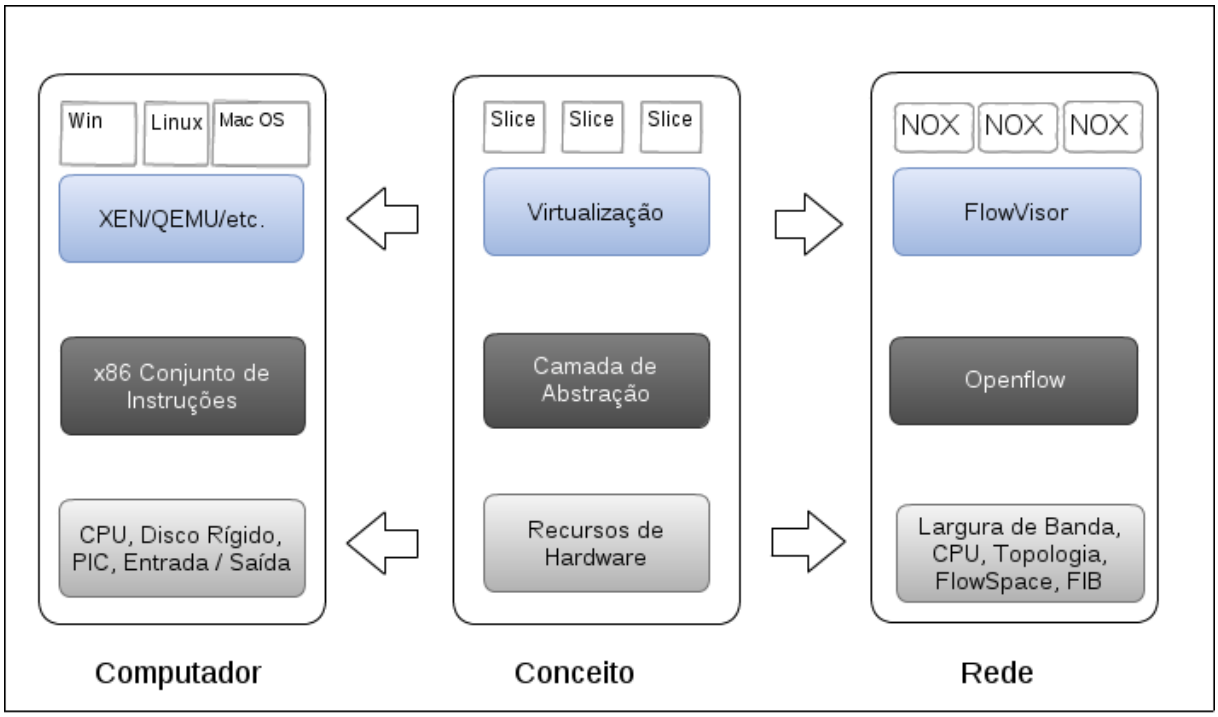


Figura 4: Comparação entre o FlowVisor como camada de virtualização com a virtualização computacional [2].

O FlowVisor é um controlador especial que atua como um proxy entre os controladores e os comutadores da rede. Todas as mensagens OpenFlow, tanto dos controladores para os comutadores como no sentido oposto, são encaminhadas para o FlowVisor que decidirá o que fazer com as mensagens baseado nas políticas de cada fatia. 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 [29].

Cada fatia está vinculada a um controlador. O FlowVisor define uma fatia como um conjunto de fluxos também chamado de "espaço de fluxo"(ou *flowspace*). Devido a versão 1.0 do protocolo OpenFlow permitir a definição de um fluxo como uma combinação de dez campos do cabeçalho de pacote, incluindo informações das camadas física, de enlace, de rede e de transporte, o FlowVisor possibilita que se implemente fatias com um elevado nível de granularidade, no que diz respeito à caracterização do *flowspace* [39]. Além disso, as fatias podem ser definidas por ações de negação, união e intersecção.

Um exemplo de operação do FlowVisor está mostrado na Figura 5. Nela a rede é dividida entre três controladores. Dois deles são para os experimentos dos pesquisadores Alice e Bob e o outro controla o tráfego de produção. As políticas da fatia da rede dada para o Bob são feitas de tal forma que o seu controlador só possa manipular e perceber os fluxos provenientes de um determinado IP de origem. Devido à transparência do FlowVisor, porém, o controlador do Bob acredita que pode controlar os fluxos para todo tráfego vindo de qualquer IP de origem. Assim, a exemplo da Figura 1, quando o

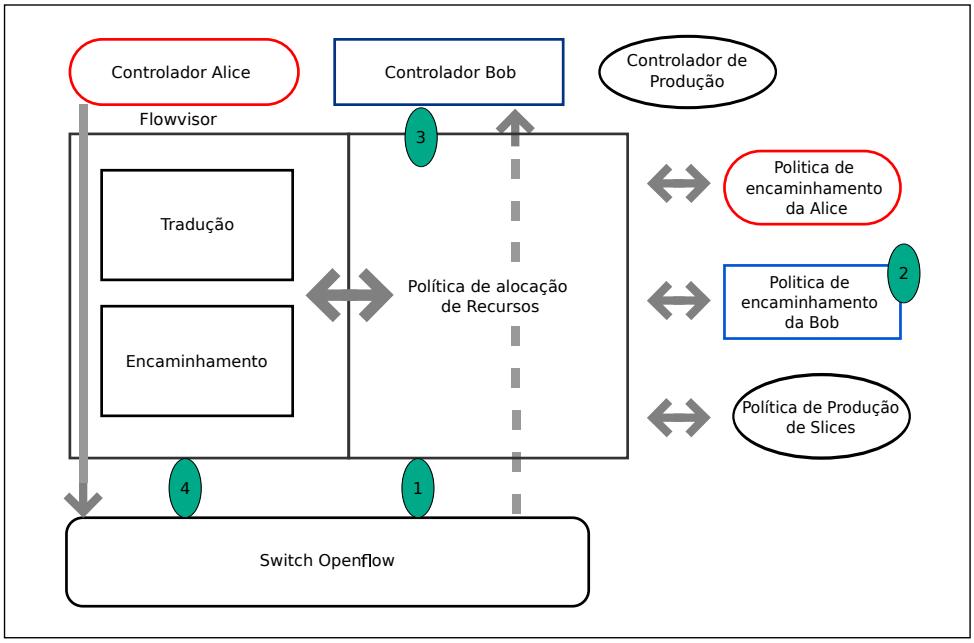


Figura 5: Funcionamento do FlowVisor.

controlador do Bob envia uma mensagem para adicionar uma entrada na Tabela de Fluxos de um comutador, o FlowVisor recebe essa mensagem (número 1), consulta as políticas da fatia do Bob (número 2), e reescreve essa entrada para incluir apenas o tráfego oriundo do IP de origem permitido para Bob. Após, envia uma mensagem com essa entrada para o comutador (número 3). Da mesma forma, as mensagens enviadas dos comutadores para o controlador do Bob são interceptadas pelo FlowVisor (número 4) .

O particionamento da rede possibilita que as ações desenvolvidas, em uma de suas fatias, não interfiram negativamente nos demais, mesmo que estes estejam compartilhando a mesma infraestrutura física. Em arquiteturas mais tradicionais, a rede é fatiada através da técnica de VLAN, porém, com a diversidade dos modelos de redes, a estrutura de VLANs torna os experimentos, como o IP *Mobility* ou *Wireless Handover*, por exemplo, bastante difíceis de gerenciar.

As características inerentes ao FlowVisor, como a virtualização transparente, o forte isolamento entre as fatias e a sua rica política de definição de flowspaces, tornam o mesmo uma ferramenta extremamente eficiente no que diz respeito à virtualização e implementação de redes programáveis orientadas a software.

2.2.4 NFV

A grande demanda por equipamentos de rede, que são cada vez mais complexos, torna os custos para criação e manutenção de data-centers, cada vez mais onerosos, como os *Storage*, *Firewall* e Servidores de Virtualização, que exigem hardwares cada vez mais específicos, provocando custos elevados. A utilização de equipamentos detalhados para uma função, praticamente inviabiliza a reciclagem, dificultando uma possível reutilização

em outra atividade. Fatores como consumo energético e computação verde se mostram prejudicados pelo grande consumo de energia que estas soluções costumam exigir.

Considerando este cenário cada vez mais especializado e custoso, o NFV foi desenvolvido como uma alternativa de diminuir custos financeiros (*Capital Expenditures* - CAPEX) e Despesas Operacionais (*Operating Expenditures* – OPEX). Enquanto o NFV tem a intenção de reduzir custos, o SDN permite a maior agilidade no desenvolvimento de aplicações e inovação, criando um ciclo sustentável [40].

Ainda segundo Hawilo [40], os principais componentes das plataformas de virtualização baseadas em NFV são: Servidor Físico, Máquina Virtual e Hypervisor. O Servidor é o equipamento que contém os recursos físicos necessários para o funcionamento da estrutura, componentes como a Unidade Central de Processamento (*Central Processing Unit* - CPU), a Memória de Acesso Aleatório (Random Access Memory - RAM), e a Unidade de Armazenamento.

Hypervisor é um sistema operacional que costuma funcionar em data-centers, e de forma a economizar recursos físicos, energéticos e operacionais, permitem que diferentes sistemas lógicos funcionem através da mesma plataforma física (Servidor Físico). Por fim, temos a Máquina Virtual, que é o sistema lógico gerenciado pelo Hypervisor, onde diferentes serviços são instalados e configurados de maneira independente, e não interferem entre si, mesmo que funcionem no mesmo Servidor Físico.

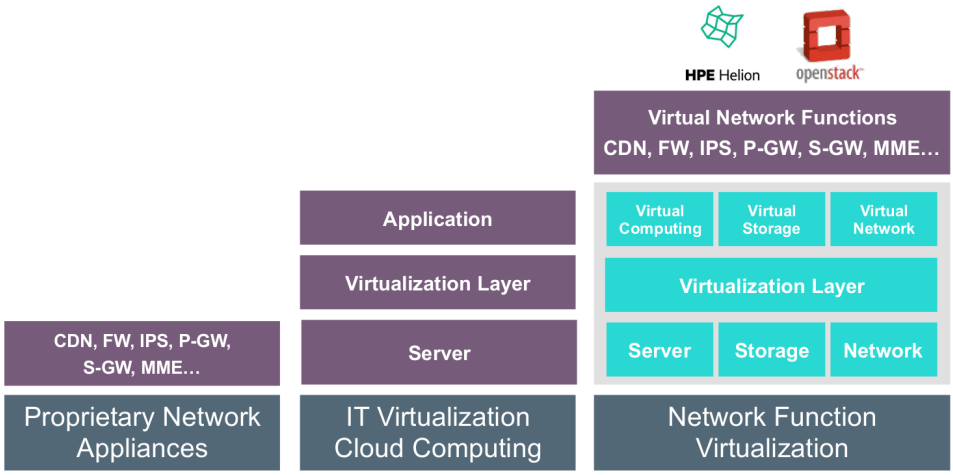


Figura 6: Comparação entre as diferentes soluções de Rede.

A Figura 6 permite a comparação entre as diferentes arquiteturas de rede. Quanto mais o serviço oferecido se aproxima da camada de hardware, mais especializado o equipamento deve ser para executar a tarefa desejada. Portanto, a coluna mais à direita apresenta uma maior camada de softwares, com o objetivo de organizar melhor a estrutura da rede e permitir que componentes menos especializados sejam capazes de executar as tarefas da rede.

Ainda na Figura 6 é possível identificar o conjunto de soluções de dispositivos físicos (appliances), no lado esquerdo, cada caixa é um hardware que precisa ser instalado em um local, geralmente um hack, onde precisa ser alocado e apresenta um consumo

energético.

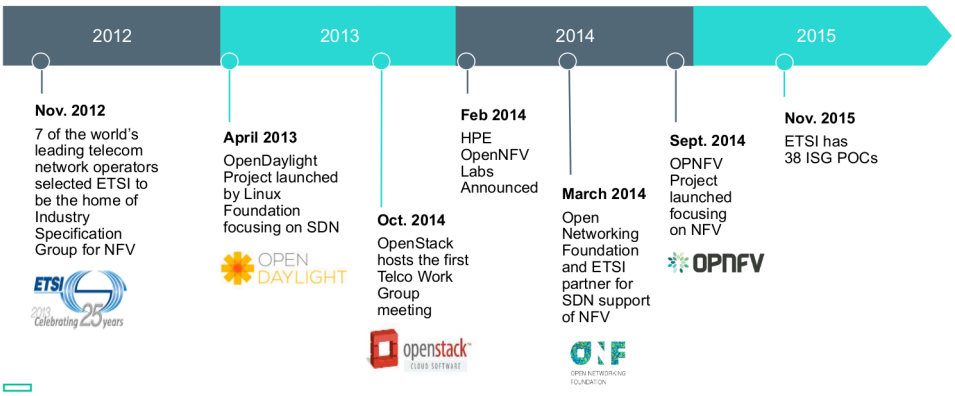


Figura 7: Comparação entre as diferentes soluções de Rede.

A evolução da aceitação da arquitetura NFV é mostrada na Figura 7. Em 2013, foi criado o projeto OpenDaylight, um controlador SDN que integra soluções em NFV, o objetivo é permitir a aceleração da implantação de SDN [41]. O Software OpenStack é uma ferramenta capaz de gerenciar e integrar diferentes soluções de serviços, máquinas virtuais, armazenamento e balanceadores de carga, a integração desta solução é conhecida como nuvem. Uma solução que integra NFV e OpenStack utiliza as capacidades de NFV na criação de rede, integrada à robustez da solução OpenStack.

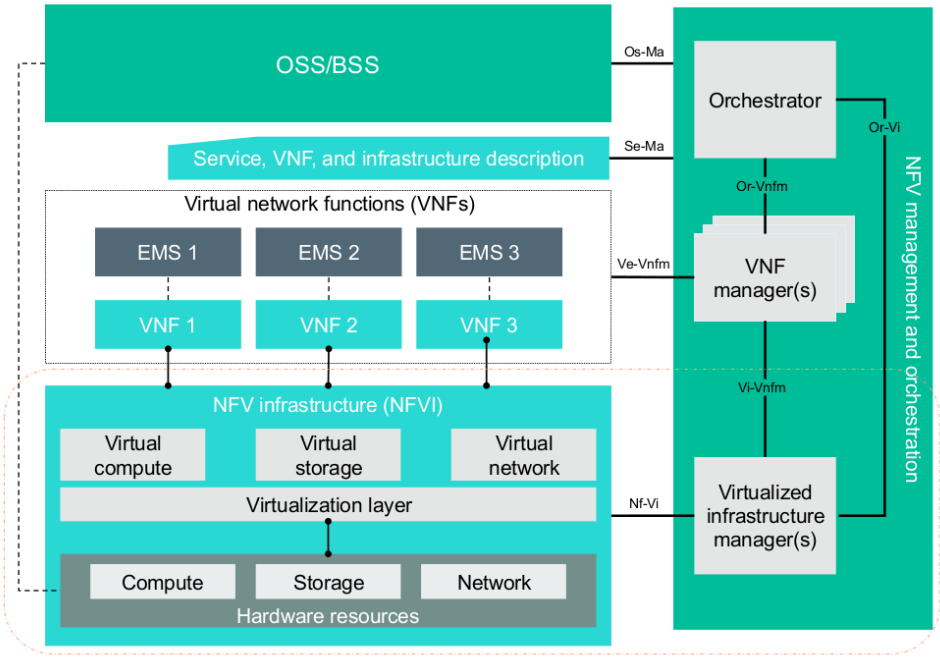


Figura 8: Arquitetura do Framework NFC da ETSI.

A Figura 8 apresenta a Arquitetura do Framework NFV da ETSI. Ele expõe a Camada de Virtualização (Virtualization Layer), e as ferramentas que trabalham próximas, em especial a Virtualização de Rede (Virtual Network). Este é o local de atuação do

hypervisor de rede, o qual é um dos focos desta Tese. Destacamos esta atuação para posicionar claramente onde o hypervisor e as demais contribuições desta tese são realizadas dentro do universo de NFV.

2.3 Conclusão do Capítulo

Este capítulo teve como objetivo apresentar uma revisão sobre SDN, mostrando sua arquitetura e a principal implementação disponível atualmente. Uma revisão sobre virtualização de rede foi apresentada, dando ênfase à solução de virtualização utilizada em SDN. Suas características e relevâncias foram descritas para fornecer o claro entendimento da proposta desse trabalho, exibidas no capítulo 4. Por consequência, este capítulo tem fundamental importância, pois o detalhamento de todas as tecnologias e características, presentes no trabalho, é relevante para o tornar claro o entendimento sobre o funcionamento da proposta.

CAPÍTULO 3

Trabalhos Relacionados

Este capítulo tem como objetivo apresentar os trabalhos relacionados às vSDNs. Tais trabalhos estão organizados de forma a auxilia na resposta de cada uma das questões de pesquisa apresentadas no Capítulo 1. Desta forma, cada uma das perguntas apresenta sua seção de trabalhos relacionados.

3.1 Como representar elementos virtualizados e os novos elementos SDN?

Esta seção apresenta alguns dos trabalhos chaves para a modelagem/representação de redes usando UML (*Unified Modeling Language* - UML e os modelos já existentes para a troca de informações entre ambientes heterogêneos.

3.1.1 Modelagem de Redes com UML

Devido à aceitação da UML pela Object Modeling Group (OMG) [1] como a linguagem oficial para modelagem de objetos[42], apresentamos os principais trabalhos relacionados à modelagem UML dentro do contexto de redes.

No trabalho de Saxena [43] é apresentada uma modelagem que utiliza UML em sistemas distribuídos. Porém, o trabalho não trata de entidades de rede, como *hosts* e comutadores, modelando apenas processos distribuídos.

O trabalho de Lei [44] mostra um conjunto de ligações entre diferentes nós em uma rede de sensores expressa em UML. Porém, os autores fornecem um uso superficial

[1] www.omg.org

da linguagem, limitando seu escopo.

Zhao apresenta outro artigo propondo uso de UML para modelar rede de sensores [45]. A abordagem é utilizada para a simplificação de sistemas complexos utilizando ZigBee (um conjunto de especificações para a comunicação sem-fio entre dispositivos eletrônicos voltado para a implantação de soluções de baixo custo, baixa potência de operação e baixa faixa de transmissão). O artigo limita-se apenas a modelar redes ZigBee.

3.1.2 Modelos de informação para ambientes Heterogêneos

O Modelo de Informação Comum (*Common Information Model* – CIM) é um modelo aberto que define como elementos gerenciados podem ser representados juntamente com suas interligações. Inicialmente utilizado na modelagem de transmissão de energia, o CIM tem se expandido, com extensões para modelagem distribuída, mercados elétricos, gestão de faturamento de clientes, troca de dados geográficos e também na modelagem de redes de computadores[46].

Com o desenvolvimento e a aceitação cada vez maior do CIM para modelagens e troca de informações entre diferentes equipamentos, diversas extensões do CIM vêm sendo produzidas para solucionar problemas específicos de determinadas redes. Uma extensão para atender a modelagens de ambientes virtuais de redes é proposta por Fuertes [44], porém o artigo aborda apenas virtualização de máquinas e não virtualização de redes.

Outra extensão para o CIM foi proposta por Challa [47]. O objetivo do trabalho deste autor é voltado para computação em nuvem. Neste caso, o armazenamento em nuvem provoca uma abstração da complexidade da rede. Os autores utilizam o CIM para simplificar as técnicas de gerência por fornecer apenas uma camada de abstração de dados que pode ser usada por todos os elementos presentes na rede. No entanto, apesar do bom trabalho desenvolvido, o mesmo não trata questões específicas de SDN.

3.1.3 Conclusões da Seção

Os trabalhos relacionados apresentados nesta seção evidenciam que a modelagem de rede é um tópico importante e que o uso de UML para modelar redes já é utilizado em outras propostas. No entanto, não encontramos trabalhos que possam absorver as particularidades dos novos elementos presentes no universo das vSDNs.

3.2 Como implementar soluções de virtualização escaláveis em SDN?

Os trabalhos apresentados nesta seção fornecem elementos para auxilar na resposta da pergunta apresentada ao fornecer os principais trabalhos no universo de virtualização em SDN que, direta ou indiretamente, abordam a questão de escalabilidade.

3.2.1 FlowVisor

A principal proposta de virtualização de redes OpenFlow é o FlowVisor [2]. Ele é proposto como um controlador especial que atua como um proxy transparente, localizado entre o comutador OpenFlow e o controlador usado na rede. Com o uso do FlowVisor é possível que um switch possa ser subdividido e tenha seus recursos controlados por mais de um controlador ao mesmo tempo, permitindo, desta forma, a criação de redes virtuais com isolamento lógico entre si.

O FlowVisor adota uma arquitetura hierárquica permitindo que vários FlowVisors presentes na rede possam ser interligados tornando possível a virtualização de um recurso já virtualizado. Como ele funciona de forma transparente, sua atuação para os comutadores se dá como um controlador, e para os controladores ele atua como um comutador OpenFlow. Sendo assim, ele tem que armazenar as informações dos comutadores e dos controladores para poder fazer o encaminhamento das mensagens entre eles.

No entanto, sua arquitetura gera um *overhead*, pois a mesma entidade responsável pelas informações de gerência da rede virtualizada é a responsável por encaminhar as mensagens de controle geradas, incluindo os packet_in que podem consumir muitos recursos ao serem reencaminhadas entre os FlowVisors até chegarem aos comutadores/controladores de destino.

3.2.2 Layer 2 Prefix-based Network Virtualization (L2PNV)

O L2PNV é uma [48] extensão do FlowVisor, com o objetivo principal de prover um mecanismo de virtualização de nível 2 sem usar VLAN, apenas baseando as redes virtuais criadas nos endereços MAC de origem e destino. Para isso foi necessário modificar o FlowVisor, firmware do switch para suportar uma versão modificada do OpenFlow e o próprio host, uma vez que foi preciso habilitar o mascaramento por MAC nesses elementos.

Uma série de questões sobre *broadcast* e *multicast* são levantadas e discutidas no artigo, uma vez que o uso de MAC é o elemento chave nessa proposta. Vale ressaltar que o FlowVisor oferece virtualização usando atributos das camadas 1 a 4, enquanto que a proposta só trabalha em camada 2.

Apesar de ser dito no artigo do L2PNV [48] que um ambiente de teste foi criado usando a solução proposta e as devidas modificações nos elementos de rede para atender as demandas da proposta, não foram apresentados resultados referentes à escalabilidade da solução ou comparativo com as demais soluções existentes. De fato, não foi apresentado nenhum tipo de avaliação.

3.2.3 ADvanced FlowVisor (ADVisor)

Visando superar umas das principais limitações do FlowVisor, o fato das topologias virtuais que podem ser criadas com o FlowVisor serem restritas a um subconjunto da topologia física, o ADVisor [49] foi proposto. A solução aproveita a tag de VLAN para diferenciar entre enlaces virtuais e rede virtuais, tornando possível a criação de enlaces virtuais através da junção de diferentes enlaces físicos.

Dentre as alterações realizadas, estão definidos mecanismos para o isolamento entre as topologias virtuais. Porém, a solução validada pelos autores ainda inclui a necessidade de configuração manual dos dispositivos de rede e o envio de comandos dpctl, não sendo expostas alterações ao protocolo OpenFlow que permitam as configurações dos datapaths pelo FlowVisor, como a alocação de filas, definição de escalonadores e estabelecimento de parâmetros como largura mínima e máxima de banda.

A solução limita o uso do cabeçalho de VLAN, deixando a solução menos transparente para o usuário, além disso, os testes mostraram aumento da latência quando comparado ao FlowVisor. Novamente testes de carga com uma rede de grande porte que mostre a escalabilidade da solução não foram realizados.

3.2.4 ViRtual TopologIes Generalization in OpenFlow networks (VeRTIGO)

Ainda para solucionar o problema de enlaces virtuais o VeRTIGO [50] foi proposto. Ele é uma extensão do FlowVisor, que basicamente coloca uma camada de inteligência que pode criar topologias virtuais sobre a topologia física, nesse caso, criando enlaces virtuais interligando enlaces físicos contínuos. Além disso, ele pode prover uma estrutura virtualizada completa, uma vez que é possível virtualizar os nós da rede juntamente com os enlaces.

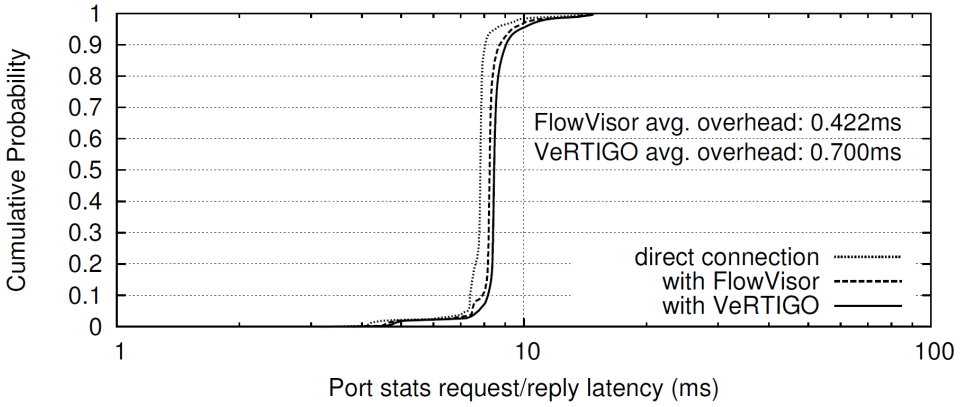


Figura 9: Probabilidade Acumulativa de Latência.

A proposta foi implantada no testbed OFELIA, onde foi possível comprovar que a solução consegue de fato prover enlaces virtuais. No entanto, é possível verificar na

Figura 9, retirado do próprio artigo, que a solução mostra um aumento de 65,88% no overhead quando comparado com o FlowVisor, fato este que torna a escalabilidade com a proposta questionável. Maiores testes relacionados à escalabilidade também não foram realizados.

3.2.5 FlowVisorQoS

A proposta FlowVisorQoS [51], resolve o problema da falta de garantia do uso de banda por cada slice existente no FlowVisor. A proposta consegue garantir a correta separação entre os fluxos ao acrescentar ao FlowVisor o arcabouço QoSFlow [52]. Sendo assim, é possível definir requisitos mínimos de Qualidade de Serviço (Quality of Service - QoS) que serão respeitados nos comutadores, garantindo que cada fluxo não ultrapasse os limites especificados.

Para a implementação da solução foram realizadas mudanças no firmware dos comutadores, aplicando neles o QoSFlow. Além disso, foi criado a uma extensão ao FlowVisor para permitir que os parâmetros de QoS possam ser adicionados à criação dos slices. Porém, apesar da solução alcançar os objetivos, não foram realizados testes que mostrem o comportamento da solução em um ambiente em larga escala.

3.2.6 OpenVirtex

O OpenVirtex [53] foi criado usando a mesma abordagem do FlowVisor e funciona como um proxy, atuando de forma transparente entre o switch e o controlador. No entanto, diferentemente do FlowVisor, o OpenVirtex provê enlaces virtuais e isolamento entre os slices.

Apesar de a solução apresentar melhorias de desempenho quando comparado com o FlowVisor, ela continua com os mesmo requisitos de *hardware*, 4 núcleos de processadores e 4GB de memória [54], dificultando a escalabilidade da rede.

3.2.7 Flow-N

O Flow-N [55] é, no nosso conhecimento, o primeiro trabalho de virtualização em redes OpenFlow que aborda o problema de escalabilidade. Ele foi proposto como uma extensão do NOX, e permite a virtualização da rede baseada em contêineres e usando bancos de dados relacionais para o mapeamento entre as topologias virtuais e as reais.

Os testes de escalabilidade apresentados no trabalho, que podem ser vistos na Figura 10, mostram que ele tem um comportamento quase que estável de latência até o numero de 100 redes virtuais criadas, enquanto que o FlowVisor começa com um valor baixo, mas vai crescendo até quase igualar o valor do FlowN. Infelizmente, apenas um gráfico é apresentado e a proposta não pode ser mais bem avaliada, uma vez que maiores detalhes sobre o funcionamento da solução não são fornecidos.

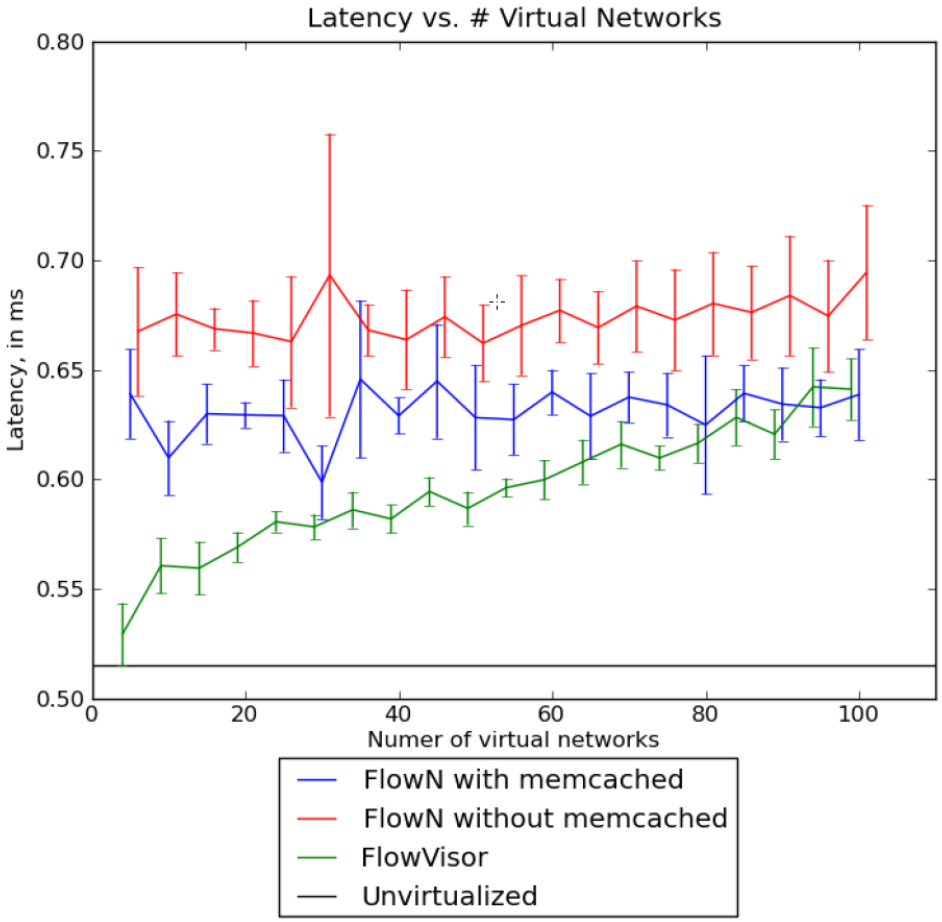


Figura 10: Latência X Número de Redes Virtuais.

3.2.8 HyperFlex

O HyperFlex [56] é uma proposta de arquitetura de hypervisor que utiliza a separação do plano de virtualização para adotar maior escalabilidade, ela funciona em 4 modos de operação variando os tipos de elementos presentes na arquitetura adotada. Na prova de conceito o FlowVisor, em conjunto com o controlador Ryu[2] foi utilizado para implementar a solução.

Apesar da excelente ideia de separar o plano de virtualização, a proposta não apresenta comparativos com outras propostas de virtualização ou testes de carga, além de oferecer uma complexidade grande de implementação, uma vez que utiliza vários tipos de elementos. A prova de conceito apresentada utiliza o FlowVisor como parte da solução, desta forma, herdando seus problemas de escalabilidade.

3.2.9 Conclusão da Seção

Dentre os requisitos estudados e discutidos nesta seção, identificamos que a solução que provê virtualização para redes definidas por software deve fornecer, isola-

[2]https://osrg.github.io/ryu/

mento, enlaces virtuais e estar preparada para atender a um grande número de equipamentos, pois todos os demais serviços estarão em execução utilizando o substrato virtualizado como base.

Tabela 1: Comparação das Soluções de Virtualização.

Trabalhos	Escalabilidade	Enlaces Virtuais	Transparente	Isolamento
FlowVisor [2]	Não	Não	Sim	Parcial
L2PNV [48]	Não	Não	Sim	Parcial
ADVisor [49]	Não	Não	Não	Parcial
VeRTIGO [50]	Não	Sim	Sim	Parcial
FlowVisorQoS [51]	Não	Não	Parcial	Sim
OpenVirtex [53]	Não	Sim	Sim	Sim
Flow-N [55]	Parcial	Sim	Sim	Parcial
HyperFlex [56]	Parcial	Sim	Sim	Sim

A Tabela 1 sumariza as principais características das propostas de virtualização apresentadas nesse capítulo. É possível concluir que nenhum dos hypervisors pode atender plenamente a todos os requisitos citados anteriormente, para fornecer uma comunicação confiável e escalável, que atenda a todas as necessidades de uma SDN no que tange à virtualização.

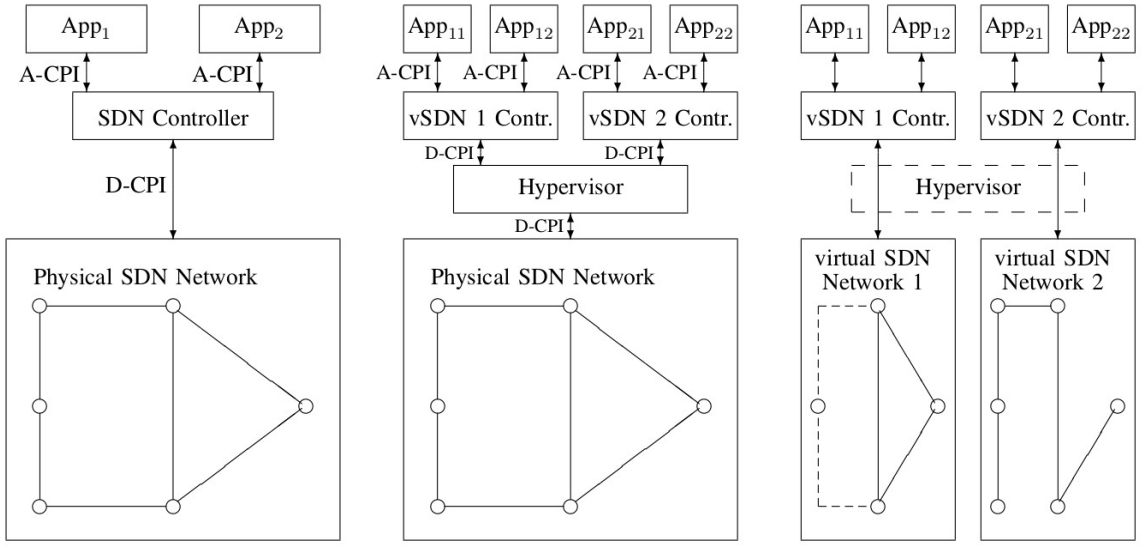
Por fim, é importante ressaltar, que apesar da clara importância de escalabilidade para as soluções de virtualização no ambiente SDN, apenas dois trabalhos [55] e [56] tentam abordar esse tema, mesmo que não forneçam as informações necessárias para uma conclusão mais satisfatória, como: comparativos de números de switches suportados, número de controladores, número de redes virtuais, etc.

3.3 Como simplificar o entendimento de virtualização de forma a minimizar as questões técnicas?

Esta seção apresenta os trabalhos relacionados ao uso de abstrações na comunicação entre switches e controladores por parte do hypervisor, mostrando como estas abstrações são usadas em vários pontos da comunicação, sofrendo traduções e expondo complexidade às camadas superiores.

O hypervisor é descrito por Andreas et al. [3] como um proxy que interage através de múltiplas interfaces do plano de controle (Data-Controller Plane Interface - D-CPI) se comunicando com vários controladores das vSDNs, como apresentado na Figura 11.

Uma SDN, sem virtualização, é ilustrada na parte (a) da Figura 11, onde as aplicações interagem com o controlador SDN através de interfaces das aplicações do plano de controle(application-controller plane interface (A-CPI)), que por sua vez interage com a rede física através das interfaces de controle do plano de dados(data-controller plane



(a) Conventional SDN Network: An SDN controller directly interacts with the physical SDN network to provide services to the applications.

(b) Virtualizing the SDN Network (Perspective of the hypervisor): The inserted hypervisor directly interacts with the physical SDN Networks as well as two virtual SDN controllers.

(c) Virtual SDN Networks (Perspective of the virtual SDN controllers): Each virtual SDN controller has the perception of transparently interacting with its virtual SDN network.

Figura 11: Redes definidas por software, com e sem virtualização [3].

interface (D-CPI)). A virtualização é ilustrada nas partes (b) e (c). Um hypervisor é inserido entre a rede física e o controlador como mostrado na parte (b). O hypervisor provê, para cada controlador, a percepção de que ele esta trabalhando diretamente com sua rede virtual, como ilustrado na parte (c). Para fazer isso, o hypervisor tem que traduzir as mensagens OpenFlow(enviadas pelo controlador) para a maneira como ele organiza a rede física e os slices, a fim de determinar o destino final das mensagens. Então, outra tradução é realizada para o OpenFlow, a fim de enviar as mensagens à parte física da rede, assim mantendo o padrão D-CPI usado.

Aplicações de rede, usualmente, precisam ter uma visão topológica global da rede, para gerenciar, controlar ou prover outros serviços. Este é um dos principais benefícios do paradigma SDN [57], motivando a pesquisa de diferentes formas de representar os elementos de rede. A forma mais comum é representada por grafos [58] e é usada em aplicações [59] e, internamente, por hypervisors de rede [2, 60, 53, 56]. No entanto, o hypervisor não tira vantagem desta representação para comunicação com o controlador, demandando sucessivas traduções como mencionado anteriormente.

O grafo de encaminhamento de funções virtuais de rede (Virtual network function forwarding graph (VNF-FG)) [61] é um caso de uso fornecido pela ETSi NFV ISG para permitir que o virtual appliances sejam encadeados juntos de uma maneira flexível. Ele usa abstração de grafo para criar o caminho de encaminhamento de rede, usando “links” e “nodes”. Além disso, o trabalho diz que switches SDN controlados (OpenFlow) podem criar diretamente grafos de encaminhamento em diferentes e dinâmicas formas.

3.3.1 Conclusão da Seção

De acordo com os trabalhos apresentados nesta Seção, todos os hypervisors estudados usam um modelo de proxy, ou seja, repassam a complexidade da camada inferior para a camada superior, restringindo o uso de grafos à parte interna de cada entidade de software, mesmo com o reconhecimento por parte da academia [59] e da indústria [61] que a abstração de grafo deveria ser mais utilizada.

3.4 Conclusão do Capítulo

Este capítulo teve como ênfase apresentar as propostas de modelagem e virtualização para redes definidas por software que podem ser utilizados como base para formular as respostas às 3 questões de pesquisa levantadas no Capítulo 1.

Através da análise dos trabalhos apresentados é possível identificar a necessidade de formalismo para a modelagem/representação dos elementos das vSDNs, a carência de soluções para escalabilidade dos hypervisors, dada sua natural centralização do plano de controle, além do pouco uso de abstrações na comunicação entre hypervisors e controladores.

CAPÍTULO 4

Um Abordagem SDN para virtualização em Redes

Conforme apresentado nos Capítulos 2 e 3, os requisitos fundamentais para o vSDNs são visualização/modelagem dos elementos da rede, enlaces virtuais (para possibilitar uma maior variedade de topologias), escalabilidade (maior número de equipamentos no substrato físico) e abstração poderosa para permitir uma mudança real no paradigma e, por conseguinte, na maneira como pensamos redes virtualizadas. Tais características são esperadas para que esta tecnologia possa realmente ser utilizada como base para as redes de produção.

Para responder à questão principal apresentada nesta tese, foram derivadas três questões que, conjuntamente, procuram identificar os requisitos essenciais para tal. Desta forma, 3 propostas foram desenvolvidas como resposta a cada uma das 3 questões derivadas da hipótese apresentada no Capítulo 1. Tais propostas são incrementais, ou seja, o produto de cada uma delas é utilizado com a posterior de forma a prover uma única resposta no final.

4.1 CIM-SDN

Para responder à questão de pesquisa I: **Como apresentar elementos virtualizados e os novos elementos SDN?** o Common Information Model for Software-Defined Networking (CIM-SDN) [1] foi proposto. Ele é uma extensão do CIM para suportar os elementos SDN e permitir que este tipo de rede seja modelada formalmente, usando como base o Diagrama de Classes, que especifica as relações entre cada tipo de elemento da rede e o Diagrama de Objetos, que é utilizado para modelar uma instância específica da rede, utilizando aspectos relacionais e estruturais com o objetivo de tornar o entendimento mais

intuitivo.

4.1.1 Design do CIM-SDN

Uma vez que o CIM não foi feito especificamente para SDN, as classes presentes nele não atendem totalmente às especificidades das vSDNs. Desta forma, faz-se necessária a definição de novos elementos através da especialização das classes já existentes.

A Figura 12 apresenta o meta modelo (classes) com os referidos relacionamentos. A formalização deste diagrama define um conjunto de classes que dividem os mesmos atributos, operações, relacionamentos e semânticas. No conjunto de relacionamentos, a ligação de classes ocorre umas com as outras, de modo em que, no contexto de redes de computadores, podem ser lógicas, ou seja, um relacionamento que interliga serviços, ou física, a exemplo de uma conexão cabeada ligada em uma rede *ethernet*.

O CIM_ComputerSystem é a classe mãe CIM usada para definir a classe Servers através de especialização. Já o CIM_VirtualComputerSystem é a classe derivada da CIM_ComputerSystem que representa a habilidade de virtualizar ou emular outro sistema computacional, no entanto essa classe está sendo descontinuada em favor do uso de CIM_ComputerSystem e da associação HostedDependency[62]. Sendo assim, todos os servidores virtualizados serão pelo uso do CIM_ComputerSystem e da associação HostedDependency. Os switches serão derivados da CIM_VirtualEthernetSwitch que é uma instância do CIM_ComputerSystem associado a um conjunto de instâncias de portas Ethernet e seus componentes virtualizados associados[63].

Em seguida são descritos cada uma das classes definidas pelo CIM-SDN e seus respectivos valores semânticos.

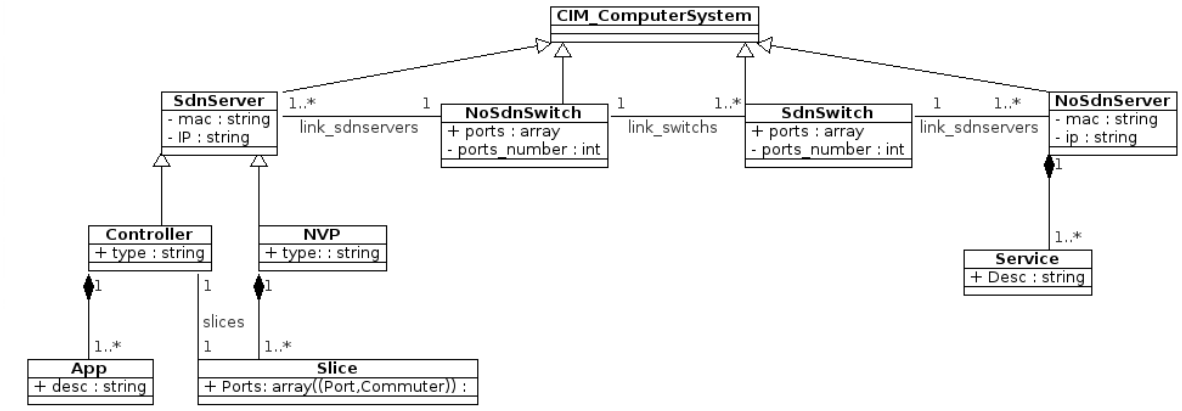


Figura 12: Diagrama de Classes [1].

- **SDNServer**: Esta classe representa os servidores SDN. Eles são usados para criar a topologia da rede e não tem equivalente em redes tradicionais.
  - ***NVP Network Virtualization Proxy***: Esta classe representa o servidor NVP, que agrega diferentes recursos de rede os disponibiliza através de slices

para os controladores. O atributo "tipo"identifica qual das diferentes implementações de NVP que podem estar na rede.

  - **Controller**: É a entidade usada para controlar os recursos disponibilizados pelo NVP através dos slices. Um controlador pode ter várias aplicações associadas e assim como o NVP ele possui um atributo que identifica os diferentes tipos de controladores que podem estar em uso na rede.

- **SdnSwitch**: São os comutadores SDN que fazem os transportes de fluxos, porém, sem tarefas de decisão. Todas as atividades de decisão são feitas pelo controlador que gerencia cada conjunto de *switches*.

- **NoSdnSwitch**: É um switch convencional usado para viabilizar a comunicação entre os diferentes elementos SDN na rede. Ele apresenta o atributo ports seguindo o mesmo conceito do SdnSwitch, ou seja, uma lista de tuplas, tendo como número da porta o índice e o nome do equipamento conectado nela como valor.

- **NoSdnServer** É um servidor de rede que provê serviços como DHCP (Dynamic Host Configuration Protocol) e DNS (Domain Name System).

- **Service**: São os diferentes serviços que podem estar presentes na rede. Cada servidor pode ter um ou mais serviços, como (*Hypertext Transfer Protocol*) - HTTP, *Simple Network Management Protocol* (SNMP) e *Simple Mail Transfer Protocol* (SMTP).

- **Slice**: Estes são os conjuntos de recursos disponibilizados pelo NVP para o controlador. O atributo "ports" é um array com o número da porta e o nome do SdnSwitch. Sua associação com o NVP é uma composição pois só pode existir um slice se existir um NVP associado a ele. Além disso, existe uma associação simples com o controlador para especificar qual controlador irá gerenciar o slice.

- **APP**: São as diferentes aplicações que podem ser executadas no controlador (Controller) da rede. Cada aplicação está associada a um controlador através de uma composição de dados, cada aplicação está obrigatoriamente associada a um controlador.

Usando o metamodelo da Figura 12 é possível modelar vSDNs. Uma rede específica pode ser representada através da instanciação deste diagrama de classe em um diagrama de objetos. Assim, cada instanciação do metamodelo será uma representação de uma rede distinta.

4.1.2 Conclusão da Seção

Esta seção apresentou o CIM-SDN, uma extensão ao CIM que permite a modelagem de vSDNs usando UML. A proposta traz maior clareza e concisão nos diagramas gerados, além de permitir que estes diagramas sejam usados ativamente da gerência da

rede uma vez que são documentos formais que poder ser utilizados para gerar configurações para as ferramentas de gerência.

4.2 NVP

Para responder a questão de pesquisa II: **Como implementar soluções escaláveis em SDN?** desenvolvemos uma pesquisa sobre os requisitos básicos necessários para a escalabilidade e projetamos o (*Network Virtualization Proxy* - (NVP)[4] para validar nossa abordagem de separação do plano de controle, trazendo escalabilidade às vSDNs.

Uma das principais características do FlowVisor é permitir sua utilização hierarquicamente. Esta característica traz enorme flexibilidade quanto às diferentes configurações possíveis. Também permite que uma rede de grande escala possa estar integrada pela mesma ferramenta de virtualização. No entanto, dada sua arquitetura, esta característica impõem uma importante limitação quanto à escalabilidade da rede, uma vez que este modelo introduz mais elementos entre o controlador e o comutador, conforme o nível hierárquico vai aumentando.

Desta forma, nós propusemos o NVP, como um novo modelo de virtualização, que prega pela simplicidade e escalabilidade da camada de virtualização de rede. Sendo assim, os principais objetivos do NVP são:

- Permitir que os recursos dos comutadores OpenFlow possam ser compartilhados entre diferentes controladores;
- Prover uma real escalabilidade para a rede, fazendo com que a camada de virtualização possa receber um número muito elevado de comutadores e controladores, sem gerar um *overhead* que inviabilize a utilização da rede;
- Prover uma abstração global da rede, na qual todos os recursos disponíveis não estejam separados hierarquicamente, ou seja, o aumento do numero de elementos de virtualização não incrementam o numero de elementos entre os comutadores e os controladores;
- Permitir que redes de diferentes domínios possam ser integradas facilmente;
- Melhorar o uso de recursos computacionais no processo de virtualização de rede;

4.2.1 Arquitetura do NVP

O NVP é composto de dois elementos, o Flow Proxy Switch - (FPS) e o Flow Proxy Manager - (FPM). Sua concepção foi inspirada no funcionamento do próprio OpenFlow, no qual a complexidade de controle é retirada dos comutadores e passada a uma

entidade de controle externa. Neste caso, o FPS atuando de maneira "burra"e o FPM concentrando todas as informações e inteligência do sistema de virtualização. Sua arquitetura é apresentada na Figura 13.

A separação de funções dentro do hypervisor em diferentes elementos de redes foi adotada, também, no HyperFlex[56], com variações nos números de elementos e de funções. A ideia de separar as funções providas pelos hypervisor em diferentes elementos de rede, que corresponde à parte central da proposta de escalabilidade, já era compartilhada por outros pesquisadores ao redor do mundo, o que corrobora para validar a solução como uma boa alternativa para oferecer escalabilidade ao hypervisor[1].

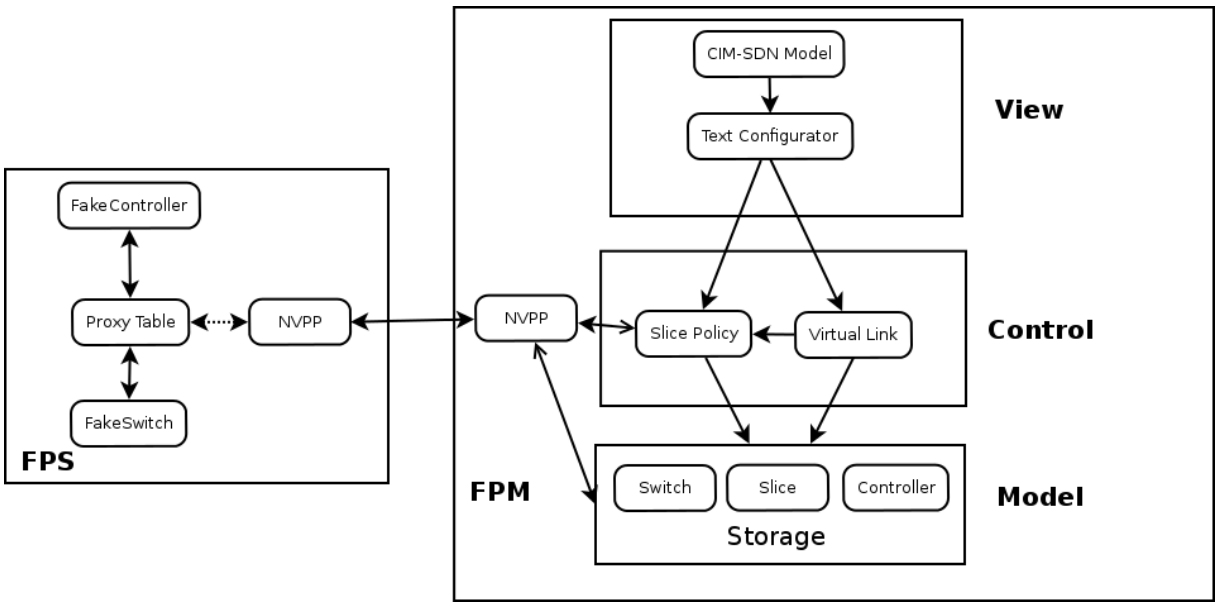


Figura 13: Arquitetura do NVP [4].

4.2.1.1 Flow Proxy Switch (FPS)

É a entidade responsável por encaminhar as informações entre os switches e os controladores. Ela é a entidade que irá trocar mensagens OpenFlow com os switches e controladores e repassá-las para o Flow Proxy Manager (FPM). Após a inicialização da rede, a única funcionalidade do FPS é consultar sua tabela de encaminhamento e decidir para qual controlador/switch as mensagens devem ser encaminhadas. Ele é formado pelos componentes: FakeController, FakeSwitch, ProxyTable e o Network Virtualization Proxy Protocol (NVPP).

- **FakeController:** É uma entidade que atua como um controlador OpenFlow, passando a idéia para o comutador OpenFlow que ele está conectado ao controlador

[1]O desenvolvimento do NVP como proposta começou no início de 2014, sendo que a primeira vez que o NVP foi tornado público em um documento, aconteceu no final de 2014, no processo de qualificação de doutorado. Durante o processo de publicação do NVP em periódicos e seus respectivos tempos de espera, o HyperFlex foi publicado em 2015.

da rede. Ele é o responsável por garantir a transparência, do lado dos comutadores, e de enviar as informações obtidas sobre os comutadores para o FPM, usando o NVPP.

- **FakeSwitch:** Assim como o FakeController, ele é o responsável por garantir a transparência, só que pelo lado dos controladores, e prover para estes as informações obtidas com o FPM, através do NVPP, sobre os recursos que cada controlador vai acessar. Ele uma entidade que atua como um controlador de rede, passando a idéia para o controlador OpenFlow que ele está conectado a um switch OpenFlow.

- **Network Virtualization Proxy Protocol (NVPP):** É um protocolo binário, similar ao OpenFlow, desenvolvido para viabilizar a comunicação entre os FPSs da rede e o FPM. Devido às necessidades de manter o FPS o mais simples possível, optamos por desenvolver um protocolo binário que possa atender às necessidades de comunicação entre as entidades, mas que fosse de fácil implementação.

- **Proxy Table:** É uma tabela que armazena quais recursos de um determinado comutador podem ser utilizados por um determinado controlador. Seus campos são apresentados na Tabela 2. Os campos ctr_addr e ctr_port identificam de forma única o controlador de rede. O campo datapath_id identifica a qual comutador este fluxo está relacionado e os campo in_port e dl_vlan dizem a porta e a possível vlan associado àquela rede virtual.

Tabela 2: Exemplo de entradas na Proxy Table.

ctr_addr	ctr_port	datapath_id	in_port	dl_vlan
192.168.1.2	6633	2	1	2
192.168.1.1	6633	2	1	1

No exemplo presente na Tabela 2, uma ação do tipo flow_mod vinda de um controlador com IP 192.168.1.2 e em execução na porta 6633 poderá ser encaminhada única e exclusivamente para o comutador com o datapath_id 2, na porta 1 e na vlan 2. Caso um dos campos difira, o pacote será descartado.

4.2.1.2 Flow Proxy Manager (FPM)

É o elemento que realiza o controle efetivo da virtualização da rede. Ele contém uma base de dados com informações sobre os switches, controladores e slices da rede. Todas as mudanças de slices devem passar por ele, que deve inserir as regras no FPS. Um FPM será usado em cada domínio, podendo haver vários FPSs associados a ele, permitindo que a rede possa ser escalável, uma vez que a real carga gerada na rede é através dos packet_in, e esses serão manipulados apenas pelo PFS ao realizar o encaminhamento. Sua arquitetura segue o padrão Model View Controller (MVC) e o mesmo é descrito a seguir.

- **Storage** É o componente responsável por armazenar as informações dos switches que foram capturadas pelo FPS, dos controladores que foram fornecidas através do arquivo de configuração e dos slices, também fornecidos pelo arquivo de configuração. Os dados armazenados aqui serão usados pelo Slice Policy para alimentar a ProxyTable do FPS.

- **Slice Policy** Responde pela parte da criação dos slices, permitindo a criação de políticas e associando os recursos aos controladores requisitados pelo administrador. Ele deve usar as informações dos switches já presentes no Storage e seguindo as políticas presentes, criar os slices na FlowTable.

- **Virtual Link** Esse elemento é responsável por mapear os enlaces físicos e permitir a criação de enlaces virtuais através da criação de regras de encaminhamento nos comutadores. Desta forma, é possível interligar enlaces físicos contínuos e oferecer a abstração de um enlace virtual, conectando diretamente duas portas dos comutadores.

- **Text Configurator** É a interface de configuração do NVP. Ela é similar à configuração usada pelo FlowVisor, para facilitar a adoção do NVP por outros serviços. Ele consiste de um arquivo usando a notação de objetos JavaScript (JavaScript Object Notation - JSON) que é apresentado no Anexo B.

- **CIM-SDN Model** Este é um arquivo XMI que modela toda a rede a ser virtualizada e as próprias redes virtuais. Ele utiliza UML para representar os elementos da rede, sendo, desta forma, além de documentação um elemento ativo no processo de gerência da rede.

  A motivação para a criação do CIM-SDN [1] surgiu no momento que iniciamos os estudos sobre virtualização e percebemos que não havia uma solução formal para modelagem de redes. Dessa forma, desenvolvemos uma extensão com CIM que pudesse representar os elementos de SDN (incluindo os slices) e criamos um script para traduzir o modelo proposto nos arquivos de configuração usado pelo NVP.

  Com o uso desta solução, é possível modelar a rede seguindo um método formal, garantindo a consistência da rede, além de manter uma documentação atualizada e de fácil entendimento sobre toda a rede.

4.2.2 Aplicação do NVP

O NVP está localizado entre os comutadores e os controladores. É possível observar na Figura 14 como os componentes do NVP se relacionam com os demais componentes da rede. Um FPS pode atender vários comutadores e controladores, no entanto, para permitir a escalabilidade da rede, o número de FPS pode ser incrementado para atender mais comutadores e comutadores. Já o FPM é único na rede, sendo responsável por prover a visão global da rede e facilitar a integração desta com outros domínios.

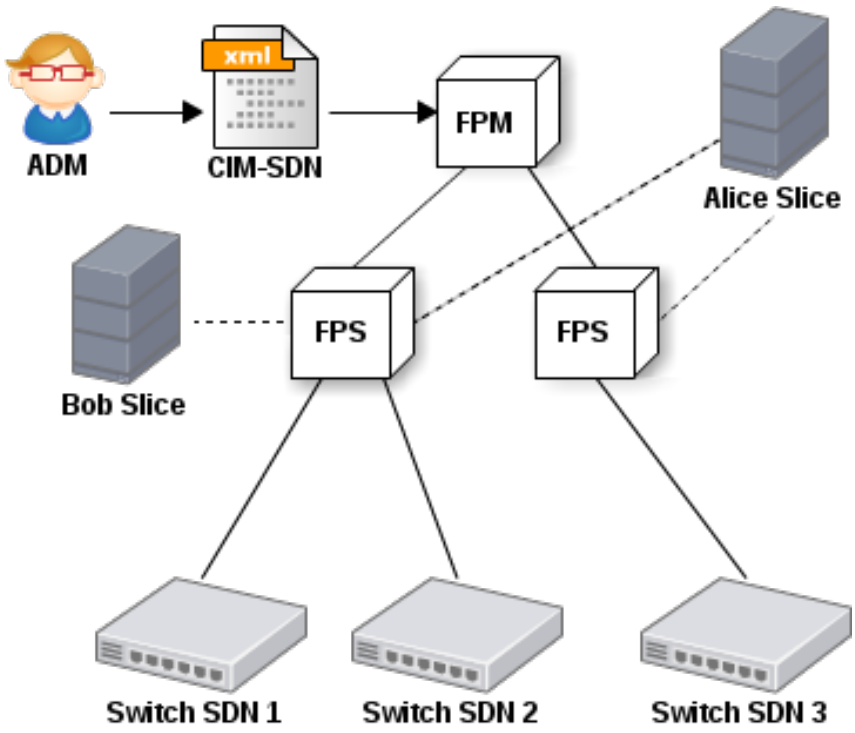


Figura 14: Aplicação do NVP [4].

Na Figura 15 são apresentados os diagramas de atividades do FPS e do FPM. No caso do FPS, as ações são iniciadas pelos controladores e comutadores, havendo, primeiramente, o registro dos comutadores e controladores no FPS e posteriormente o uso da ProxyTable para decidir sobre o encaminhamento dos pacotes. No FPM é o administrador que inicia as atividades, fornecendo o modelo CIM-SDN como entrada para gerar a criação dos slices. É importante observar que nunca um pacote é encaminhado ao FPM, apenas o FPS manipula pacotes e faz isso apenas usando a ProxyTable como fonte de decisão.

O uso do NVP é similar ao FlowVisor, tendo como diferença apenas o uso no CIM-SDN como entrada. No entanto, é possível a utilização direta do arquivo de configuração, fato este que possibilita a fácil integração com outras ferramentas que já tenham o FlowVisor como camada de virtualização.

4.2.3 Conclusões da Seção

O NVP tem sua arquitetura dividida em dois elementos principais: o Flow Proxy Switch (FPS) e o Flow Proxy Manager (FPM), visando prover, ao mesmo tempo, escalabilidade e uma visão global da rede. A escalabilidade acontece devido aos múltiplos FPSs que podem ser utilizados na rede a fim de prover o melhor uso dos recursos dos comutadores e controladores. A visão global é disponível pelo FPM, que contém todas as informações da rede e pode prover integração com outras redes através da interconexão com outros FPMs de outras redes.

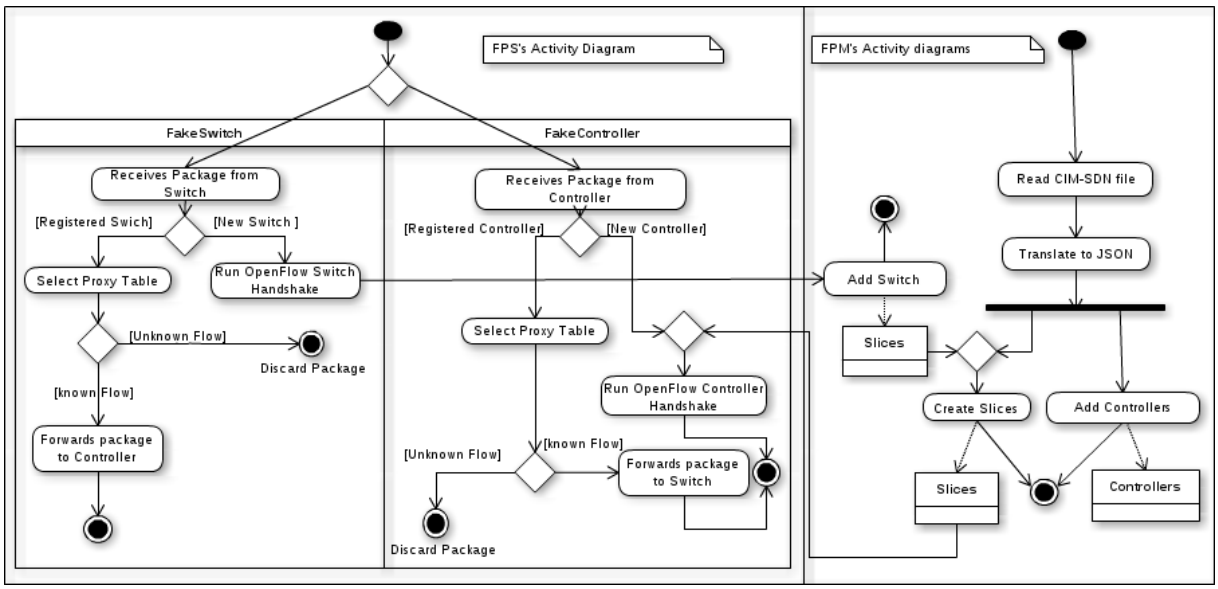


Figura 15: Diagramas de Atividade do NVP [4].

4.3 Graph Virtualization Layer

A resposta à questão de pesquisa III: **Como simplificar o entendimento de virtualização de forma a minimizar as questões técnicas?** é dado pelo GVL. Ele atua oferecendo maior abstração ao processo de virtualização, através do uso de uma abordagem SDN, considerando o paradigma e não apenas as tecnologias.

Ele é baseado no fato de que é possível usar modelos matemáticos para modelar redes de computadores e representar redes complexas, onde os vértices são os computadores(hosts e switches) e as arestas são os enlaces entre eles. O estudo de redes na forma de grafos é um dos pilares da matemática discreta, que começa em 1735 com Euler propondo uma solução para o problema da ponte de Königsberg, resultando na teoria dos grafos [64]. Assim, é possível definir uma rede de computador como um grafo no formato: $G = (V, E)$, onde V é um conjunto de vértices de rede e E um conjunto de enlaces físicos (arestas), sendo este grafo necessariamente conectado.

Quando estamos interessados em estabelecer comunicação entre dois elementos de uma rede G, N_s (source node) and N_d (destination node), deve existir um enlace direto (L) entre os elemento ou um caminho (Virtual Link - VL), que é a sequência de arestas e vértices da rede, isto é, $path(L)$, onde L é o conjunto de enlaces físicos interconectando dois pontos. A sub-rede é definida como um subconjunto dos elementos de uma rede formada por um subconjunto de V_s, V_d , enlace direto (direct link - L) ou enlaces virtuais (Virtual Link - VL) sobrepondo a rede física. Isto pode ser representado como um subgrafo de G, ou seja, $subG(V, E, VL)$.

A função essencial da rede é prover comunicação entre dois pontos V_s e V_d, mantendo os parâmentros de qualidade de serviço (Quality of Services - QoS) preestabelecidos, ou seja, descobrimento de caminhos entre dois pontos levando em considerações os diferentes pesos existentes nas arestas entre os vértices.

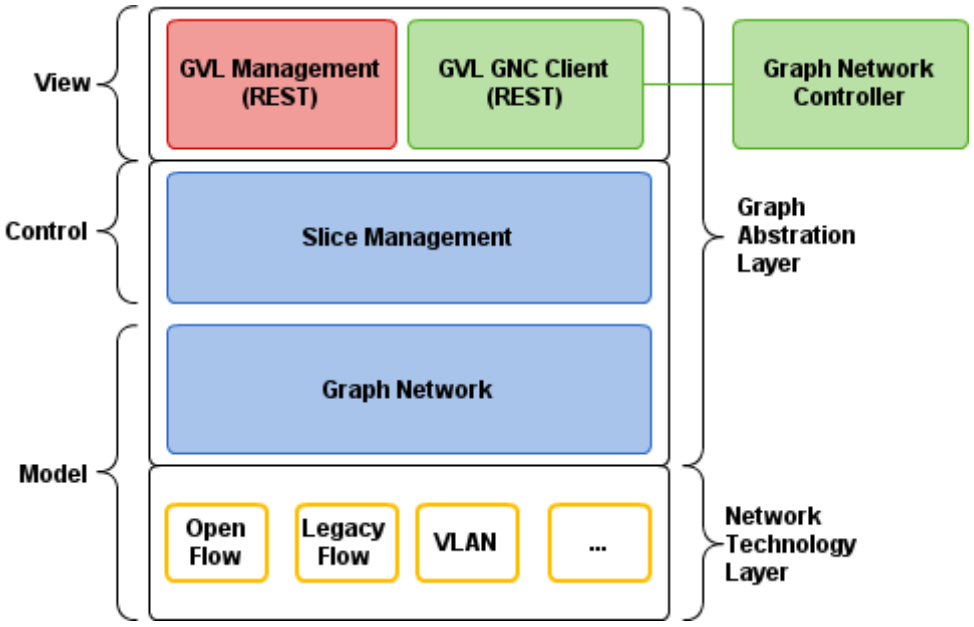


Figura 16: Arquitetura de Componentes do GVL.

Tradicionalmente, uma série de protocolos foram criados para prover comunicação com QoS, como MPLS, VPN e etc. Estas implementações foram necessárias devido à arquitetura da rede não possuir a mesma expressividade presente na teoria de grafos[20].

No contexto SDN, a arquitetura baseada em fluxos provê exatamente um modelo em grafos para gerenciar a rede, incluindo a visão global da rede [57], tornando possível simplificar a definição e o uso de redes usando teoria dos grafos.

A criação de sub-redes isoladas, usando campos da pilha de protocolo, é um requisito para habilitar virtualização de rede. Esta definição coincide com a definição de slices [10], que é, $Slice = subG(N, L, VL)$, um subconjunto da rede mais um conjunto de enlaces virtuais.

4.3.1 Arquitetura do GVL

A Figura 16 mostra a arquitetura de componentes do GVL. Na camada Model estão presentes os módulos que irão se comunicar com a tecnologia de rede utilizada(ex OpenFlow) e traduzí-la para grafo. Na camada Control estão as partes relacionadas à lógica de slices e na camada View esta presente a parte de comunicação com os controladores usando Notação de Objetos JavaScript (JavaScript Object Notation - JSON).

O GVL GNV Client é o componente responsável por encaminhar para o *Graph Network Controller* - GNC, as informações enviadas pelo *Graph Network*. Desta forma, o GNC recebe os pacotes de informações já organizadas em um formato de banco de dados em grafo.

No sentindo inverso, o GVL GNC Client recebe um grafo contendo os caminhos por onde os pacotes devem ser encaminhados da origem até o destino final. Sendo assim,

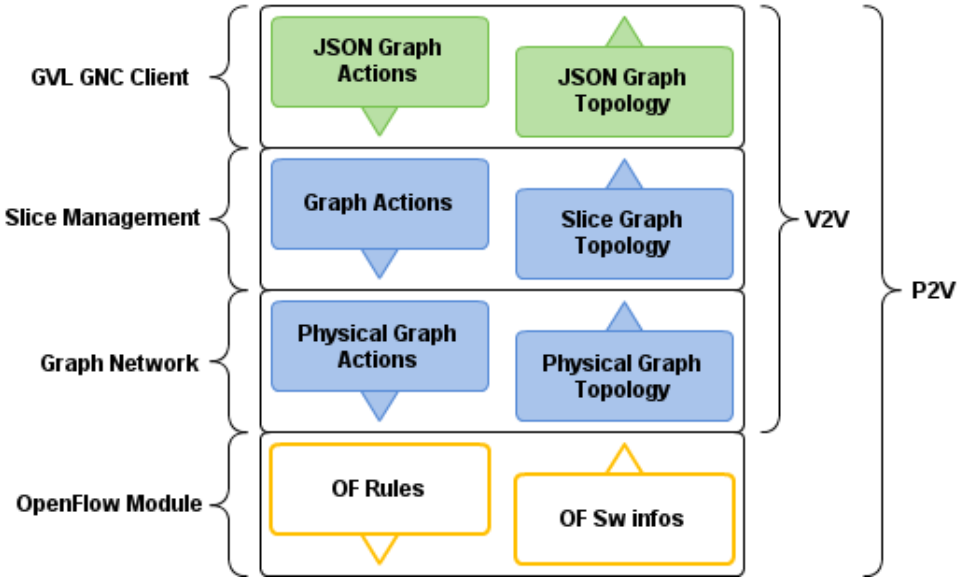


Figura 17: GVL MSG Architecture.

a tecnologia usada na *Graph Abstraction Layer* não é a mesma usada na camada de rede. Desta forma, esta escolha(usar apenas formato de grafo) é feita deliberadamente para abstrair as camadas inferiores e permitir que os desenvolvedores/gerentes possam desenvolver/gerenciar de maneira simples por meio do uso da abstração de grafos.

As mensagens trocadas entre o GVL GNC Client, o Graph Network e o GNC usam REpresentational State Transfer (REST) [65] para possibilitar a transferência dos grafos sem maiores custos de rede. A tradução da camada de grafo para a camada de tecnologia de rede é realizada pelo Graph Network.

O GVL, como mostrado na Figura 17, pode operar em dois modos: P2V (*physical to virtualization*) e V2V (*virtualization to virtualization*). Esses módulos são necessários para permitir a virtualização de uma rede já virtualizada, sem precisar manipular *switches*, ou seja, apenas o grafo da rede é exportado para o próximo nível de virtualização.

O diagrama de atividades apresentado na Figura 18 mostra como uma conexão iniciada por um *host* é tratada no GVL. Primeiro, o cabeçalho do pacote é usado para decidir quando ele deve ser encaminhado do *switch* para outro porta ou se ele deve ser enviado para o GVL onde o pacote ARP será tratado, seguindo uma abordagem centralizada de ARP[66]. Com o conhecimento do endereço MAC pelo GVL, uma requisição contendo o endereço MAC de origem e destino é enviado para o GNC associado ao slice, desta forma, o controlador pode gerar as regras que serão aplicadas nos *switches* da rede.

4.3.2 GVL Escalável

A proposta inicial do GVL pode ser incrementada para melhorar a escalabilidade da solução. A mesma abordagem utilizada pelo NVP [4] pode ser aplicado ao GVL, isto é, o plano de virtualização será dividido, centralizando as funções globais e menos

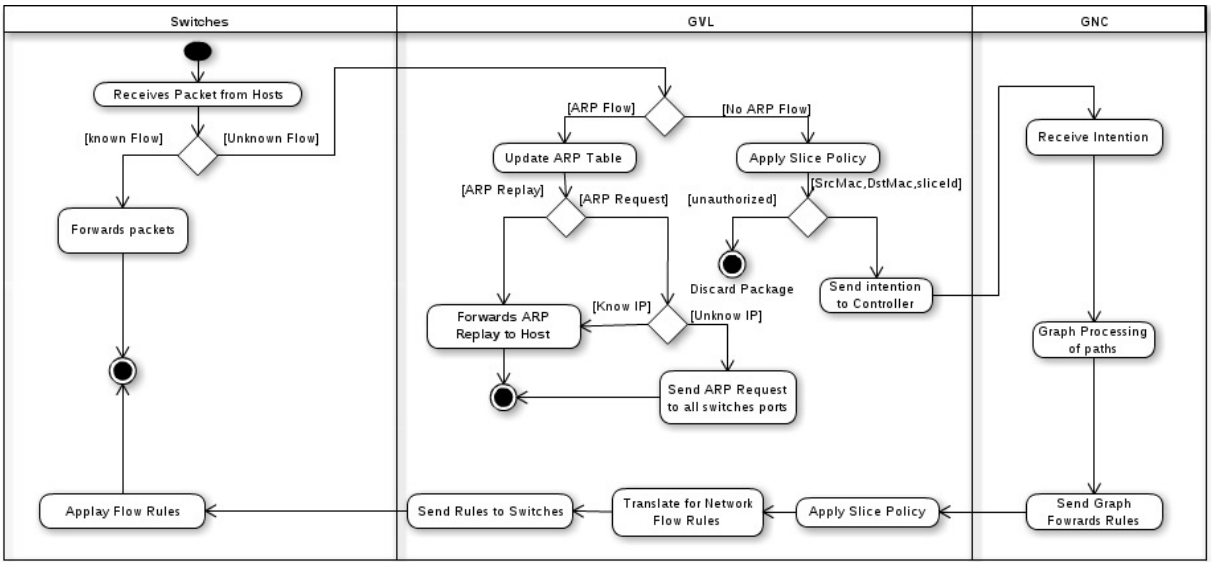


Figura 18: GVL Packet-in Activity Diagram.

acessadas da rede, distribuindo as funções que interagem ativamente com os switches e controladores.

A Figura 19 apresenta a arquitetura do GVL levando em conta os dois componentes existentes nele que proveem escalabilidade. Tais componentes herdam a mesma lógica aplicada no NVP, ou seja, o *Local Graph Virtualization* (LGV) e o *Global Graph Virtualization* (GGV) são análogos ao FPS e o FPM respectivamente.

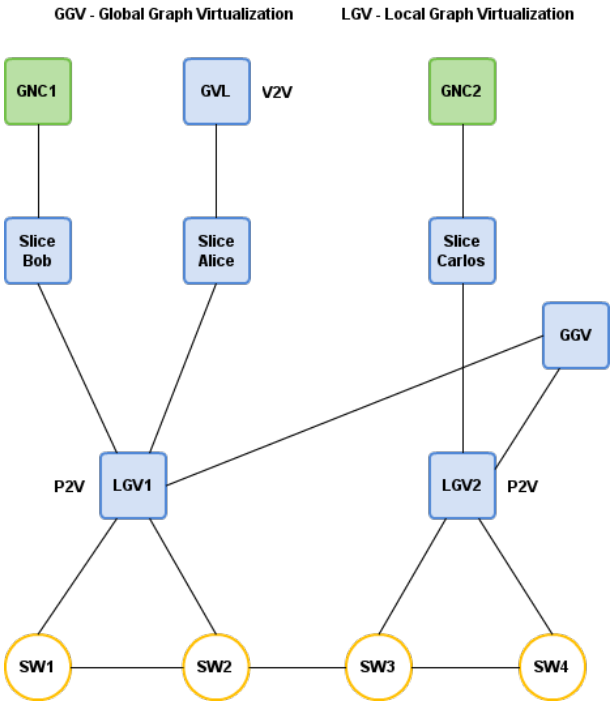


Figura 19: GVL Componentes Escaláveis.

4.3.2.1 Local Graph Virtualization

O LGV é responsável pelas funções executadas em um contexto local e que são bastante acessadas. Tais funções precisam estar distribuídas na rede para melhorar a escalabilidade da solução uma vez que estas são as mais requisitadas. São elas:

- Slice Flow Table: Tabela contendo as regras de como o fluxos devem ser tratados pelo virtualizador;
- Switch e Controller Interaction: responsável pela interação com os switches e o controladores;
- Gerência de ARP Local: Manter uma tabela com os ARPs já conhecidos da rede e coordenar o processo de descoberta dentro do seu contexto.

4.3.2.2 Global Graph Virtualization

O GGV é a parte central do GVL que provê a visão global da rede e coordena todos os LGV, alimentando suas Slice Flow Tables. As funções executadas por este elemento são pouco acessadas e suas informações pouco voláteis. São elas:

- Gerenciamento de Slice: Funções de criação e modificação de slices são feitas no GGV e depois repassadas ao LGV específicos;
- Gerência Global de ARP: Quando um determinado endereço não é encontrado dentro de um LGV, o GGV é acionado para encaminhar a solicitação aos demais LGV que fazem parte do slice.

O numero de instâncias do LGV na rede deve crescer conforme as limitações do hardware do equipamento que está executando o LGV e o tamanho do número de equipamentos associados a ele. Já o GGC deve possuir apenas uma instância em execução na rede.

4.3.3 Controladores SDN para o GVL

Com o desenvolvimento de controladores mais robustos para SDN, como o ONOS [67], um controlador que possui uma interface gráfica intuitiva, assim como possui aplicações instaladas por padrão e a habilidade de gerar tráfico de teste, se torna mais importe repensar algumas das necessidade de um controlador SDN.

Os controladores, como apresentados na Seção 2, duplicam parte do trabalho executado pelo virtualizador no mesmo contexto, uma vez que o virtualizador opera entre a camada física e o controlador, atuando como um falso *switch*/controlador, camuflando dessa forma sua própria presença.

Uma das funções do controlador SDN é analisar o estado da rede, prover informações da topologia para as aplicações, descoberta de dispositivos, distribuição de configurações de rede, gerência de dispositivos, rotear regras das aplicações para os switches, mecanismos de segurança, assim como receber, processar e rotear eventos.

O virtualizador intercepta todas as mensagens enviadas aos controladores envolvidos, processa as informações, divide a rede entre os controladores, através da configuração de slices e então reescreve as mensagens OpenFlow com as informações que cada controlador irá receber. Comparando as funções dos hypervisors e dos controladores é possível identificar que controladores realizam operações de controle de tráfico que são redundantes no contexto atual dos hypervisors e controladores SDN e se considerarmos um virtualizador de rede como o GVL, estas tarefas tornam-se desnecessárias.

Se um controlador não precisa de toda as informações de tráfego da rede, esta estrutura pode apenas considerar receber informações dos slices correspondentes e solicitar regras de roteamento para as aplicações. Nesse novo contexto, um controlador SDN para o GVL não mais deve interpretar OpenFlow e sim trabalhar com grafos, tornando a redes mais fáceis de gerenciar e os controladores menos complexos que os atuais.

4.3.4 Conclusão da Seção

Esta seção apresentou o GVL, um hypervisor para vSDNs, caracterizado pela tradução das mensagens de rede em uma linguagem de grafo, tornando os controladores SDN menos complexos ao retirar deles a necessidade de entender as mensagens de rede, permitindo que eles possam usar apenas a linguagem de grafos, ou seja, provendo maior abstração para as camadas superiores.

Esse incremento de abstração torna possível simplificar as redes, removendo complexidade das camadas superiores e tornando mais fácil o processo de desenvolver/gerenciar as redes, uma vez que melhores abstrações permitem um olhar diferente para problemas existentes na arquitetura tradicional, que eventualmente são migrados para o universo SDN.

4.4 Conclusão do Capítulo

As três propostas apresentadas neste capítulo respondem às três questões de pesquisa apresentadas no Capítulo 1. A resposta à Questão de Pesquisa I foi dada pelo CIM-SDN através da modelagem formal da rede, especificamente, do novos elementos presentes em vSDNs. A resposta à Questão de Pesquisa II foi dada pelo NVP ao separar o plano de virtualização, permitindo a distribuição em diferentes elementos de rede das funções mais utilizadas no hypervisor, enquanto que as funções que necessitam de visão global da rede permanecem centralizadas. Já a resposta para a questão de Pesquisa III foi o GVL, que propôs a utilização de abstração de grafos na comunicação entre o hypervisor e os controladores, permitindo assim uma mudança na maneira de pensar a rede.

CAPÍTULO 5

Análise do Trabalho

Este Capítulo está dividido em três seções, sendo cada uma referente às respostas apresentadas no Capítulo 4 que respondem às perguntas de pesquisa apresentadas no Capítulo 1. Cada seção contém uma descrição da implementação feita, a metodologia utilizada nos experimentos, quando aplicável, e a análise dos resultados obtidos.

5.1 CIM-SDN

Para fazer a modelagem de uma rede usando o CIM-SDN é necessária a criação de um diagrama de objetos que corresponde à instanciação do metamodelo (Classes) propostas pelo CIM-SDN. Tal modelagem pode ser realizada por qualquer ferramenta de modelagem UML que suporte diagramas de objetos. No entanto, para facilitar o processo de modelagem, uma ferramenta de Modelagem CIM-SDN [5], exibida na Figura 20, foi desenvolvida utilizando a biblioteca JsUML2 e está disponível online[1].

5.1.1 Uso do CIM-SDN

A rede presente na Figura 21 é um exemplo de uma modelagem usando diagramas convencionais. Além dos elementos de redes visíveis na figura, existem dois slices, sendo o Client1 e o SNMP Server parte do Slice1 e o Client2 e o HTTP Server parte do Slice2.

A Figura 22 apresenta um diagrama de objetos para a rede apresentada na Figura 21. Cada objeto é uma instância de uma classe presente no metamodelo CIM-SDN.

Usando o diagrama é possível identificar claramente as relações de dependência

[1] www.gercom.ufpa.br/cim-sdn

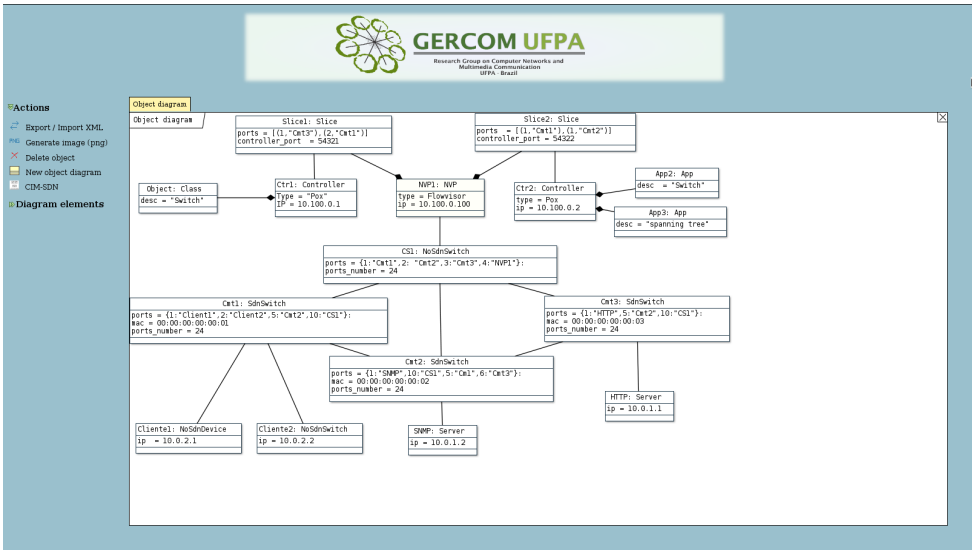


Figura 20: Ferramenta de Modelagem CIM-SDN [5].

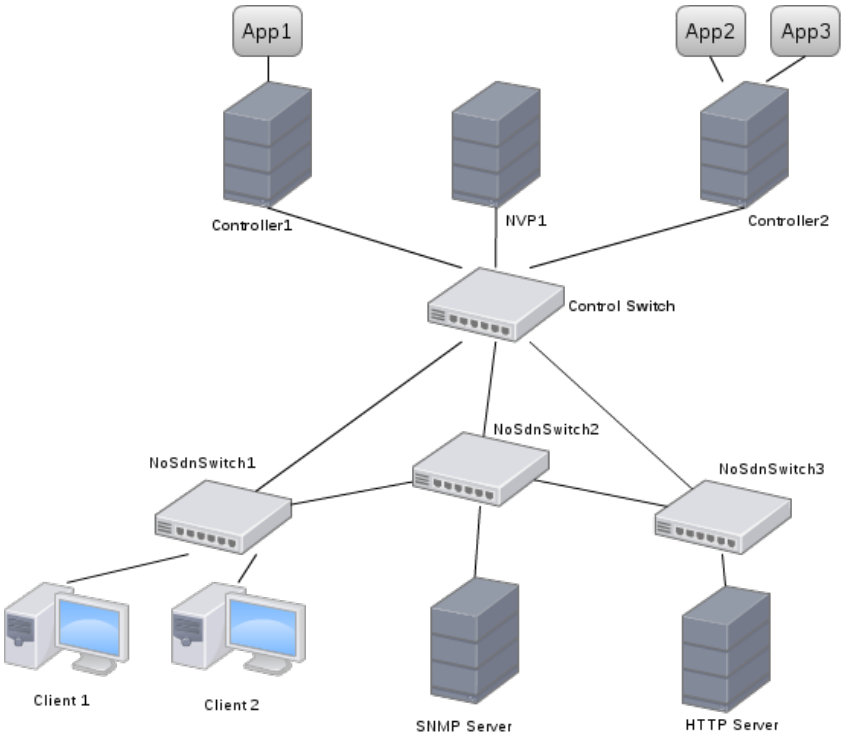


Figura 21: Diagrama Convencional

entre os elementos de rede e seus atributos (ex., o Cliente o SNMP Server pertencem ao Slice1). Alem disso, é possível utilizar o diagrama para melhorar o planejamento da rede, facilitando a identificação de pontos de falhas. Em seguida, 3 possíveis usos para o diagrama são destacados, mostrando como aplicá-los e seus benefícios.

- Documentação: Usando o diagrama é possível manter a documentação da rede de maneira formal usando UML. Entender o diagrama é simples, uma vez que a UML é amplamente aceita e estudada tanto pela academia como pela indústria, evitando

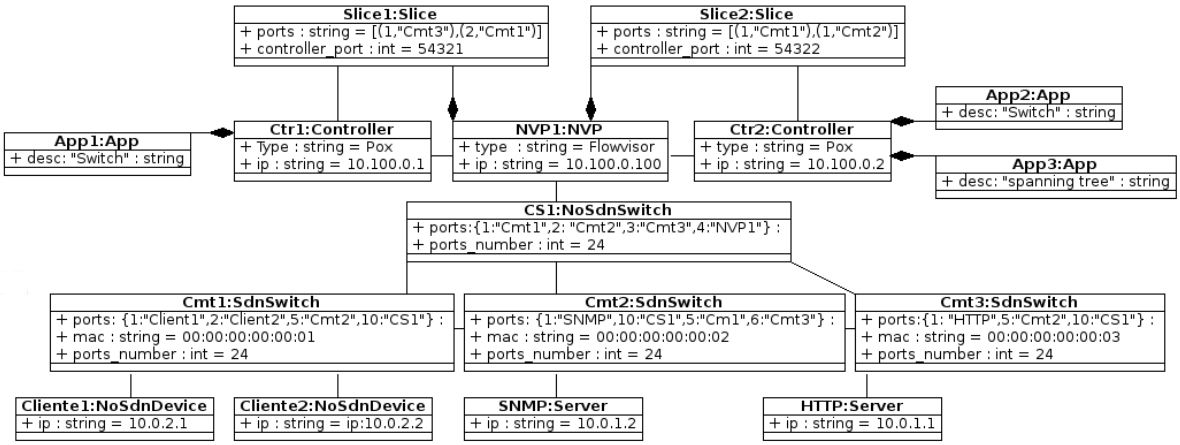


Figura 22: Diagrama de Objetos CIM-SDN

problemas de interpretação ao utilizar notações próprias;

- Gerar Configuração de rede: Com a rede modelada, é possível usar o arquivo com o modelo para gerar configurações para os elementos/ferramentas de rede. Tudo que é necessário é um pequeno script que converte as informações presentes no modelo, que já estão organizadas formalmente, nos formatos usados pelas ferramentas ou equipamentos de rede;

- Verificação de Inconsistências na Rede: O uso de UML com OCL (*Object Constraint Language*) se tornou uma das melhores maneiras de verificar sistemas modelados [68]. Assim, um conjunto de regras OCL foi criado para validar o modelo e procurar por inconsistências na rede antes da implantação desta, otimizando o uso dos recursos e evitando erros de projeto.

  Com os modelos prontos é possível aplicar um conjunto de regras OCL (*Object Constraint Language*) para validar o modelo e identificar inconsistências na rede [68].

  A Tabela 3 apresenta as regras OCL usadas. A primeira regra verifica se o número de ligações nos *switches* SDN são menores ou iguais ao número de portas disponíveis no equipamento, enquanto a segunda realiza verificação similar para *switches* convencionais. A terceira verifica se um slice associado a um controlador existe em um NVP.

5.1.2 Conclusão da Seção

Usando a proposta foi possível criar a documentação da rede usando UML, gerar arquivos de configuração a partir do modelo e realizar a verificação de consistência da rede usando OCL.

Tabela 3: OCL Rules

Number	Rule
1	context SdnSwitch inv : self.ports_number <= self.link_nosdnservers.size() + self.link_switchs.size()
2	context NoSdnSwitch inv : self.ports_number <= self.lik_sdnservers.size() + self.link_switchs.size()
3	context NVP inv : Controller.slices->notEmpty() implies self.slices->notEmpty()

5.2 NVP

Uma implementação para prova de conceito do NVP foi desenvolvida e o seu desempenho foi comparado com o FlowVisor e o OpenVirtex, sendo o número de requisições atendidas a métrica utilizada para avaliação. Além disso, foram mensurados valores de carga do NVP como uso de CPU e memória.

5.2.1 Implementação do NVP

Visando manter o uso mais eficiente possível da solução, o desenvolvimento do protótipo do NVP foi realizado em C++ com o uso da biblioteca BOOST::ASIO[2]. Nesta versão, os componentes FakeController e FakeSwitch foram implementados em sua totalidade além de uma versão resumida da ProxyTable.

A única biblioteca externa utilizada foi um arquivo de definição dos tipos de dados disponível no próprio OpenFlow para viabilizar a comunicação entre os comutadores e controladores.

A implementação parcial do FPS permite que ele atue como um proxy de virtualização, ainda que não possa contemplar todos os elementos da proposta, já que o FPM, apesar de já projetado, ainda não foi completamente implementado.

[2]http://www.boost.org/

5.2.2 Metodologia

A metodologia utilizada para avaliação e validação do NVP é baseada na execução da aplicação no ambiente real, porém tendo ferramentas que emulam comutadores OpenFlow e permitem a criação de um número elevado de elementos para o experimento, não sendo necessária a utilização de comutadores OpenFlow reais [69].

A ferramenta utilizada para emular os comutadores OpenFlow foi o CBENCH, que faz parte do framework oflops[3]. Com ele é possível definir o número de comutadores a serem emulados e indicar a qual controlador de rede ele deve se conectar.

O controlador utilizado nos experimentos foi o POX[4], tendo os comutadores emulados pelo CBENCH como clientes. Sendo que, os experimentos foram realizados na mesma máquina para que o tráfego de rede não influencie no tempo de resposta dos componentes. A máquina é uma CPU Intel core 2 quad 2.5 GHz com 4 GB de memória RAM, usando o Debian Whezzy.

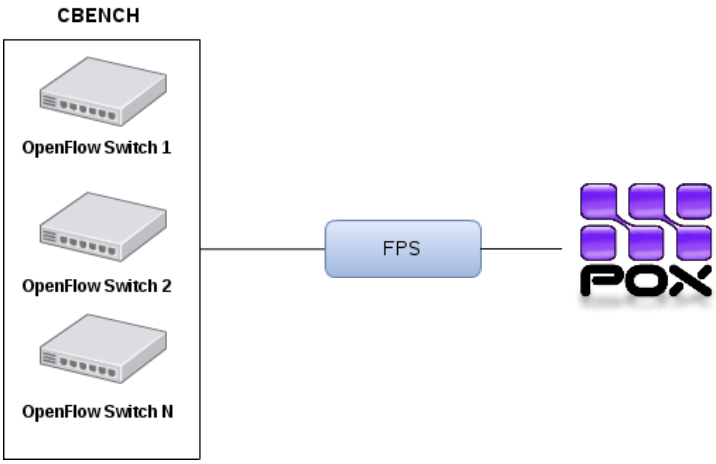


Figura 23: Arquitetura do Experimento.

A Figura 23 mostra como os elementos estão conectados. Foram realizadas 10 repetições para cada experimento, sendo usado o número de 1,5,10,20 e 40 switches, para gerarem requisições simultaneamente.

Nos experimentos realizados o hypervisor foi posicionado entre os switches e o controlador. Ou seja, em um experimento o NVP tomou a posição de virtualizador, no outro o FlowVisor, e por último o OpenVirtex. Um experimento sem virtualizador foi realizado para servir de parâmetro, ou seja, apenas os switches ligados diretamente ao POX.

[3]http://archive.openflow.org/wk/index.php/Oflops
[4]http://www.noxrepo.org/pox/about-pox/

5.2.3 Análise dos resultados

O objetivo desses experimentos é de validar, ainda que parcialmente, o NVP e mostrar testes de carga com ele para verificar seu comportamento. O POX é apresentado, não com o objetivo de comparação direta, pois eles executam tarefas diferentes na rede, mas para mostrar o valor do overhead criado quando existe um virtualizador na rede.

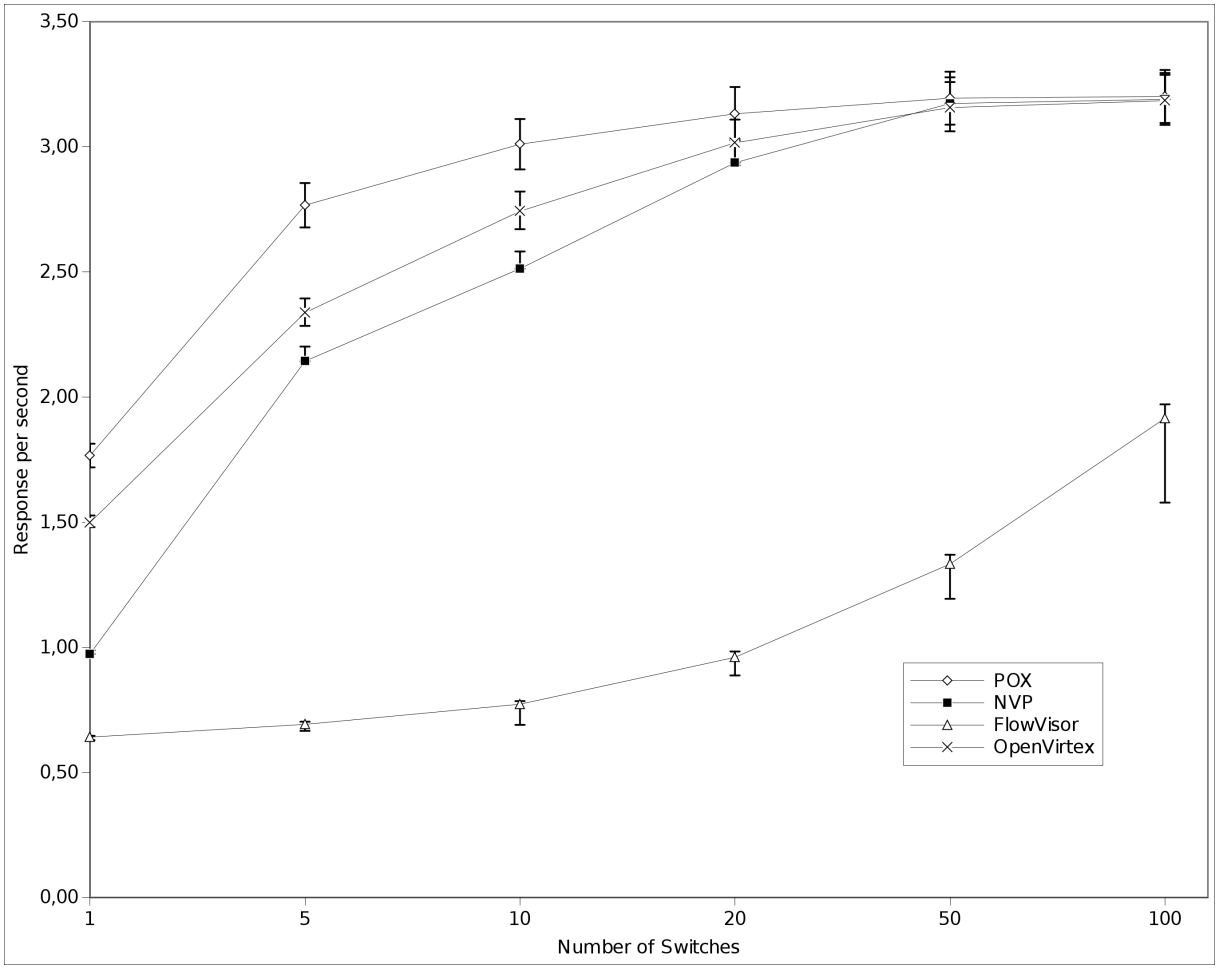


Figura 24: Resposta por segundo X Número de Switches

A Figura 24 e a Tabela 4 mostram o número de respostas que o controlador consegue responder por segundo. A comparação direta com o FlowVisor mostra uma grande diferença entre o numero de requisições respondida por cada Hypervisor. Para 1 switch o NVP é 34,17% melhor que o FlowVisor e quando o número de switches alcança 100, o resultado do NVP é 39,96% melhor, mostrando melhoria no desempenho. Quando o NVP é comparado com o OpenVirtex os resultados, que inicialmente são melhores para o OpenVirtex, vão se aproximando conforme o número de switches aumenta, até que para 100 swiches o NVP apresenta melhor resultado, com um incremento de 0,16

A Tabela 5 apresenta a comparação dos resultados do NVP com os demais, sendo que os valores negativos são as situações onde o NVP teve resultados piores. O POX é um valor de referencia e sempre será melhor, uma vez que não foi utilzado hypervisor no experimento.

Tabela 4: Resposta por segundo X Número de Switches

	1	5	10	20	50	100
NVP	0,97	2,14	2,51	2,94	3,17	3,19
POX	1,77	2,77	3,01	3,13	3,19	3,20
FlowVisor	0,64	0,69	0,77	0,96	1,33	1,91
OpenVirtex	1,50	2,34	2,74	3,02	3,16	3,18

Tabela 5: Percentual de melhoria do NVP comparado com as demais propostas.

	1	5	10	20	50	100
POX	-81,50	-29,9	-19,79	-6,66	-0,67	0,36
FlowVisor	34,13	67,72	69,26	67,31	58,00	39,96
OpenVirtex	-54,02	-9,05	-9,13	-2,71	0,51	0,16

Ainda na Figura 24 é possível verificar que em comparação com o POX (apenas o POX sem virtualizador) o NVP apresenta um número menor de resposta, como esperado, uma vez que as mensagens são mandadas ao POX de qualquer forma. No entanto, com o aumento no número de switches atendidos, esta diferença diminui. Para um switch, a perda de desempenho é de 81,50%, mas ela diminui para 29,09% com 5 switches e chega a 0,36% para 100 switches, mostrando uma melhora no desempenho quando o número de switches é elevado.

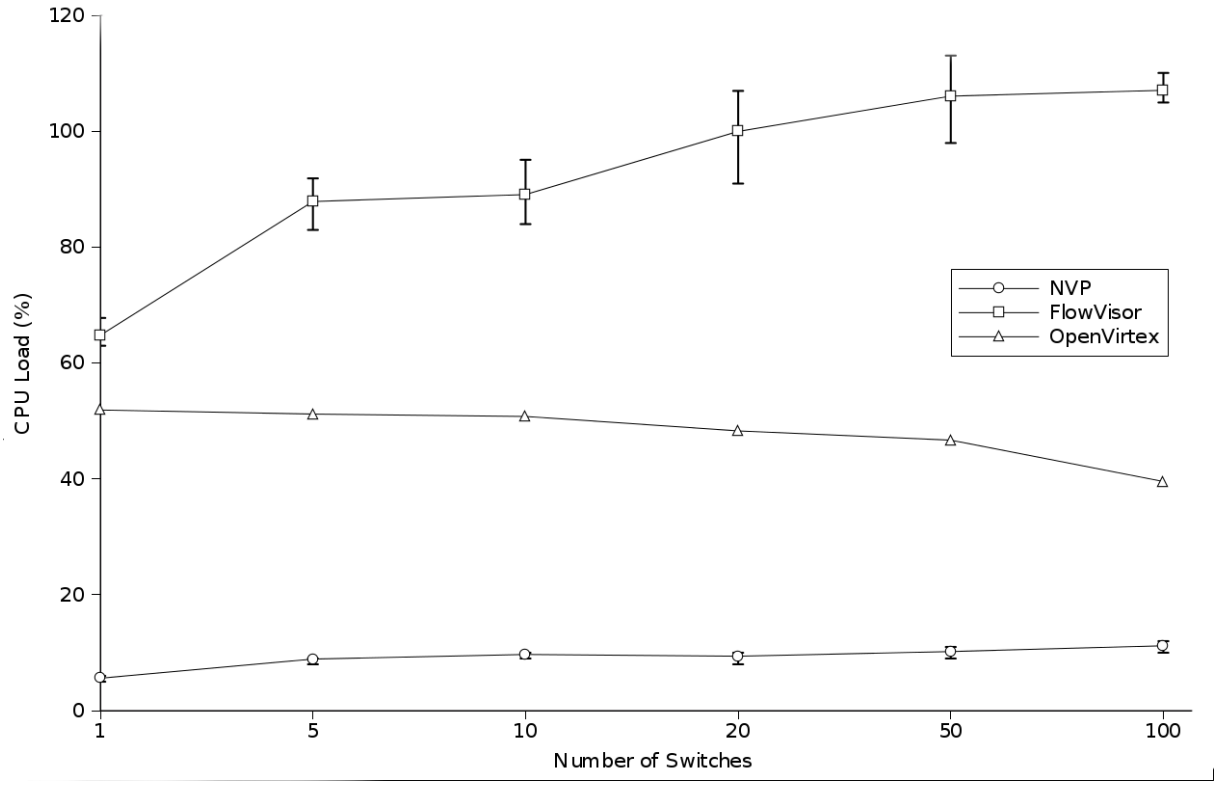


Figura 25: Tempo de Execução em segundos X Número de Switches

A Figura 25 apresenta a carga de uso de CPU em percentual do NVP FlowVisor e do OpenVirtex durante os experimentos. É possível verificar que, mesmo com 100 switches

ao mesmo tempo, o uso de CPU do NVP é menor que 12% enquanto o FlowVisor utiliza, pelo mesmo número de switches, 107% de CPU, ou seja, mais de 1 núcleo de CPU é necessário para atender as requisições. Já o OpenVirtex, apresenta resultados melhores que o FlowVisor, porém utiliza 3 vezes mais CPU quando comparado ao NVP com 100 switches.

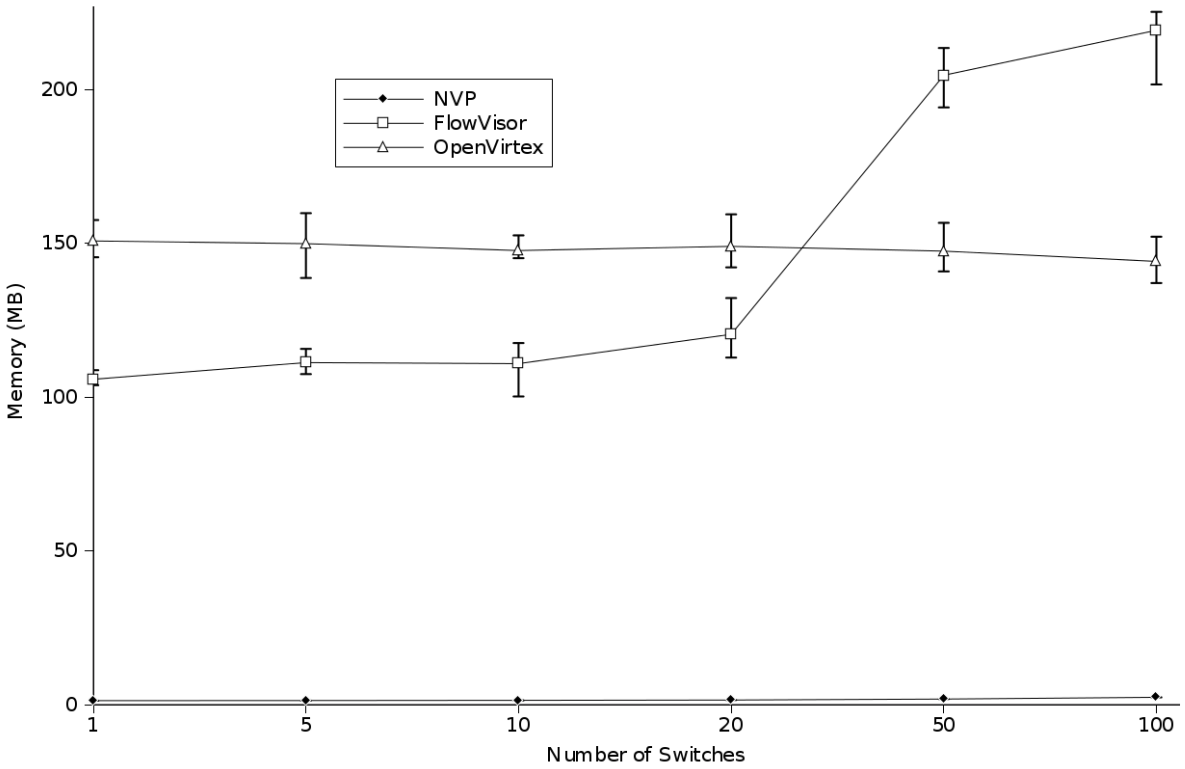


Figura 26: Uso de Memória X Número de Switches

A Figura 26 apresenta o consumo de memória do NVP, FlowVisor e OpenVirtex durante os experimentos, os valores apresentados são de RSS (Resident Set Size) que é a memória física sendo usada. Novamente, mesmo com 100 switches, o uso total de memória não ultrapassou os 2,5 MB, mostrando que a solução apresenta ume bom uso da memória e, no que tange a este aspecto, pode receber mais clientes que o FlowVisor, que usou mais de 200 MB para atender às requisições de 100 switches. A memória usada pelo OpenVirtex permanece quase constante em 150 MB, suplantando em quase 60 vezes a quantidade de memória utilizada pelo NVP.

5.2.4 Conclusão da Seção

Os resultados apresentados nesta seção, permitem fazer uma validação da proposta quanto ao uso de recursos da máquina que irá hospedar a solução de virtualização além de um comparativo com outras ferramentas de virtualização. Foi demonstrado que a atual implementação do NVP pode atuar como proxy de virtualização de rede sem precisar usar muitos recursos do hospedeiro.

5.3 GVL

Uma implementação para prova de conceito do GVL foi desenvolvida. Os testes, nesta fase, foram focados na demonstração do slice funcionando corretamente ao isolar tráfegos dentro do mesmo comutador.

5.3.1 Implementação do GVL

Esta implementação foi desenvolvida na línguagem C++ com o uso da LibFluid[5] como base, uma vez que esta oferece uma série de facilidades para o trabalho com OpenFlow, inclusive o suporte ao OpenFlow 1.3 que é o foco deste protótipo.

O principal objetivo desta prova de conceito é mostrar a viabilidade de um hypervisor que utilize grafo na comunicação com o controlador desta forma, simplificando os próprios controladores. Sendo assim, implementamos o GVL com componentes presentes na Figura 16, com a exceção da interface de gerência da ferramenta.

5.3.2 Metodologia

A metodologia utilizada para avaliação e validação do GVL é baseada na execução da aplicação no ambiente real, porém utilizando o Mininet[6] para emular a os comutadores, criar a topologia e instanciar os hosts onde foi executado o iperf[7].

O controlador utilizado nos experimentos foi uma implementação do GNC, que será devidamente discutida em outros trabalhos. Sendo que, os experimentos foram realizados na mesma máquina para que o tráfego de rede não influencie no tempo de resposta dos componentes. A máquina é uma CPU Intel core 2 quad 2.5 GHz com 4 GB de memória RAM, usando o Debian Whezzy.

As implementações do GVL e do GNC estão disponíveis online[8]. Tais versões ainda não estão prontas para produção, porém já podem ser utilizadas para comprovar o conceito que foi apresentado.

A Figura 27 mostra como os elementos estão dispostos nos experimentos. O hypervisor foi posicionado entre o switch e os controladores. Toda a topologia foi criada com o mininet, sendo esta composta por dois comutadores com 3 portas, estas portas foram divididas entre dois slices e os hosts conectados em cada slice foram configurados com a mesma sub-rede, o que em uma arquitetura tradicional iria provocar colisão.

No primeiro experimento dois tráfegos TCP foram inseridos na rede ao mesmo tempo por 120 segundos. No segundo experimento, os tráfegos começaram com uma

[5]https://github.com/OpenNetworkingFoundation/libfluid
[6]http://mininet.org/
[7]https://iperf.fr/
[8]https://gitlab.com/gercom/gvl/tree/Develop

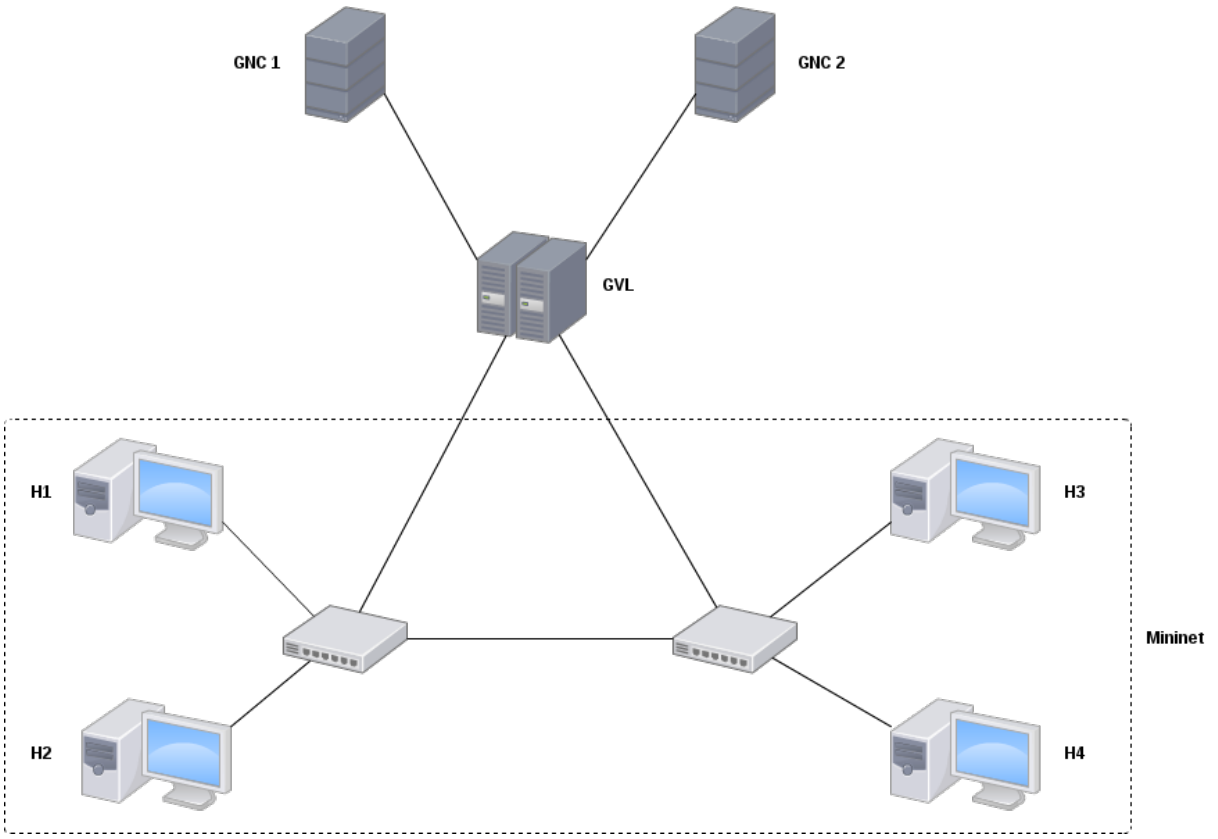


Figura 27: Arquitetura do Experimento.

diferença de 60 segundos entre eles e tiveram a duração individual de 120 segundos, ou seja, neste segundo caso o experimento completo demorou 180 segundos. Cada um dos experimentos foram executados 10 vezes para melhorar a confiabilidade dos resultados.

5.3.3 Análise dos resultados

O objetivo desses experimentos é de validar o conceito adotado pelo GVL, ainda que parcialmente. Demonstrando o isolamento provido pelo slice e a viabilidade de comunicação e representatividade do uso de grafo entre o hypervisor e os controladores.

A Figura 28 mostra a vazão em cada um dos slices quando dois tráfegos TCP são iniciados ao mesmo tempo, sendo um em cada slice. Podemos ver, então, que eles alcançam uma média de 15 Gbits/s apresentando uma concorrência equânime. Ainda é possível perceber, que houve a separação do domínio de colisão de maneira correta, uma vez que as redes configuradas nos slices são iguais. De fato, os hosts H1 e H2 possuem o mesmo ip (192.168.0.1), assim como os hosts H3 e H4 (192.168.0.2).

A Figura 29 apresenta a vazão quando os fluxos são iniciados em tempos distintos. Podemos ver que o primeiro fluxo consome toda a banda, em uma média de 30 Gbits/s, e que no momento em que o segundo tráfego é iniciado eles são estabilizados em aproximadamente 15 Gbits/s. Aos 120 segundos, o primeiro fluxo é finalizado e o segundo fluxo pode tomar controle da banda e chega a 30 Gbit/s. Tal comportamento

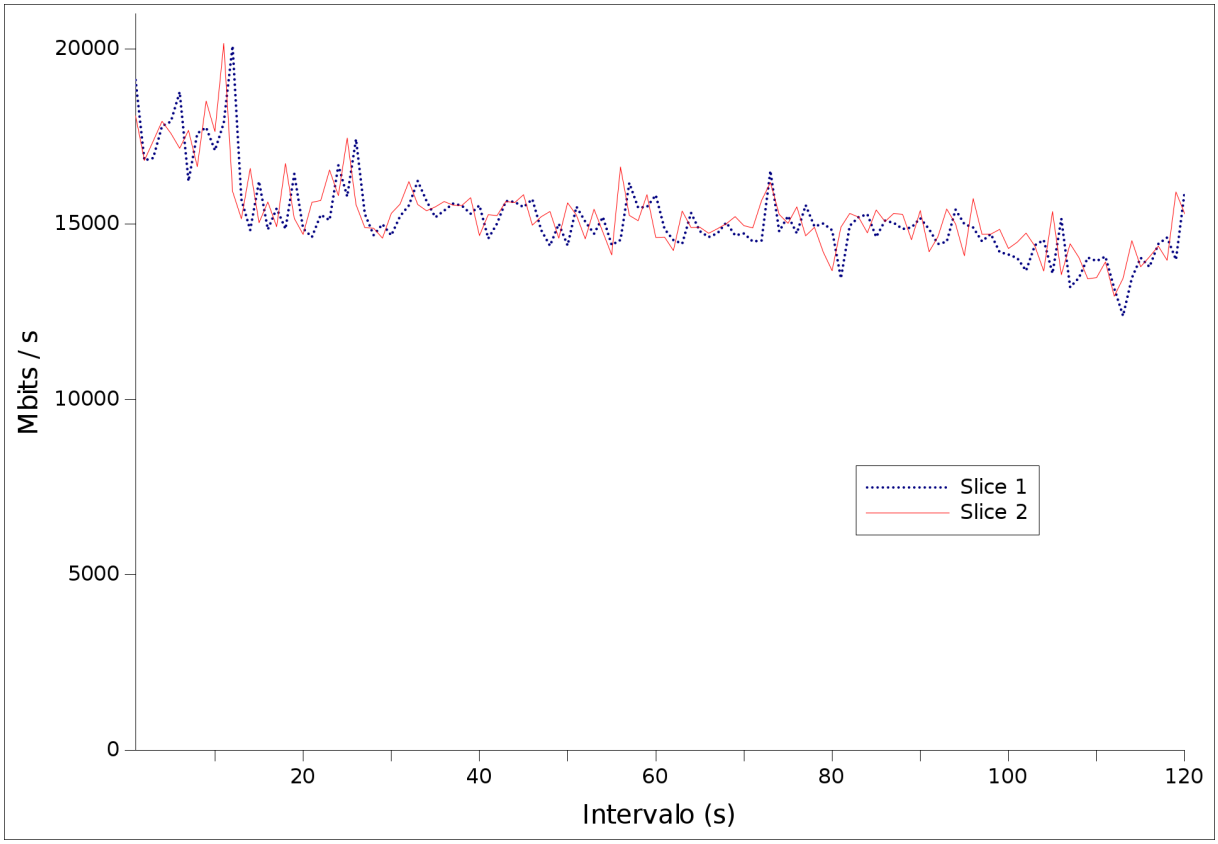


Figura 28: Experimento TCP de Vazão Concorrente X Intervalo de Tempo

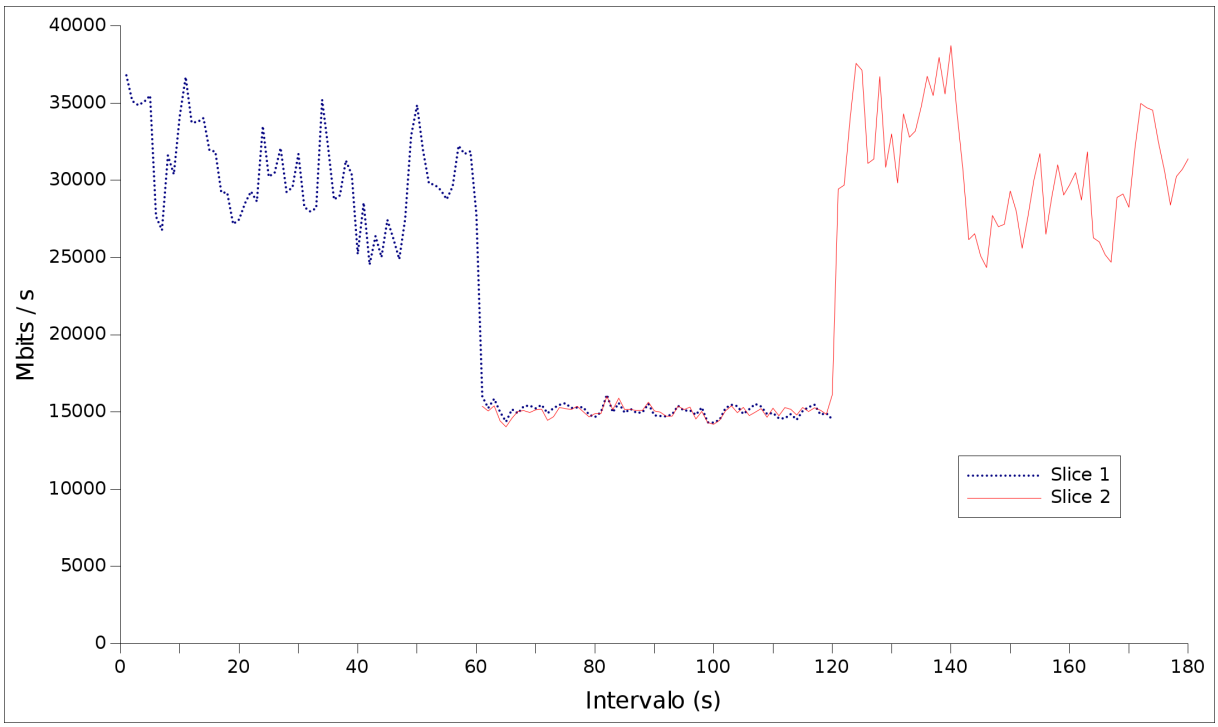


Figura 29: Experimento TCP de Vazão em tempos distintos X Intervalo de tempo

era esperado e demonstra que o GVL se comportou permitindo o isolamento dos tráfegos na rede.

O diagrama de atividades demonstrado na Figura 18 do Capítulo 4 apresenta as mensagens necessárias para que o GVL possa viabilizar a comunicação entre os hosts da rede, respeitando as políticas dos slices. Com o correto funcionamento dos experimentos apresentados nas Figuras 28 e 29 é possível verificar que as mensagens definidas no diagrama de atividades funcionaram corretamente pois o fluxo só pode ser estabelecido após a resolução dos endereços ARP envolvidos e o consequente encaminhamento do pedido de rota por parte do GVL ao GNC.

5.3.4 Conclusão da Seção

Os resultados apresentados nesta seção permitem fazer uma prova de conceito da proposta. Validando a manipulação de ARP pelo GVL, o isolamento do fluxo de controle pelos slices, as políticas de acesso a recursos e a criação de fluxos pelo GVL seguindo as ordens do GNC. Tal implementação demonstrou que é viável utilizar grafos para a comunicação entre o hypervisors e os controladores.

5.4 Conclusão do Capitulo

Este capítulo apresentou a validação e a análise de cada uma das 3 propostas apresentadas no Capítulo 4, que correspondem às respostas das perguntas apresentadas no Capítulo 1. Para o CIM-SDN foi demonstrado como efetuar seu uso e quais aplicações o modelo possui; Para o NVP, foram realizados testes de carga e comparativos com outros hypervisors e para o GVL foi exposto uma prova de conceito que validou o isolamento, a resolução de ARP e a iteração via grafos do hypervisor com o controlador.

CAPÍTULO 6

Considerações Finais e Trabalhos Futuros

Nesta seção apresentamos nossas conclusões sobre o que foi desenvolvido na tese, as respostas as questões de pesquisa elaboradas no Capitulo 1, as principais contribuições e algumas sugestões de trabalhos futuros a serem desenvolvidos.

6.1 Conclusões

A questão de pesquisa principal levantada nesta Tese foi: **Como oferecer virtualização em SDN de forma escalável, intuitiva e simplificada?**. Como resposta a esta questão, elaboramos a seguinte hipótese: **As limitações em vSDNs podem ser reduzidas através de uma abordagem SDN para a virtualização de redes.** Assumimos a abordagem SDN como aquela que apresenta a separação entre planos de dados(distribuído) e controle(centralizado), visão global da rede e uso de abstração de fluxo para gerir a comunicação entre os diferentes pontos da rede.

Para auxiliar no entendimento da questão principal e na comprovação de nossa hipótese, três questões de pesquisa foram formuladas, trabalhos relacionados a estas questões foram levantados, requisitos elicitados e, finalmente, uma resposta foi dada. Sendo que em cada uma delas abordamos uma parte da pergunta principal e, por conseguinte, da hipótese.

As três soluções desenvolvidas nesta Tese: CIM-SDN, NVP e o GVL suportam nossa hipótese ao demonstrar que o uso de representação formal dos novos elementos das vSDNs, separação do plano de controle em partes centralizadas e descentralizadas e maior uso de abstrações entre o hypervisors e o controlador, permitem que a virtualização em SDN possa ser intuitiva, escalável e simplificada.

6.2 Respostas Para as Questões de Pesquisa

A motivação por trás da definição das questões de pesquisa foi delimitar os principais pontos a serem analisados durante a investigação da hipótese e para estabelecer o roteiro para atingir as contribuições desta tese. Portanto, a descrição a seguir resume e destaca respostas para cada questão de pesquisa.

Questão de pesquisa I: Como apresentar elementos virtualizados e os novos elementos SDN?

Resposta: Os novos elementos presentes em vSDNs podem ser representados utilizando instâncias de classes que especifiquem cada um destes elementos, ou seja, a modelagem formal da rede é possível com a utilização de diagramas de classes e objetos. O CIM-SDN foi a extensão do CIM, proposta para, a partir dos elementos já existentes no CIM, modelar os novos elementos, tanto de virtualização (slices) como de SDN (controladores, hypervisors, comutadores, aplicações).

Questão de pesquisa II: Como implementar soluções escaláveis em SDN?

Resposta: O principal problema para a escalabilidade das vSDNs é justamente uma das principais características desta, a centralização do plano de controle, que acaba por sobrecarregar os hypervisors da rede. Para resolver este problema o NVP propôs a separação deste plano de controle centralizado em funções que podem ser executadas distribuídas nos casos em que a utilização do hypervisor é intensa.

Questão de pesquisa III: Como simplificar o entendimento de virtualização de forma a minimizar as questões técnicas?

Resposta: O uso de grafos, internamente, nos controladores e hypervisors é uma constante no universo SDN. No entanto, a comunicação entre eles sempre ocorre usando outra linguagem, perdendo, assim, representatividade. Desta forma, o GVL propôs a utilização de grafos na comunicação entre os controladores e os hypervisors, permitindo que abstrações de mais alto nível sejam utilizadas, aplicando de maneira mais intensa os conceitos de grafos para resolver a questões de comunicação na rede, trazendo maior simplicidade para os controladores e para a própria maneira de pensar a rede.

6.3 Contribuições

Provocar a discussão sobre a maneira como trabalhamos com SDN e virtualização é a contribuição mais ampla desta tese. A investigação de como é realizada a comunicação entre o hypervisor e os controladores e a consequente constatação da falta de abstrações mais poderosas como grafos nesta área permite que seja repensada a maneira como utilizamos SDN, levando a utilização maior do paradigma e não apenas da tecnologia.

Repensar a maneira como modelamos e representamos nossas redes é uma contribuição que não precisa ficar restrita ao universo das vSDNs. Tal formalismo na mo-

delagem traz benefícios tanto de compreensão, para os outros humanos envolvidos no processo, como para as entidades de software, uma vez que estes modelos formais podem ser utilizados para configurar a própria rede.

Listamos abaixo as principais contribuições que esta tese gerou:

- Um survey na área de modelagem de redes;
- Um survey dos hypervisor SDN;
- Definição dos requisitos de virtualização para uma SDN;
- Definição de procedimento para validar uma rede antes de sua implementação;
- Definição de metamodelos para a modelagem de vSDNs;
- Projeto de um framework escalável para vSDNs;
- Projeto de uma camada de virtualização usando Grafos;
- Implementação um hypervisor como proxy voltado a escalabilidade;
- Implementação um hypervisor usando grafos para se comunicar com os controladores.

Todos os temas específicos investigados e desenvolvidos nesta tese auxiliam na discussão de como vemos a própria área de estudo de redes. O foco em protocolos, em detrimento de abstrações, é evidenciado pela forma como ainda migramos problemas das redes tradicionais para as redes SDN. Desta forma, incentivar a reflexão sobre o próprio ensino de rede é uma contribuição que esta tese almeja, porem é um ponto que depende muito do leitor. Nesta tese lançamos a semente, esperamos que ela germine.

6.4 Trabalhos Futuros

Durante as investigações realizadas nesta tese, uma série de novas perguntas foram encontradas. Estas perguntas se tornam mais evidentes quando aplicamos o que foi desenvolvido nesta tese na rede.

Uso de outros diagramas da UML para modelar a execução da rede: No CIM-SDN elementos estruturais foram modelados, no entanto estados da rede que variam com o tempo não foram levados em consideração. Talvez a modelagem da rede usando outros diagramas, como o de atividades, possa ser útil para identificação de problemas durante a execução da rede e para unificar mecanismos de log e trocas de mensagens;

Definição de um plano de Virtualização e suas funções: Os elementos presentes no processo de virtualização parecem estar mais relacionados ao plano de gerência (slices, serviços, máquinas), porém uma parte considerável do plano de controle é redefinida dentro da virtualização e com a possibilidade de virtualizar uma rede já virtualizada

estas relações entre plano se tornam mais confusas. Talvez uma definição mais clara das funções presentes em cada plano e de como elas se relacionam possa melhorar o entendimento de redes virtualizadas.

Especificação de controladores para o GVL: A simplificação das funções de um controlador que está executando em uma rede com o GVL, gera a possibilidade de redefinir os controladores, tornando-os mais simples e alinhados com os conceitos de NFV.

Mecanismos de ARP para o GVL: Uma implementação simples de ARP centralizado foi desenvolvida para o GVL, porém, não fizemos uma discussão mais aprofundada sobre como o ARP, intrinsecamente distribuído, deve se comportar em uma rede com o GVL, onde a visão global nos permite centralizar uma série de informações.

Um serviço por slice: Se estendermos a ideia de um serviço por máquina, que já é extremamente usada na virtualização de máquinas, para a virtualização de redes. Teremos um controlador executando de maneira dedicada para um único serviço no slice. Este cenário abre discussão para a necessidade de aplicações executando diferentes funções e comportamentos dentro do controlador. Qual seria a utilidade de aplicações no controlador em uma ambiente de um serviço por slice?

Referências

[1] B. Pinheiro, R. Chaves, E. Cerqueira, and A. Abelem, "Cim-sdn: A common information model extension for software-defined networking," in *Globecom Workshops (GC Wkshps), 2013 IEEE*, Dec 2013, pp. 836–841.

[2] R. Sherwood, G. Gibb, K.-K. Yap, G. Appenzeller, M. Casado, N. McKeown, and G. Parulkar, "Flowvisor: A network virtualization layer," *OpenFlow Switch Consortium, Tech. Rep*, pp. 1–13, 2009.

[3] A. Blenk, A. Basta, M. Reisslein, and W. Kellerer, "Survey on network virtualization hypervisors for software defined networking," *CoRR*, vol. abs/1506.07275, 2015. [Online]. Available: http://arxiv.org/abs/1506.07275

[4] B. Pinheiro, E. Cerqueira, and A. Abelem, "Nvp: A network virtualization proxy for software defined networking," *International Journal of Computers Communications & Control*, vol. 11, no. 5, pp. 697–707, 2016. [Online]. Available: http://univagora.ro/jour/index.php/ijccc/article/view/2681

[5] A. Bahia, "Modelagem de redes definidas por software usando cim-sdn," Trabalho de Conclusao de Curso, 2013, universidade Federal do Para. [Online]. Available: Disponível em goo.gl/e2XdTe

[6] A. S. dos Santos, F. Le Gall, M. Baumberger, M. Y. Miyake, and R. Sathya, "A survey with owners and users of experimental facilities aimed at applications on future internet."

[7] D. Staessens, S. Sharma, D. Colle, M. Pickavet, and P. Demeester, "Software defined networking: Meeting carrier grade requirements," in *Local & Metropolitan Area Networks (LANMAN), 2011 18th IEEE Workshop on*. IEEE, 2011, pp. 1–6.

[8] L.-D. Chou, Y.-T. Yang, W.-P. Chang, Y.-S. Chen, T.-C. Chang, C.-K. Shieh, and S.-W. Huang, "Hierarchical management system of virtual networks on netfpga," in *Network Operations and Management Symposium (APNOMS), 2011 13th Asia-Pacific*. IEEE, 2011, pp. 1–4.

[9] A. Blenk, A. Basta, J. Zerwas, M. Reisslein, and W. Kellerer, "Control plane latency with sdn network hypervisors: The cost of virtualization," *IEEE Transactions on Network and Service Management*, vol. 13, no. 3, pp. 366–380, Sept 2016.

[10] R. Sherwood, G. Gibb, K. kiong Yap, M. Casado, N. Mckeown, and G. Parulkar, "Flowvisor: A network virtualization layer," Tech. Rep., 2009.

[11] E. Keller and J. Rexford, "The "platform as a service" model for networking," in *Proceedings of the 2010 internet network management conference on Research on enterprise networking*, vol. 4, no. 5. USENIX Association, 2010.

[12] M. Casado, T. Koponen, R. Ramanathan, and S. Shenker, "Virtualizing the network forwarding plane," in *Proceedings of the Workshop on Programmable Routers for Extensible Services of Tomorrow*. ACM, 2010, p. 8.

[13] S. Sallent, A. Abelém, I. Machado, L. Bergesio, S. Fdida, J. Rezende, S. Azodolmolky, M. Salvador, L. Ciuffo, and L. Tassiulas, "Fibre project: Brazil and europe unite forces and testbeds for the internet of the future," in *Testbeds and Research Infrastructure. Development of Networks and Communities*. Springer, 2012, pp. 372–372.

[14] S. 2, "Geni spiral 2 overview. relátorio técnico," http://groups.geni.net/geni/attachment/wiki/SpiralTwo/GENIS2Ovrvw060310.pdf, 2010.

[15] A. Köpsel and H. Woesner, "Ofelia: pan-european test facility for openflow experimentation," in *Proceedings of the 4th European conference on Towards a service-based internet*, ser. ServiceWave'11. Berlin, Heidelberg: Springer-Verlag, 2011, pp. 311–312. [Online]. Available: http://dl.acm.org/citation.cfm?id=2050869.2050905

[16] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "Openflow: enabling innovation in campus networks," *SIGCOMM Comput. Commun. Rev.*, vol. 38, no. 2, pp. 69–74, Mar. 2008. [Online]. Available: http://doi.acm.org/10.1145/1355734.1355746

[17] N. Gude, T. Koponen, J. Pettit, B. Pfaff, M. Casado, N. McKeown, and S. Shenker, "Nox: Towards an operating system for networks," *SIGCOMM Comput. Commun. Rev.*, vol. 38, no. 3, pp. 105–110, Jul. 2008. [Online]. Available: http://doi.acm.org/10.1145/1384609.1384625

[18] Open Networking Foundation, "Software-Defined Networking: The New Norm for Networks," Open Networking Foundation, Palo Alto, CA, USA, White paper, Apr. 2012. [Online]. Available: http://www.opennetworking.org/images/stories/downloads/sdn-resources/white-papers/wp-sdn-newnorm.pdf

[19] J. F. Kurose and K. W. Ross, *Redes de Computadores e a Internet: Uma abordagem top-down*, trad. 5 ed. ed. São Paulo: Addison Wesley, 2010.

[20] J. Day, *Patterns in Network Architecture: A Return to Fundamentals*.

[21] M. Casado, N. Foster, and A. Guha, "Abstractions for software-defined networks," *Commun. ACM*, vol. 57, no. 10, pp. 86–95, Sep. 2014. [Online]. Available: http://doi.acm.org/10.1145/2661061.2661063

[22] J. L. King, "Centralized versus decentralized computing: Organizational considerations and management options," *ACM Comput. Surv.*, vol. 15, no. 4, pp. 319–349, Dec. 1983. [Online]. Available: http://doi.acm.org/10.1145/289.290

[23] D. Marschke, J. Doyle, and P. Moyer, *Software Defined Networking (SDN): Anatomy of OpenFlow*. Lulu Publishing Services, 2015, no. v. 1. [Online]. Available: https://books.google.com.br/books?id=ecM5CgAAQBAJ

[24] N. M. K. Chowdhury and R. Boutaba, "A survey of network virtualization," *Comput. Netw.*, vol. 54, no. 5, pp. 862–876, Apr. 2010. [Online]. Available: http://dx.doi.org/10.1016/j.comnet.2009.10.017

[25] S. Gutz, A. Story, C. Schlesinger, and N. Foster, "Splendid isolation: A slice abstraction for software-defined networks," in *Proceedings of the First Workshop on Hot Topics in Software Defined Networks*, ser. HotSDN '12. New York, NY, USA: ACM, 2012, pp. 79–84. [Online]. Available: http://doi.acm.org/10.1145/2342441.2342458

[26] R. Jain and S. Paul, "Network virtualization and software defined networking for cloud computing: a survey," *Communications Magazine, IEEE*, vol. 51, no. 11, pp. 24–31, 2013.

[27] D. Levin, A. Wundsam, B. Heller, N. Handigol, and A. Feldmann, "Logically centralized?: state distribution trade-offs in software defined networks," in *Proceedings of the first workshop on Hot topics in software defined networks*, ser. HotSDN '12. New York, NY, USA: ACM, 2012, pp. 1–6. [Online]. Available: http://doi.acm.org/10.1145/2342441.2342443

[28] S. Bhatia, M. Motiwala, W. Muhlbauer, Y. Mundada, V. Valancius, A. Bavier, N. Feamster, L. Peterson, and J. Rexford, "Trellis: A platform for building flexible, fast virtual networks on commodity hardware," in *Proceedings of the 2008 ACM CoNEXT Conference*, ser. CoNEXT '08. New York, NY, USA: ACM, 2008, pp. 72:1–72:6. [Online]. Available: http://doi.acm.org/10.1145/1544012.1544084

[29] R. Sherwood, M. Chan, A. Covington, G. Gibb, M. Flajslik, N. Handigol, T.-Y. Huang, P. Kazemian, M. Kobayashi, J. Naous, S. Seetharaman, D. Underhill, T. Yabe, K.-K. Yap, Y. Yiakoumis, H. Zeng, G. Appenzeller, R. Johari, N. McKeown, and G. Parulkar, "Carving research slices out of your production networks with openflow," *SIGCOMM Comput. Commun. Rev.*, vol. 40, no. 1, pp. 129–130, Jan. 2010. [Online]. Available: http://doi.acm.org/10.1145/1672308.1672333

[30] F. N. Farias, J. M. D. Júnior, J. J. Salvatti, S. Silva, A. J. Abelém, M. R. Salvador, and M. A. Stanton, "Pesquisa experimental para a internet do futuro: Uma proposta utilizando virtualização e o framework openflow," *Minicursos*, vol. 1, pp. 1–61, 2011.

[31] L. Subramanian, I. Stoica, H. Balakrishnan, and R. H. Katz, "Overqos: An overlay based architecture for enhancing internet qos," in *NSDI*, vol. 4, 2004, p. 6.

[32] T. Anderson and M. K. Reiter, "Geni: Global environment for network innovations distributed services working group," 2006.

[33] B. Krishnamurthy, C. Wills, and Y. Zhang, "On the use and performance of content distribution networks," in *Proceedings of the 1st ACM SIGCOMM Workshop on Internet Measurement*. ACM, 2001, pp. 169–182.

[34] E. K. Lua, J. Crowcroft, M. Pias, R. Sharma, and S. Lim, "A survey and comparison of peer-to-peer overlay network schemes," *Communications Surveys Tutorials, IEEE*, vol. 7, no. 2, pp. 72–93, 2005.

[35] F. Dabek, M. F. Kaashoek, D. Karger, R. Morris, and I. Stoica, "Wide-area cooperative storage with cfs," *ACM SIGOPS Operating Systems Review*, vol. 35, no. 5, pp. 202–215, 2001.

[36] L. Peterson, S. Muir, T. Roscoe, and A. Klingaman, "Planetlab architecture: An overview," *PlanetLab Consortium May*, 2006.

[37] M. Campanella, "The federica project: creating cloud infrastructures," *Proceedings of Cloudcomp*, pp. 19–21, 2009.

[38] "Geni system overview. relatório técnico," http://groups.geni.net/geni/attachment/wiki/GeniSysOvrvw/GENISysOvrvw092908.pdf, 2008.

[39] R. Sherwood, G. Gibb, K.-K. Yap, G. Appenzeller, M. Casado, N. McKeown, and G. M. Parulkar, "Can the production network be the testbed?" in *OSDI*, vol. 10, 2010, pp. 1–14.

[40] H. Hawilo, A. Shami, M. Mirahmadi, and R. Asal, "Nfv: state of the art, challenges, and implementation in next generation mobile networks (vepc)," *IEEE Network*, vol. 28, no. 6, pp. 18–26, Nov 2014.

[41] J. Medved, R. Varga, A. Tkacik, and K. Gray, "Opendaylight: Towards a model-driven sdn controller architecture," in *Proceeding of IEEE International Symposium on a World of Wireless, Mobile and Multimedia Networks 2014*, 2014.

[42] G. Booch, J. Rumbaugh, and I. Jacobson, *Unified Modeling Language User Guide, The (2Nd Edition) (Addison-Wesley Object Technology Series)*. Addison-Wesley Professional, 2005.

[43] V. Saxena and D. Arora, "Uml modeling of network topologies for distributed computer system," *CIT. Journal of Computing and Information Technology*, vol. 17, no. 4, pp. 327–334, 2009.

[44] W. Fuertes, J. Lopez de Vergara, F. Meneses, and F. Galan, "A generic model for the management of virtual network environments," in *Network Operations and Management Symposium (NOMS), 2010 IEEE*, April 2010, pp. 813–816.

[45] J. Zhao, D. Yang, D. Tang, and X. Fang, "The application of uml in zigbee networks," in *Measuring Technology and Mechatronics Automation (ICMTMA), 2010 International Conference on*, vol. 3, 2010, pp. 794–798.

[46] R. Khare, M. Khadem, S. Moorty, K. Methaprayoon, and J. Zhu, "Patterns and practices for cim applications," in *Power and Energy Society General Meeting, 2011 IEEE*, 2011, pp. 1–8.

[47] C. Narsimha Reddy, "A cim (common information model) based management model for clouds," in *Cloud Computing in Emerging Markets (CCEM), 2012 IEEE International Conference on*, Oct 2012, pp. 1–5.

[48] J. Matias, B. Tornero, A. Mendiola, E. Jacob, and N. Toledo, "Implementing layer 2 network virtualization using openflow: Challenges and solutions," in *Software Defined Networking (EWSDN), 2012 European Workshop on*, 2012, pp. 30–35.

[49] E. Salvadori, R. Corin, A. Broglio, and M. Gerola, "Generalizing virtual network topologies in openflow-based networks," in *Global Telecommunications Conference (GLOBECOM 2011), 2011 IEEE*, 2011, pp. 1–6.

[50] R. Corin, M. Gerola, R. Riggio, F. De Pellegrini, and E. Salvadori, "Vertigo: Network virtualization and beyond," in *Software Defined Networking (EWSDN), 2012 European Workshop on*, 2012, pp. 24–29.

[51] V. S. Gomes, A. Ishimori, I. M. A. Peres, F. N. Farias, E. C. Cerqueira, and A. J. Abelém, "Flowvisorqos: aperfeiçoando o flowvisor para aprovisionamento de recursos em redes virtuais definidas por software."

[52] A. Ishimori, J. Salvatti, F. Farias, L. Gaspary, L. Granville, E. Cerqueira, and A. Abelém, "Qosflow: Gerenciamento automático da qualidade de serviço em infraestruturas de experimentação baseadas em framework openflow," in *III Workshop de Pesquisa Experimental da Internet do Futuro-Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos (SBRC 2012), Ouro Preto*, 2012.

[53] A. Al-Shabibi, M. De Leenheer, M. Gerola, A. Koshibe, G. Parulkar, E. Salvadori, and B. Snow, "Openvirtex: Make your virtual sdns programmable," in *Proceedings of the third workshop on Hot topics in software defined networking*. ACM, 2014, pp. 25–30.

[54] *OpenVirtex System Requirements.*, OnLab Std., [Online; Ultimo acesso em 02/11/2016]. [Online]. Available: http://ovx.onlab.us/getting-started/installation/

[55] D. Drutskoy, E. Keller, and J. Rexford, "Scalable network virtualization in software-defined networks," *Internet Computing, IEEE*, vol. 17, no. 2, pp. 20–27, 2013.

[56] A. Blenk, A. Basta, and W. Kellerer, "Hyperflex: An sdn virtualization architecture with flexible hypervisor function allocation," in *2015 IFIP/IEEE International Symposium on Integrated Network Management (IM)*. IEEE, 2015, pp. 397–405.

[57] D. Levin, A. Wundsam, B. Heller, N. Handigol, and A. Feldmann, "Logically centralized?: state distribution trade-offs in software defined networks," in *Proceedings of the first workshop on Hot topics in software defined networks*. ACM, 2012, pp. 1–6.

[58] R. Raghavendra, J. Lobo, and K.-W. Lee, "Dynamic graph query primitives for sdn-based cloudnetwork management," in *Proceedings of the first workshop on Hot topics in software defined networks*. ACM, 2012, pp. 97–102.

[59] G. Pantuza, F. Sampaio, L. F. Vieira, D. Guedes, and M. A. Vieira, "Network management through graphs in software defined networks," in *10th International Conference on Network and Service Management (CNSM) and Workshop*. IEEE, 2014, pp. 400–405.

[60] E. Salvadori, R. D. Corin, A. Broglio, and M. Gerola, "Generalizing virtual network topologies in openflow-based networks," in *Global Telecommunications Conference (GLOBECOM 2011), 2011 IEEE*. IEEE, 2011, pp. 1–6.

[61] ETSI, "Network functions virtualisation (nfv); use cases," 2014. [Online]. Available: http://www.etsi.org/deliver/etsi_gs/NFV/001_099/001/01.01.01_60/gs_NFV001v010101p.pdf

[62] "Dmtf schema documentation schema documentation," http://schemas.dmtf.org/wbem/cim-html/2.34.0+/CIM_VirtualComputerSystem.html, [Online; Last accessed 04-July-2013].

[63] *Virtual Ethernet Switch Profile*, DMTF Std., Rev. 1.1.0, 2012. [Online]. Available: http://www.dmtf.org/sites/default/files/standards/documents/DSP1097_1.1.0.pdf

[64] A.-L. Barabasi, *Linked: How Everything Is Connected to Everything Else and What It Means for Business, Science, and Everyday Life*. Plume Books, April 2003. [Online]. Available: http://www.amazon.ca/exec/obidos/redirect?tag=citeulike04-20&path=ASIN/0452284392

[65] M. Jakl, "Rest representational state transfer," 2008.

[66] H. Cho, S. Kang, and Y. Lee, "Centralized arp proxy server over sdn controller to cut down arp broadcast in large-scale data center networks," in *2015 International Conference on Information Networking (ICOIN)*. IEEE, 2015, pp. 301–306.

[67] P. Berde, M. Gerola, J. Hart, Y. Higuchi, M. Kobayashi, T. Koide, B. Lantz, B. O'Connor, P. Radoslavov, W. Snow, and G. Parulkar, "Onos: Towards an open, distributed sdn os," in *Proceedings of the Third Workshop on Hot Topics in Software Defined Networking*, ser. HotSDN '14. New York, NY, USA: ACM, 2014, pp. 1–6. [Online]. Available: http://doi.acm.org/10.1145/2620728.2620744

[68] J. Cabot, R. Clariso, E. Guerra, and J. Lara, "A uml/ocl framework for the analysis of graph transformation rules," *Software & Systems Modeling*, vol. 9, pp. 335–357, 2010. [Online]. Available: http://dx.doi.org/10.1007/s10270-009-0129-0

[69] M. Jarschel, S. Oechsner, D. Schlosser, R. Pries, S. Goll, and P. Tran-Gia, "Modeling and performance evaluation of an openflow architecture," in *Proceedings of the 23rd International Teletraffic Congress*, ser. ITC '11. International Teletraffic Congress, 2011, pp. 1–7. [Online]. Available: http://dl.acm.org/citation.cfm?id=2043468.2043470