UNIVERSIDADE FEDERAL DE SANTA MARIA
CENTRO DE TECNOLOGIA
PROGRAMA DE PÓS-GRADUAÇÃO EM INFORMÁTICA

Lucas Powaczuk

OPENVISOR – FRAMEWORK PARA REDES DE EXPERIMENTAÇÃO OPENFLOW

Santa Maria, 2016
2016

Lucas Powaczuk

OPENVISOR – FRAMEWORK PARA REDES DE EXPERIMENTAÇÃO OPENFLOW

Dissertação de mestrado apresentado ao Programa de Pós-Graduação em Informática da Universidade Federal de Santa Maria (UFSM), como requisito parcial para a obtenção do título de **Mestre em Ciência da Computação**.

Orientador: Dr.a Roseclea Duarte Medina

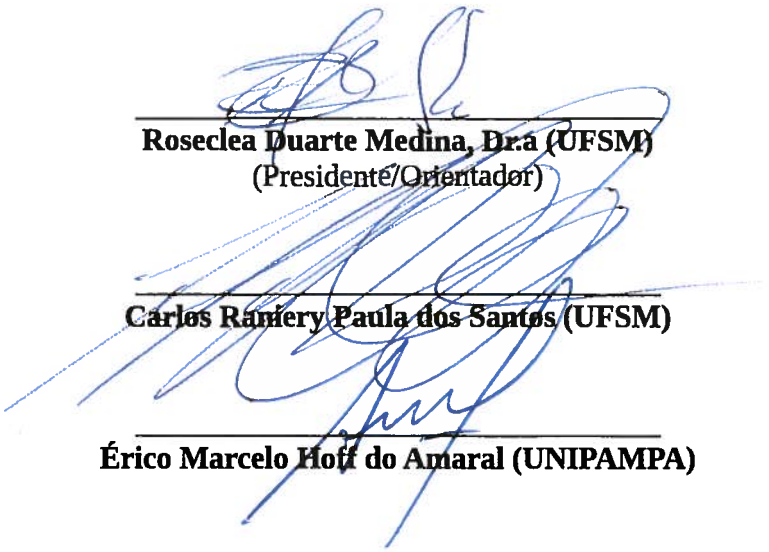
Santa Maria, RS
2016

Lucas Powaczuk

OPENVISOR – FRAMEWORK PARA REDES DE EXPERIMENTAÇÃO OPENFLOW

Dissertação de mestrado apresentado ao Programa de Pós-Graduação em Informática da Universidade Federal de Santa Maria (UFSM), como requisito parcial para a obtenção do título de **Mestre em Ciência da Computação**.

Aprovado em 20 de Dezembro de 2016:



Roseclea Duarte Medina, Dr.a (UFSM)
(Presidente/Orientador)

Carlos Raniery Paula dos Santos (UFSM)

Érico Marcelo Hoff do Amaral (UNIPAMPA)

Santa Maria, RS
2016

Ficha catalográfica elaborada através do Programa de Geração Automática
da Biblioteca Central da UFSM, com os dados fornecidos pelo(a) autor(a).

Powaczuk, Lucas
OpenVisor - Framework para Redes de Experimentação OpenFlow / Lucas Powaczuk.- 2016.
140 p.; 30 cm

Orientadora: Roseclea Duarte Medina
Dissertação (mestrado) - Universidade Federal de Santa Maria, Centro de Tecnologia, Programa de Pós-Graduação em Informática, RS, 2016

1. Redes Definidas por Software 2. Redes de Experimentação 3. Virtualização de Redes I. Duarte Medina, Roseclea II. Título.

AGRADECIMENTOS

Finalizar um trabalho desta natureza remete-nos a uma retomada de um tempo que abriga inúmeras experiências. Experiências, as quais foram vividas e compartilhadas por diferentes pessoas que fazem parte da minha vida. Portanto, ao concluir esta etapa, gostaria de agradecer a todos que me acompanharam neste percurso:

Inicialmente a minha esposa Vanessa e meu filho Matheus com suas presenças de amor, carinho e alegria, possibilitaram-me o esteio para que este trabalho fosse realizado. O companheirismo e a compreensão que demonstraram durante estes dois anos foram inestimáveis. Um agradecimento especial a minha mãe Ana Carla, do qual teve papel fundamental, auxiliando nas discussões em busca de soluções para os desafios encontrados, mesmo que sem conhecimento técnico sobre os assuntos. Ao meu pai Fernando, irmãos Fernando, Luana e cunhado Hyorran que acompanharam de perto toda esta trajetória, foram essenciais na contribuição dos momentos de descontração. Todos foram muito importantes, cada um com seu papel individual auxiliaram na totalidade deste árduo trabalho. Obrigado por existirem. Amo muito vocês.

Agradeço também à minha orientadora, professora Roseclea Duarte Medina. Este trabalho encerra um período de intensas aprendizagens. Agradeço pela forma como me acolheste no grupo, especialmente pelo seu comprometimento e confiança depositada em mim. Obrigado pela forma como conduziu este processo.

Aos amigos e membros do GRECA que além de colegas de trabalho, foram meus amigos, Leonardo, Gabriel, Ricardo, Eduardo, Vania, Tassiana e Luis. Agradeço pela disposição e pelas inúmeras discussões e reflexões. Também para o amigo Leonardo pelo companheirismo, ajudas e discussões sobre os desafios encontrados em diversos momentos deste trabalho. Agradeço a todos pelos momentos de convívio neste período.

Enfim, aos demais que, de uma forma ou de outra, contribuíram para a realização deste estudo.

RESUMO

OPENVISOR – FRAMEWORK PARA REDES DE EXPERIMENTAÇÃO OPENFLOW

AUTOR: Lucas Powaczuk
ORIENTADOR: Roseclea Duarte Medina

As redes de experimentação (*testbeds*) baseadas em OpenFlow tem-se constituído em um campo de investigação emergente, tendo em vista a necessidade de criar ambientes de experimentação que viabilizem o desenvolvimento de novas tecnologias sobre infraestruturas de redes reais. A revisão bibliográfica evidenciou que as redes de experimentação existentes, ainda, carecem de mecanismos que garantam aos usuários formas operacionais simplificadas, desacopladas do substrato físico e que sejam resilientes. Neste contexto, a problemática da investigação é: como garantir aos usuários de redes de experimentação *OpenFlow* um ambiente que possibilite criar redes virtuais de baixa complexidade de operação, flexíveis e resiliente a rupturas de enlaces? A hipótese que direcionou o estudo é que através da integração das ferramentas OpenVirteX e FlowVisor e, consequentemente de suas funcionalidades, o framework resultante possibilitaria atingir tal propósito. O OpenVirteX e FlowVisor são hypervisors de rede com funcionalidades distintas onde o primeiro dispõe da utilização de topologias virtuais e arbitrárias, recuperação de falhas de conectividade e controle absoluto. Já o FlowVisor tem sua principal contribuição em fornecer uma ampla flexibilidade na definição das redes virtuais. Logo, o objetivo deste estudo foi desenvolver um framework para redes de experimentação OpenFlow, objetivando proporcionar aos usuários redes virtuais flexíveis, de baixa complexidade de operacionalização, dispondo de controle absoluto e resiliente a falhas. A metodologia do estudo caracteriza-se pelo método hipotético-dedutivo. Os procedimentos aplicados para o desenvolvimento da proposta foram: a criação do contexto da experimentação, testes individuais dos *hypervisors OpenVirteX* e *FlowVisor*, integração das ferramentas, avaliação do *Framework* e, finalmente a análise e discussões dos resultados. O estudo realizado confirmou parte da hipótese norteadora da proposta uma vez que o framework se mostrou: Flexível, ao permitir utilizar quaisquer métricas do cabeçalho OpenFlow para a segmentação das redes virtuais; Baixa complexidade, pois permite utilizar uma topologia virtual e arbitrária composta por um único *switch* virtual correspondendo a totalidade da rede física; Resiliente a falhas de conectividade, pois a ferramenta se mostrou capaz de redefinir a comunicação através de rotas alternativas. No que se refere ao controle absoluto, os resultados refutam a presença dessa funcionalidade. Observou-se que disponibilizar o controle total da rede para o usuário tem o impacto de fragilizar a flexibilidade do ambiente de experimentação.

Palavras-chave: Redes Definidas por Software. Redes de Experimentação. Virtualização de Redes.

ABSTRACT

OPENVISOR – FRAMEWORK PARA AMBIENTES DE EXPERIMENTAÇÃO OPENFLOW COM REDES OVERLAY TOLERANTE À FALHAS

AUTHOR: Lucas Powaczuk
ADVISOR: Roseclea Duarte Medina

OpenFlow-based testbeds have been established as an emerging field of research in order to create experimental environments that enable the development of new technologies on real network infrastructures. The bibliographic review showed that existing experimentation networks still lack mechanisms to guarantee users simplified operational forms, decoupled from the physical substrate and that are resilient. In this context, the research problem is: how to guarantee the users of OpenFlow experimentation networks an environment that allows creating virtual networks with low complexity in operation, flexible and resilient to link failures. The hypothesis that guided the study is that by integrating the tools OpenVirteX and FlowVisor and, consequently of its functionalities, the resulting framework would allow to achieving this purpose. OpenVirteX and FlowVisor are network hypervisors with distinct functionalities where the former has the use of virtual and arbitrary topologies, connectivity failure recovery, and absolute control. The FlowVisor has its main contribution in providing a wide flexibility in the definition of virtual networks. Therefore, the objective of this study was to develop a framework for OpenFlow experimentation networks, aiming to provide flexible virtual networks to users, with low complexity of the operation, having absolute control and resilient to failures. The study methodology is characterized by the hypothetical-deductive method. The procedures used to develop the proposal were: create the experimentation context, individual testing of the OpenVirteX and FlowVisor hypervisors, integration of the tools, evaluation of the framework and, finally, analysis and discussion of the results. The study confirmed some of the guiding hypothesis of the proposal since the framework was: Flexible, allowing to use any metrics of the OpenFlow header for the segmentation of virtual networks; Low complexity, because it allows to use a virtual and arbitrary topology composed of a single virtual switch corresponding to the entire physical network; Resilient to connectivity failures, because the tool was able to redefine the communication through of alternative routes. Regarding absolute control, the results refute the presence of this functionality. It was observed that providing total control of the network to the user has the impact of weakening the flexibility of the experimentation environment.

Keywords: Software Defined Networking. Experimentation Networks. Network Virtualization.

LISTA DE FIGURAS

LISTA DE QUADROS

LISTA DE ABREVIATURAS E SIGLAS

ADVisor	AdVanced FlowVisor
API	Application Programming Interface
CIDR	Classless Internet Domain Routing
CLI	Command Line Interface
DPID	Datapath Identifier
EIGRP	Enhanced Interior Gateway Routing Protocol
FIBRE	Future Internet Brazilian Environment for Experimentation
FV	FlowVisor
GENI	Global Environment for Network Inovation
GUI	Grafical User Interface
HTTP	Hypertext Transfer Protocol
IF	Internet do Futuro
IP	Internet Protocol
JSON	JavaScript Object Notation
LLDP	Link Layer Discovery Protocol
MPLS	Multiple Protocol Label Switching
NAT	Network Address Translation
NITOS	Network Implementation Testbed Using Open Source Plataform
OFELIA	OpenFlow in Europe: Linking Infrastructure and Applications
OSPF	Open Shortest Path First
OVX	OpenVirteX
OVX^R	OpenVirteX Recursivo
REST	Representational State Transfer
RIP	Routing Information Protocol
RPC	Remove Procedure Call
RTP	Real Time Protocol
SDN	Software Defined Networking
SPF	Shortest Path First
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
VLAN	Virtual Local Area Network
vSDN	Virtual Software Defined Networking
WDM	wavelength-division-multiplexing

SUMÁRIO

1 INTRODUÇÃO

As redes de experimentação (*testbeds*) têm-se constituído em um campo de investigação emergente. A mobilização dos estudos direciona-se à criação de ambientes de experimentos para que pesquisadores testem e desenvolvam novas tecnologias sobre infraestruturas de redes reais, obtendo resultados mais significativos.

Tal empreendimento decorre da constatação de que a infraestrutura atual da Internet encontra-se ossificada[1] (MCKEOWN et al., 2008), atingindo um nível de saturação que a torna pouco flexível (GUEDES, 2012). Segundo (BANNIZA, 2009) a possibilidade de inovações tornou-se restrita e (FARIAS et al., 2011) acreditam que uma das soluções efetivas aos problemas existentes seja o reprojeto da atual arquitetura da Internet. No entanto, o desenvolvimento de uma nova arquitetura para a Internet é um trabalho complexo e delicado, sendo um dos principais obstáculos a validação das propostas (FARIAS et al., 2011), em virtude da inviabilidade de testes em infraestruturas reais em produção de modo a obter resultados reais e mais significativos.

Neste sentido, as redes de experimentação apresentam como desafio o suporte a virtualização de redes de modo a possibilitar múltiplas redes lógicas operando simultaneamente na mesma infraestrutura física, técnica conhecida como sobreposição de redes (*overlay*) e possibilite a programabilidade por parte dos usuários (FARIAS et al., 2011).

Uma das soluções que vem se destacando no âmbito de redes programáveis e virtualizadas é o *framework OpenFlow* (MCKEOWN et al., 2008) através do paradigma de Redes Definidas por *Software* (*Software Defined Networking* – SDN). O *OpenFlow* possibilita utilizar a rede de produção para testar seus protocolos e arquiteturas experimentais, de forma isolada do tráfego de produção. Além disso, outra característica é a virtualização de rede. A utilização de técnicas de virtualização no contexto de redes com o *framework OpenFlow* permite que múltiplas redes virtuais compartilhem o mesmo substrato físico, através do uso de switches, roteadores e *enlaces* virtuais (CHOWDHURY; BOUTABA, 2010).

1 A Internet desde sua criação, passou por inúmeras mudanças para viabilizar seu funcionamento. Após os anos 90, com o aumento expressivo da utilização da rede, surgiram inúmeros problemas evidenciando limitações na arquitetura TCP/IP. No entanto, ocorreram somente ajustes paliativos necessários para atender as novas demandas e os novos requisitos, como é o caso do *Classless Internet Domain Routing* (*CIDR*), *Network Address Translation* (*NAT*), Serviços Integrados (*Intserv*), Serviços Diferenciados (*Diffserv*), IP Seguro, IP Móvel, *firewalls* e filtros de spam, dentre outros (FARIAS et al., 2011), sem modificar sua estrutura básica. A pesar das diversas adaptações, ainda se utiliza a mesma arquitetura do projeto inicial, resultando no surgimento de inúmeras limitações no modelo TCP/IP.

Atualmente existem várias redes de experimentação baseadas em *OpenFlow* em funcionamento, como o projeto GENI (GENI, 2011), FIBRE (FIBRE, 2011), OFELIA (OFELIA, 2011), entre outras. A grande maioria dos *testbeds OpenFlow* utilizam o *FlowVisor* (FV) (SHERWOOD et al., 2009), que é um *hypervisor* responsável pela virtualização da rede, permitindo que diversas redes lógicas, operem simultaneamente e independentemente sobre a mesma infraestrutura física. No entanto, o FV possui algumas fragilidades (*e. g.*, falta de suporte para topologias virtuais, compartilhamento do flowspace e sem sistema de recuperação de falhas) fazendo com que diversos ambientes de experimentação busquem alternativas, indicando que ele não continuará com o monopólio das redes de experimentação (ELLIOT, 2015).

Diante disso, começaram a surgir propostas de *hypervisors* com o intuito de corrigir as limitações existentes no FV. É o caso do *FlowSpace Firewall*[2], *ADVisor* (SALVADORI et al., 2011), *VeRTIGO* (CORIN et al., 2012) e *Double-FlowVisors* (YIN et al., 2013), todos criados a partir do FV. No entanto, nenhum destes solucionaram a totalidade de fragilidades existentes no FV.

Diante deste contexto, o *hypervisor OpenVirteX* (OVX) (AL-SHABIBI et al., 2014) surgiu com o intuito de corrigir as fragilidades. Diferente do FV, o OVX virtualiza a rede e proporciona redes virtuais com controle do *flowspace* completo, com topologias arbitrárias e resilientes a falhas físicas, possibilitando para cada operador a abstração dos dispositivos físicos e um ambiente com *failover* nativo, ou seja, sem a necessidade de programar soluções próprias para falhas de conectividade. Entretanto, apesar de sanar as fragilidades encontradas no FV, o OVX também possui suas limitações, cuja a principal delas é solucionada pelo FV. O OVX é incapaz de criar redes virtuais por métricas de identificação de fluxos, não permitindo separar redes por tráfegos distintos. Consequentemente a segmentação das redes pelo OVX não é tão flexível quanto a desempenhada pelo FV.

Considerando este contexto, este estudo tem por objetivo desenvolver um *framework* para ser incorporado em redes de experimentação *OpenFlow.* Acredita-se que por meio da integração das ferramentas *OpenVirteX* e *FlowVisor*, é possível viabilizar aos usuários de ambientes de experimentação, redes virtuais com topologia virtual e arbitrária de fácil manejo, flexíveis e ainda resilientes, com recuperação automática de falhas de conectividade entre enlaces físicos.

2 FlowSpace firewall, GlobalNOC – FSFW: FlowSpace Firewall, URL: Acesso dia 23/05/2016).

1.1 MOTIVAÇÃO

Garantir aos usuários de redes de experimentação OpenFlow um ambiente que possibilite criar redes virtuais de baixa complexidade de operação, flexíveis e resilientes a rupturas de enlaces é a motivação central deste trabalho. Considera-se de extrema relevância o desenvolvimento de ferramentas que sejam promotoras de espaços de experimentação que viabilizem aos usuários, ambientes e infraestrutura reais, capazes de garantir o desenvolvimento de testes e avaliações cada vez mais aprimorados e adequados às necessidades emergentes.

1.2 PROBLEMA DE PESQUISA

Como garantir aos usuários de redes de experimentação *OpenFlow* um ambiente que possibilite criar redes virtuais de baixa complexidade de operação, flexíveis e resiliente a rupturas de *enlaces*?

1.3 HIPÓTESE

Por meio da integração das ferramentas *OpenVirteX* e *FlowVisor,* é possível viabilizar aos usuários de ambientes de experimentação, redes virtuais com topologia virtual e arbitrária de fácil manejo, flexíveis e ainda resilientes, com recuperação automática de falhas de conectividade entre enlaces físicos.

1.4 OBJETIVOS

O objetivo geral deste projeto é desenvolver um *framework* para redes de experimentação *OpenFlow*, através da integração dos *hypervisors OpenVirteX* e *FlowVisor*, proporcionando aos experimentadores redes virtuais flexíveis, com topologia arbitrária e resiliente a falhas físicas.

Para contemplar o objetivo proposto, definiu-se alguns objetivos específicos:

- Identificar as redes de experimentação *OpenFlow* existentes reconhecendo seus modos de funcionamento e características;

- Reconhecer os métodos de virtualização de redes utilizados em redes de experimentação *OpenFlow*;
- Mapear as ferramentas provedoras de virtualização de redes *OpenFlow*;
- Identificar o modo de funcionamento e vantagens da ferramenta *FlowVisor*;
- Identificar o modo de funcionamento e vantagens da ferramenta *OpenVirteX*;
- Identificar as vantagens e limitações de integrar as ferramentas *OpenVirteX* e *FlowVisor*;
- Identificar os requisitos necessários para a integração das ferramentas *OpenVirteX* e *FlowVisor*;
- Integrar as ferramentas *OpenVirteX* e *FlowVisor*;
- Comparar o *framework* proposto com o *FlowVisor* e o *OpenVirtex*;
- Testar as funcionalidades da solução proposta, verificando as vantagens e as limitações;
- Comparar o *framework* proposto com os demais usados nas redes de experimentação *OpenFlow* existentes, nos quesitos usabilidade, desempenho, tempo de convergência e funcionalidades;

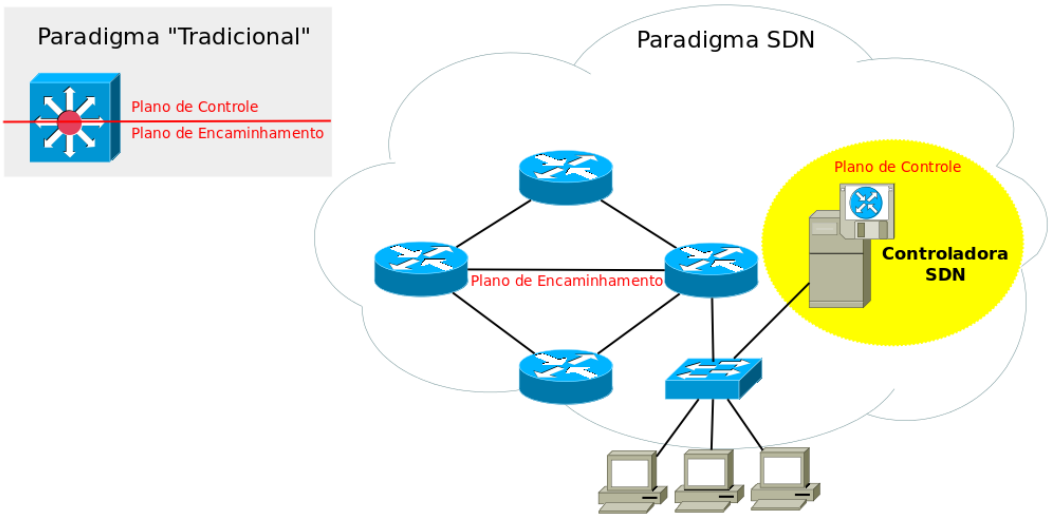
Tendo em vista o problema da investigação e os objetivos almejados, o estudo realizado exigiu a revisão nas bases de dados acerca dos seguintes temas: Redes Definidas por *Software*, Redes de Experimentação *OpenFlow* e Virtualização em Redes *OpenFlow*. Este estudo fundamentou a análise entre as diferentes redes de experimentação *OpenFlow* existentes, identificando suas características e modos de funcionamento. Destacou-se neste processo, a relevância da identificação dos trabalhos correlatos, tendo em vista que o mapeamento realizado corroborou para a consolidação da proposta do estudo. Finalizado este processo passou-se a implementação da proposta e a realização de testes direcionados a avaliar a eficácia do modelo proposto.

2 REDES DEFINIDAS POR SOFTWARE

Redes Definidas por Software (*Software Defined Networking* - SDN) caracterizam-se como um contexto emergente sobre redes de computadores, evidenciando-se uma mudança paradigmática. Seu caráter revolucionário propõe transformações nas formas de operações conhecidas da atualidade, especialmente por introduzir a habilidade de programação dentro da rede (KREUTZ et al. 2015).

Pode-se dizer que uma SDN é caracterizada pela existência de um sistema de controle que pode gerenciar o mecanismo de encaminhamento dos elementos de comutação da rede (GUEDES et al., 2012). Diferente da arquitetura de rede tradicional, no paradigma SDN o plano de controle e de dados dos equipamentos são separados, onde o primeiro é responsável por definir as políticas de operação da rede, processando os fluxos e determinando as diretrizes de controle das mensagens. Já o plano de dados é responsável por encaminhar os pacotes de acordo com as regras definidas pelo plano de controle. Além da separação dos planos, a inteligência da rede (plano de controle) é centralizada em uma entidade gerenciadora, chamada de controlador, conforme ilustra a Figura 1. Os comutadores, encarregam-se exclusivamente no encaminhamento dos fluxos (plano de dados) e recebem instruções de operação do controlador.

Figura 1 – Paradigma SDN.



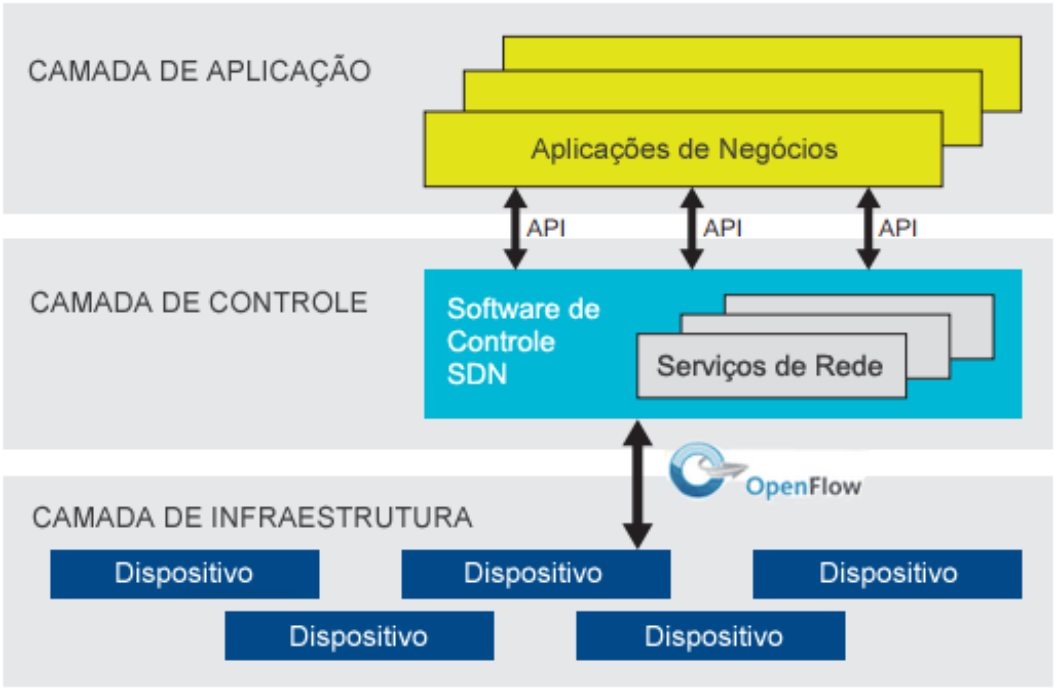
Fonte: Adaptado de (BRITO, 2013).

As redes definidas por *software* são definidas por (KREUTZ et al. 2015) da seguinte maneira:

- Os planos de controle e de dados são desacoplados. A funcionalidade de controle é removida dos equipamentos de rede e se tornam apenas comutadores simples de encaminhamento de pacotes;
- Decisões de encaminhamento são baseadas em fluxo ao invés de destinos. Um fluxo é basicamente definido por um conjunto de campos com valores agindo como filtros e um conjunto de ações (instruções). No contexto de SDN e *OpenFlow*, um fluxo é uma sequência de pacotes entre uma ou N fontes e um ou N destinos. A abstração desse fluxo permite que o comportamento de diferentes tipos de dispositivos, como roteadores, switches e *firewalls*, seja unificado;
- A lógica de controle é movida para uma entidade externa, o controlador SDN. O controlador fornece abstrações para facilitar a programação de dispositivos de encaminhamento de pacotes baseado em uma visão centralizada da rede;
- A rede é programada através de aplicações de *software* rodando sobre o controlador SDN que interage com os dispositivos do plano de dados.

Para estabelecer a comunicação entre o plano de controle e o plano de encaminhamento foi preciso criar e padronizar uma interface de programação. De forma mais específica, os elementos de comutação exportam uma interface de programação que permite ao controlador definir e alterar as entradas da tabela de roteamento do comutador (GUEDES et al., 2012). A figura 2 representa a interação entre as diferentes camadas que compõem a arquitetura SDN.

Figura 2 – Arquitetura de Redes Definidas por *Software*.



Fonte: Adaptado de (ONF, 2012).

Como resultado, a camada de infraestrutura refere-se aos dispositivos da rede. A interface de comunicação entre os comutadores e o controlador é realizada por meio de um protocolo de padrão aberto, denominado *OpenFlow*. A camada de controle possui a visão global da rede. Já a camada de aplicação permite criar aplicações, por meio das APIs da camada de controle. A comunicação entre o controlador e as aplicações da rede é realizada por meio de uma interface de alto nível, como por exemplo, *REST API* ou *JSON-RPC*.

2.1 OPENFLOW

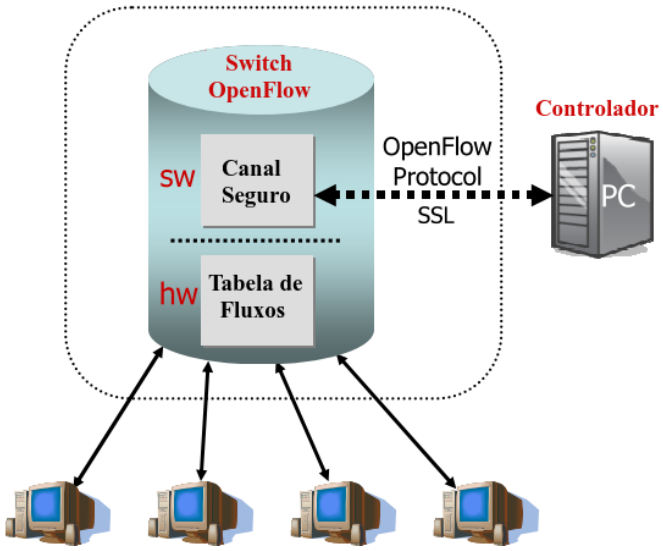
O surgimento do protocolo *OpenFlow* (MCKEOWN et al., 2008) possibilitou programar o comportamento dos elementos de comutação da rede, tornando-se uma ótima solução para o desenvolvimento de novas aplicações não se limitando exclusivamente as implementações dos fabricantes. Uma de suas características é possibilitar testar novos protocolos em ambientes reais, já que é praticamente inviável testá-los em uma rede em

produção pois o risco de comprometer o tráfego original é alto. Logo o *OpenFlow* proporciona uma forma de testar experimentos juntamente na rede de produção (MCKEOWN et al., 2008), pois possibilita a manipulação das tabelas de encaminhamento dos comutadores, no plano de dados (*datapath*), por um elemento remoto denominado controlador.

Conforme (ROTHENBERG et al., 2011), *OpenFlow* define um protocolo-padrão para determinar as ações de encaminhamento de pacotes em dispositivos de rede, como, por exemplo, comutadores, roteadores e pontos de acesso sem fio. As regras e ações instaladas nos comutadores são responsabilidades do controlador. Seu funcionamento é baseado no conceito de fluxos onde um fluxo é constituído por um conjunto de pacotes que possuem a mesma combinação de informações dos campos do cabeçalho.

A Figura 3 apresenta a arquitetura de um comutador *OpenFlow*, que consiste nos seguintes elementos: uma tabela de fluxos (*flowtable*), que possui as políticas de operação e encaminhamento dos pacotes e com base em ações os *switchs* encaminham as mensagens; um canal seguro para comunicação entre os dispositivos e o controlador externo; o protocolo *OpenFlow*, que provê uma forma padronizada e aberta do controlador se comunicar com o *switch*; e por fim o controlador, que é responsável por gerenciar a tabela de fluxo dos comutadores *OpenFlow*.

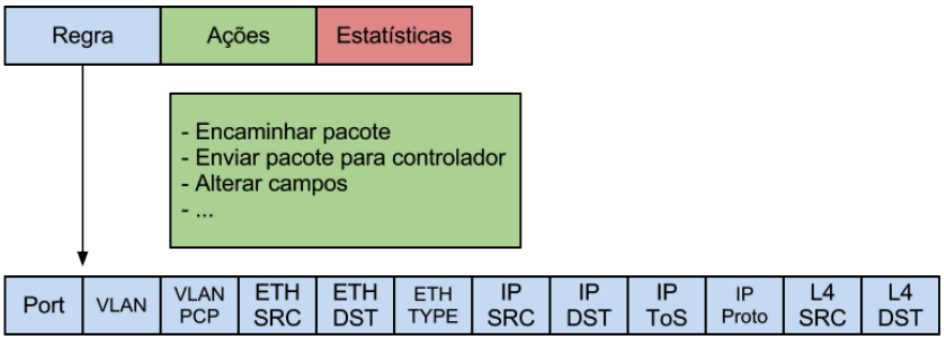
Figura 3 – Arquitetura do *switch OpenFlow*.



Fonte: Adaptado de (MCKEOWN et al., 2008).

Todos os pacotes processados pelo *switch* são comparados com cada entrada na tabela de fluxos. De acordo com (ROTHENBERG et al., 2011), cada entrada na tabela de fluxos do *hardware* de rede consiste em regra, ações e contadores, conforme a Figura 4. A regra é formada com base na definição do valor de um ou mais campos do cabeçalho do pacote. Logo, caso exista uma regra correspondente ao pacote em análise pelo *switch*, então aplica-se as ações relacionadas a esse fluxo. Caso não exista uma entrada na tabela de fluxos para aquele pacote, ele é encaminhado para o controlador, para determinar como será processado. Os contadores são utilizados para a elaboração de estatísticas de fluxos e tráfego.

Figura 4 – Exemplo de um fluxo em uma tabela de fluxos *OpenFlow*.



Fonte: Adaptado de (GUEDES et al., 2012).

3 REDES DE EXPERIMENTAÇÃO OPENFLOW

Na comunidade de pesquisa em redes, há muito tempo atenta ao crescimento de problemas da Internet, aumentou-se o interesse em estudar os desafios da Internet do Futuro e, com isso, modelar a arquitetura que conduzirá a uma nova geração da Internet. Estas novas propostas e especulações teóricas deveriam ser suportadas por uma infraestrutura real de rede experimental e testadas em ambientes de larga escala (FARIAS et al., 2011).

Estas infraestruturas experimentais desempenhariam o papel de rede para experimentação ou ambiente de testes (*testbed*), possibilitando as validações necessárias para a prova de conceitos de novas tecnologias, arquiteturas, protocolos e serviços. Uma coexistência com a rede de tráfego de produção é essencial para observação e captura de certos aspectos e fenômenos perceptíveis apenas em instalações operacionais e assim permitir que sejam avaliados os seus impactos sobre a sociedade e a economia (FARIAS et al., 2011).

Nesta direção, apresenta-se as principais redes de experimentações com suporte ao *OpenFlow*, em operação na atualidade, apresentando suas arquiteturas e características referentes aos métodos de virtualização utilizadas.

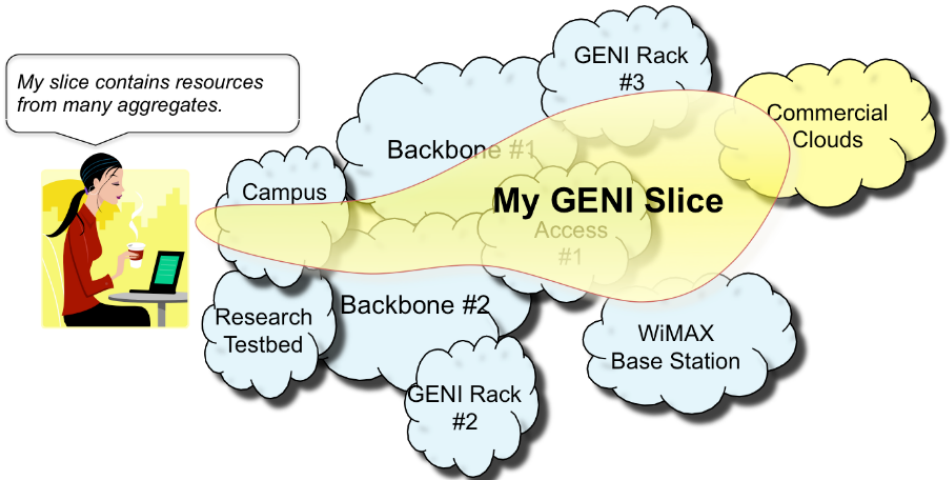
3.1 GENI

O GENI (*Global Environment for Network Inovation*) é uma rede de experimentação norte-americana patrocinada pela *National Science Foundation* (NSF) com o intuito de investigar soluções para a ossificação da Internet (BERMAN et al., 2014). Os autores (FARIAS et al., 2011) definem o GENI como um conjunto de infraestruturas de redes para experimentação, dos mais variados modelos tais como: sem fio, óptico, elétrico e *OpenFlow*. Seu objetivo é montar um grande *testbed* para fornecer um ambiente destinado para experimentações em redes de computadores, com foco em validar novas possibilidades para Internet do Futuro.

SDN é um aspecto profundamente integrado a infraestrutura do projeto GENI. Os conceitos-chave do projeto são fornecer flexibilidade aos operadores, através da capacidade de fatiamento e a programabilidade da rede (BERMAN et al., 2014). Os experimentadores, nome designado para os usuários que utilizam os *testbeds*, podem escolher quais agregados (dispositivos, *hosts*, recursos da rede, etc) serão incorporados a sua rede. Cada rede criada e destinada aos usuários são chamadas de *slices*. Já a programabilidade permite aos

pesquisadores o controle quase total dos aspectos do experimento, incluindo rede, *storage* e recursos computacionais (BERMAN et al., 2014). A Figura 5 ilustra o processo de definição de um *slice*.

Figura 5 – Criando um slice na infraestrutura GENI.



Fonte: http://groups.geni.net/geni/wiki/GENIConcepts.

O GENI é permite que várias redes e experimentadores operem e realizem experimentos simultaneamente sobre a mesma infraestrutura física através da utilização de *slices*. Para (BERMAN et al., 2014) um *slice* GENI significa:

- Uma unidade de isolamento para os experimentos. Um experimento do GENI vive em uma fatia. Somente experimentadores que são membros de um *slice* podem fazer alterações naquele experimento/*slice*.
- Um contêiner de recursos usados em um experimento. Experimentadores do GENI adicionam recursos (recursos computacionais, *links* de rede, etc.) aos *slices* e executam experimentos que usam esses recursos. Um experimento só pode utilizar recursos em sua fatia.

O projeto GENI permite que cada experimentador defina qual controlador utilizar sobre seu slice. A arquitetura do projeto GENI utiliza o *hypervisor FlowVisor* através do gerenciador de recursos FOAM (*FlowVisor OpenFlow Aggregate Manager*), que é uma ferramenta que tem a função de gerenciar os *slices* e a alocação de recursos *OpenFlow* dos experimentadores (MUSSMAN, 2012).

3.2 OFELIA

O OFELIA (*OpenFlow in Europe: Linking Infrastructure and Applications*) é uma rede de experimentação Europeia composta por dez *testbeds* individuais, chamados de ilhas, de parceiros do projeto (SUÑÉ et al., 2014). Sua criação objetivou o desenvolvimento em pesquisas sobre implementações práticas de Redes Definidas por *Software* e *OpenFlow*. Atualmente o *testbed* OFELIA oferece uma infraestrutura *OpenFlow* diversificada e poderosa que permite experimentos multicamada e de tecnologias diversas de rede. O OFELIA foi concluído em 2011 e permanece operacional e de uso livre para os pesquisadores interessados (SU et al., 2014).

A infraestrutura do OFELIA possui dez ilhas *OpenFlow* que compõem a rede de experimentação, conforme é visto na Figura 6 a localização das ilhas, que encontram-se nos seguintes países: Alemanha, Bélgica, Suíça, Espanha, Reino Unido, Itália e Brasil.

Figura 6 – Ilhas *OpenFlow* do projeto OFELIA.

Fonte: http://www.fp7-ofelia.eu/ofelia-facility-and-islands/.

Através das ilhas, o *testbed* OFELIA possui um conjunto diversificado de equipamentos que compõem a rede de experimentação, possibilitando aos experimentadores fazer testes, por exemplo, sobre dispositivos ópticos, equipamentos sem fio entre outros. O OFELIA também utiliza o conceito de *slices* para isolar os experimentos dentro da rede de experimentação e para isso é utilizado o *FlowVisor* (SUÑÉ et al., 2014), através de diferentes *frameworks* de controle. Além do FV, encontram-se estudos que demonstram a utilização do

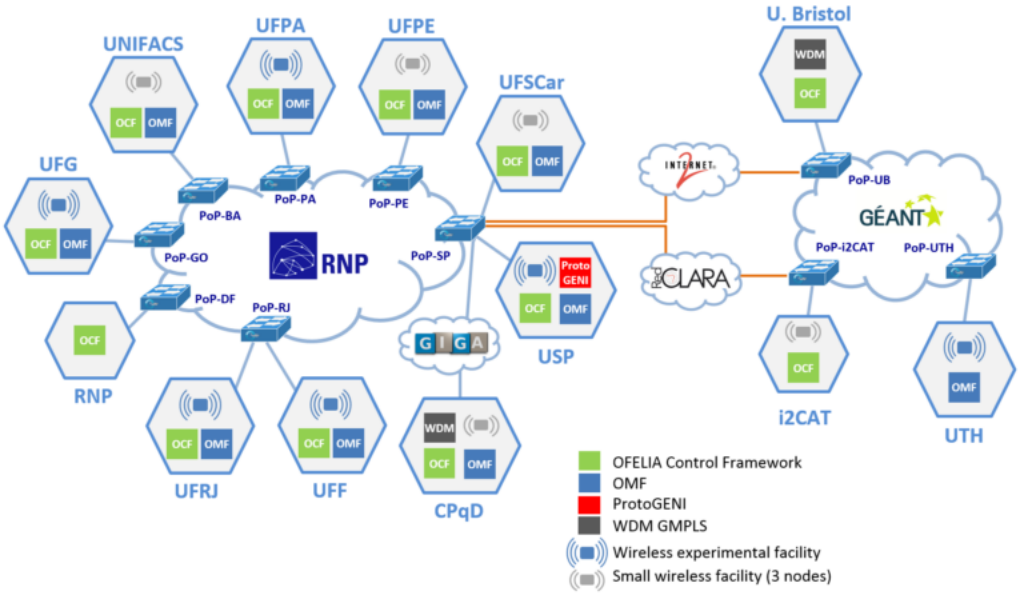
ADVisor (SALVADORI et al., 2011) e do *VeRTIGO* (CORIN et al., 2012) sobre o *testbed* OFELIA.

A política de fatiamento (*slicing*) adotada no *testbed* é por VLAN, depois de ser testado vários outros esquemas de segmentação, por exemplo por IP, tipo de aplicação, etc, pois as demais formas ocasionavam no problema de limitações de regras de fluxos na memória TCAM (*Ternary Content Addressable Memory*) nos switches. Logo, para cada *slice* é atribuído um identificador de VLAN, garantindo um isolamento de camada 2. Para testes locais, envolvendo ilhas individuais é possível usar outras técnicas de fatiamento suportadas pelo FV.

3.3 FIBRE

O FIBRE (*Future Internet Brazilian Environment for Experimentation*) é uma rede de experimentação criada em 2011 com objetivo de desenvolver pesquisas no Brasil sobre Internet do Futuro (SALMITO et al., 2014). O FIBRE é uma rede de experimentação composta por 13 *testbeds* individuais, chamadas de ilhas FIBRE. No Brasil existem 10 ilhas em universidades federais e outras 3 ilhas na Europa. A Figura 7 apresenta a visão global do *testbed* FIBRE.

Figura 7 – Visão global das ilhas do FIBRE.



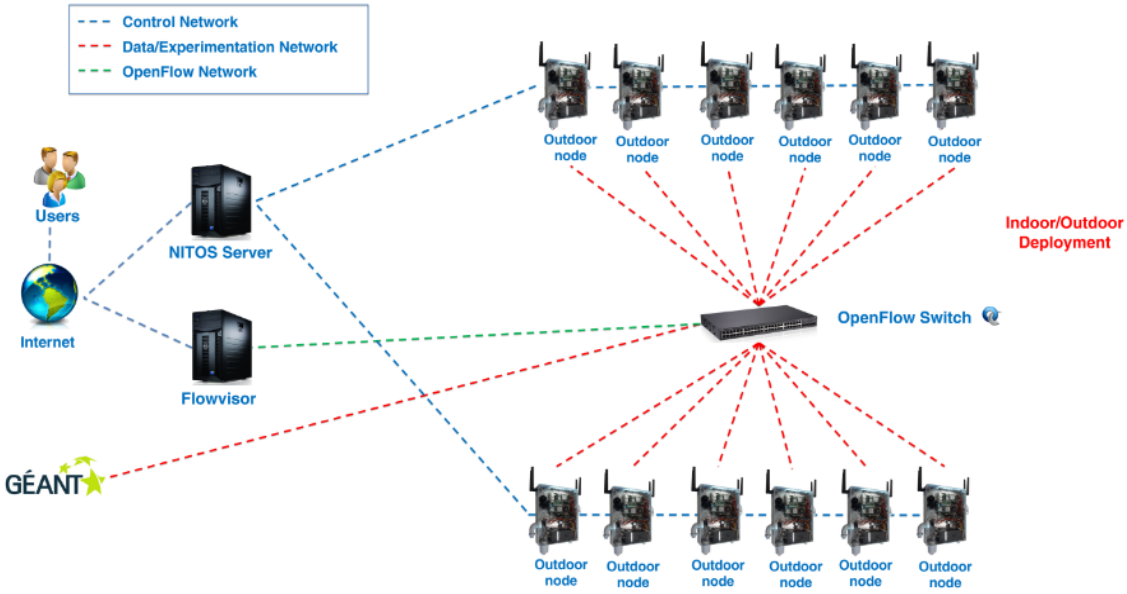
Fonte: (SALMITO et al., 2014).

Cada ilha possui um conjunto de dispositivos *OpenFlow*, bem como servidores de computação e armazenamento e um conjunto de nós sem fios virtualizados. A partir disso, os experimentadores podem definir quais recursos, de quais ilhas, irão compor seus *slices*. Assim como os demais *testbeds*, o FIBRE utiliza o *FlowVisor* através do *framework* de controle OFELIA *Control Framework* (OCF), de modo a virtualizar os recursos da rede, como os nós de processamento, dispositivos e redes (SALMITO et al., 2014). A política de *slicing* utilizada é semelhante ao do *testbed* OFELIA, onde para *slice* é atribuído um identificador de VLAN, e para experimentos locais é possível usar outras técnicas de fatiamento suportadas pelo FV.

3.4 NITOS

O NITOS (*Network Implementation Testbed Using Open Source Plataform*) é um ambiente de experimentação desenvolvido pelo grupo NITLAB (*Network Implementation Testbed Laboratory*) que possibilita aos usuários fazeem testes com nós sem fio em uma rede *OpenFlow*. O *testbed* foi projetado com o intuito de reproduzir experimentos práticos e servir como ambiente de apoio para avaliação de protocolos e aplicações. O NITOS é um *testbed* com menor escala se comparado com o GENI, OFELIA e FIBRE. É composto predominantemente por nós de processamento sem fio, além de switches *OpenFlow*, conforme ilustra a Figura 8. Utiliza-se o *FlowVisor* para a virtualização e o fatiamento da rede (PECHLIVANIDOU et al., 2014).

Figura 8 – Arquitetura do testbed NITOS.

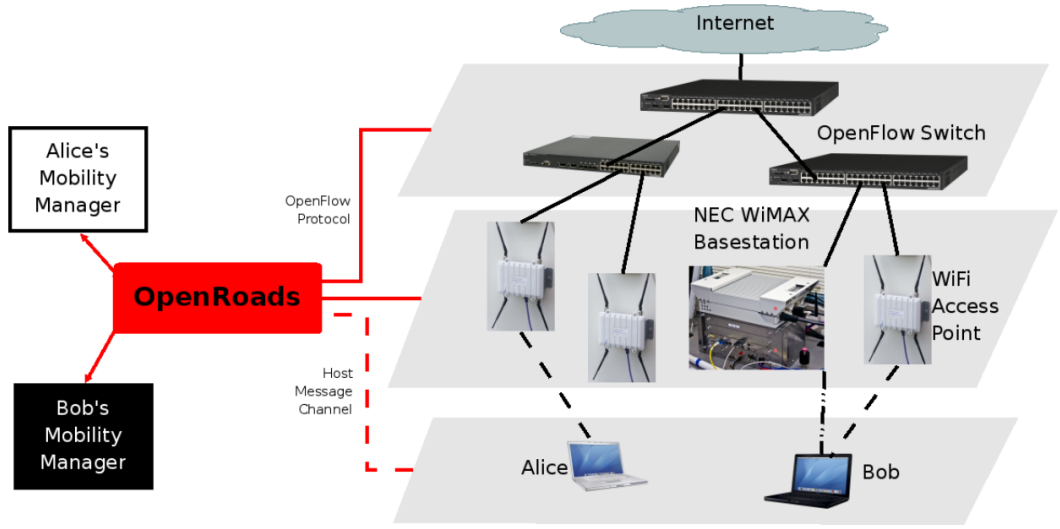


Fonte: http://nitlab.inf.uth.gr/NITlab/nitos/openflow.

3.5 STANFORD – OPENROADS

A universidade de *Stanford*, criadora do *OpenFlow* em 2008 também desenvolveu uma rede de experimentação sobre sua infraestrutura de produção (ELLIOTT, 2015). A infraestrutura do *testbed* é dividido em duas partes: uma rede Wi-Fi *OpenFlow*, denominada *ofwifi* e outra de produção com diversos equipamentos. Através do *testbed ofwifi* foi criado o *OpenRoads* que é uma plataforma de experimentação para desenvolvimento de pesquisas em redes *wireless*, que permite fatiar a rede e ainda, conduzir experimentos simultaneamente, conforme ilustra a Figura 9. O *OpenRoads*, utiliza o *FlowVisor* para dispor o *overlay* de redes e permitir o compartilhamento do controle da rede por múltiplos pesquisadores, sobre a mesma infraestrutura física (YAP et al., 2009).

Figura 9 – Plataforma OpenRoads para pesquisas em redes wireless.



Fonte: (YAP et al., 2009)

3.6 MCNC

MCNC é uma organização sem fins lucrativos nos Estados Unidos que cria e gerencia infraestruturas de rede para a educação, pesquisa e outras instituições importantes (ELLIOT, 2015). Em novembro de 2013, MCNC criou um grupo de trabalho para desenvolver e implementar uma infraestrutura de experimentação *OpenFlow* entre cinco universidades nos Estados Unidos, Universidade de *Duke*, Universidade da Carolina do Norte em *Chapel Hill* (*University of North Carolina at Chapel Hill*), Universidade do Estado da Carolina do Norte (*North Carolina State University*) e do RENCI (*Renaissance Computing Institute*). Para fornecer virtualização e fatiamento da rede, utiliza-se o *FlowVisor* e o *FlowSpace Firewall* (ELLIOT, 2015), ferramenta apresentada na seção 4.1.2.

3.7 AMBIENTES DE EXPERIMENTAÇÃO OPENFLOW: COMPARATIVO

A análise das principais redes de experimentações com suporte ao *OpenFlow*, em operação na atualidade, permite destacar que o *FlowVisor* atualmente é o *hypervisor* mais utilizado para redes *OpenFlow*, utilizado em todos *testbeds* citados neste trabalho. A afirmação pode ser visualizada a partir do quadro comparativo a seguir na tabela 2:

Quadro 1- Comparativo entre os ambientes de experimentação *OpenFlow*.

Ambientes de Experimentação OpenFLow	**Ilhas de experimentação**	**Hypervisor de rede**
GENI	Diversos	FlowVisor
OFELIA	TUB – Alemanha IBBT – Bélgica ETH – Suíça i2CAT – Espanha UNIVBRIS – Reino Unido CNIT – Catania, Itália CNIT – Roma, Itália CREATE-NET – Itália CNIT – Pisa, Itália UFU – Brasil	FlowVisor VeRTIGO ADVisor
FIBRE	UFG UNIFACS UFPA UFPE UFSCar USP CpqD UFF UFRJ RNP	FlowVisor
NITOS	-	FlowVisor
Stanford	-	FlowVisor
MCNC	-	FlowVisor; FlowSpace Firewall

A constatação evidenciada, corrobora com as indicações de (ELLIOT, 2015) quando afirma que a maioria dos *testbeds* e redes de pesquisa com suporte ao *OpenFlow* optam pela utilização do FV em virtude da sua principal funcionalidade: a flexibilidade no *slicing* das redes virtuais, ou seja, na forma como é segmentado as redes virtuais. Contudo, o mesmo autor, enfatiza que em virtude das limitações existentes no FV, diversos *testbeds* com ampla repercussão mundial, como o *GENI*, *OFELIA* e *MCNC* estão em busca de ferramentas alternativas ao *FlowVisor*, indicando que sua preponderância será reduzida, se não sofrer mudanças e/ou melhorias.

4 VIRTUALIZAÇÃO DE REDES OPENFLOW

Redes definidas por *software* abriram espaço para que pesquisadores e desenvolvedores usassem a rede de produção como ambiente de testes. Entretanto, ao se conectar os elementos de comutação de uma rede a um controlador único, a capacidade de se desenvolver e instalar novas aplicações na rede fica restrita ao responsável por aquele controlador. Mesmo que o acesso a esse elemento seja compartilhado entre diversos pesquisadores, ainda há a questão da garantia de não-interferência entre as diversas aplicações (GUEDES et al., 2012).

Analogamente à virtualização computacional, que possibilita o compartilhamento de recursos por múltiplos sistemas virtualizados, a virtualização de rede permite que múltiplas redes heterogêneas compartilhem o mesmo substrato físico (CHOWDHURY; BOUTABA 2010). Em geral, a virtualização de rede permite que várias redes virtuais coexistam na mesma infraestrutura de rede física, possibilitando que cada rede virtual tenha seus próprios recursos da topologia física, trabalhando de forma isolada das demais redes virtuais.

As máquinas virtuais, através da virtualização computacional, tornaram-se muito importante na computação, pois permitem que os aplicativos operem de forma flexível, com seus sistemas operacionais, sem a necessidade de se preocupar com os detalhes específicos da plataforma de computação subjacente. Analogamente, as redes virtuais surgiram com o intuito de permitir esta mesma flexibilidade, ou seja, para que novos serviços de rede possam operar sem se preocupar com os detalhes específicos da infraestrutura física, por exemplo, o *hardware* subjacente da rede (dispositivos, portas e *links*).

A virtualização de redes abstrai a infraestrutura física subjacente e em seguida, cria redes virtuais isoladas, chamados de *slices*. No entanto, existem várias técnicas para criar *slices* ou “fatias” da rede, como por exemplo, *Wavelength-Division-Multiplexing* (WDM), cria fatias na camada física do modelo de referência OSI; as redes locais virtuais (VLANs), que criam fatias na camada de enlace; e *Multiple Protocol Label Switching* (MPLS), que criam fatias de tabelas de encaminhamento em switches (BLENK et al., 2015).

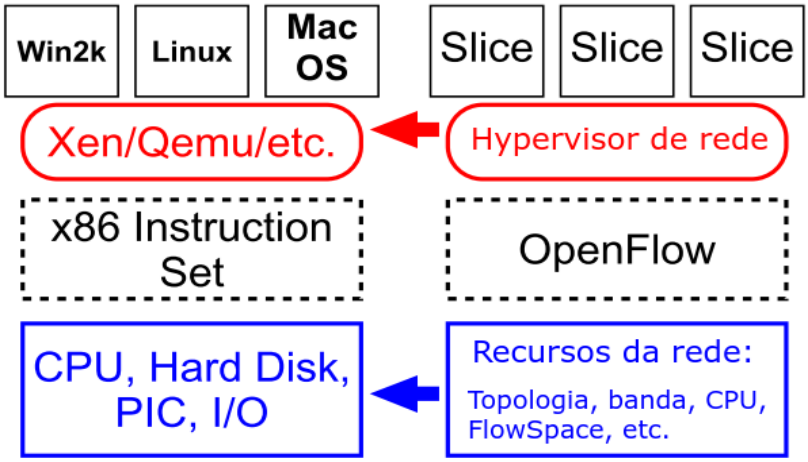
Segundo (BLENK et al., 2015) inicialmente a virtualização de redes era definida como uma fatia (*slice*) de uma determinada infraestrutura de rede sobre uma camada do modelo de referência OSI. Em contraste, a virtualização de redes procura criar *slices* de toda a rede, isto é, para formar redes virtuais completas, sobre todas as camadas da rede. Uma determinada rede virtual deve ter os seus próprios recursos, incluindo o sua própria visão da rede, as suas

próprias fatias de larguras de banda e de suas próprias fatias de recursos da CPU dos switches e tabelas de encaminhamento (SHERWOOD et al., 2009).

As características supracitadas são providas pelo arcabouço *OpenFlow*. Diversos pesquisadores acreditam que a virtualização de redes ganhou novas funcionalidades com Redes Definidas por *Software*. Kreutz et al., (2015) afirmam que a virtualização de redes ganhou um novo impulso com o advento de SDN. Farias et al., (2011) dizem que a utilização de técnicas de virtualização sobre o contexto de redes, com o *framework OpenFlow*, tem se destacado como uma excelente alternativa para a execução de experimentos de novas arquiteturas diretamente sobre ambientes de produção. Blenk et al., (2015) acredita que as redes virtuais ou *slices*, podem acelerar o desenvolvimento de novos conceitos de redes através da criação de ambientes de experimentação. Chowdhury; Boutaba, (2010) acreditam que a virtualização de redes *OpenFlow* ajudará a minimizar a ossificação da Internet.

A virtualização em redes *OpenFlow* deve possuir uma camada de abstração de *hardware*, assim como ocorre através dos *hypervisors* na virtualização computacional, como exemplo o *XEN*, *VirtualBox* ou *VMware*. Se tratando de SDN, os *hypervisors* de rede agem como um *proxy* transparente entre os dispositivos *OpenFlow* e os controladores, oferecendo redes virtuais para cada um deles, conforme ilustra a Figura 10. Os autores (BLENK et al., 2015) afirmam que uma das razões pelo qual as SDNs possibilitam uma nova perspectiva de virtualização de redes é porque é possível inserir um *hypervisor* entre a infraestrutura física da rede e o plano de controle.

Figura 10 – Comparação entre a virtualização computacional e de rede.



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

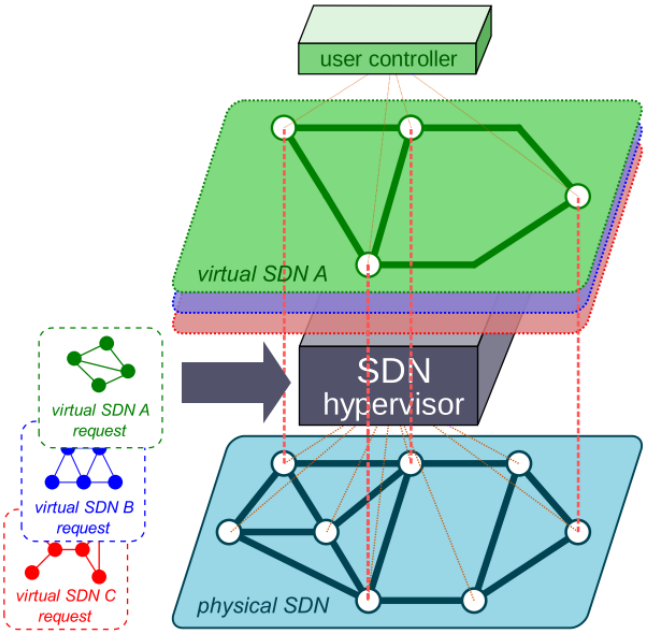
Portanto, os *hypervisors* gerenciam as comunicações entre o plano de dados (comutadores) e o plano de controle (controladores) da rede, possibilitando a abstração da infraestrutura da rede para as camadas superiores (aplicações de rede), desta forma, permitindo que múltiplas redes virtuais compartilhem o mesmo *hardware*. Ainda, no contexto de rede virtual, um *hypervisor* também possui como funcionalidade o monitoramento das redes virtuais e a alocação de recursos de rede, como por exemplo a capacidade do link e a comutação de pacotes individualmente para cada fatia virtual (KREUTZ et al., 2015).

4.1 HYPERVISORS DE REDES OPENFLOW

Os *hypervisors* de redes *OpenFlow* são responsáveis por fornecer a virtualização, permitindo que múltiplas redes virtuais compartilhem o mesmo *hardware*. A função dos *hypervisors OpenFlow* é abstrair a rede física e criar redes virtuais isoladas para ser controladas por diferentes controladores (BLENK et al., 2015). Portanto, redes virtuais viabilizam uma maior flexibilidade, permitindo que novos serviços de rede possam ser implementados, abstendo-se de preocupações com detalhes específicos da rede subjacente (CHOWDHURY; BOUTABA, 2009).

Usando o protocolo *OpenFlow*, um *hypervisor* pode estabelecer múltiplas redes SDN virtuais (*virtual* SDNs - vSDNs) na mesma rede física onde cada vSDN é controlada por um usuário (chamado de *tenant*) através do controlador, conforme a Figura 11 apresenta. Cada *tenant* pode operar sua própria vSDN utilizando seu próprio sistema operacional de rede, sem que esse tráfego interfira as demais vSDNs (BLENK et al., 2015). Diferentes serviços, por exemplo, voz e vídeo podem ser executado em fatias isoladas virtuais para melhorar a qualidade de serviço e desempenho geral da rede (ANDERSON et al., 2005).

Figura 11 – Visão geral de um Hypervisor SDN.

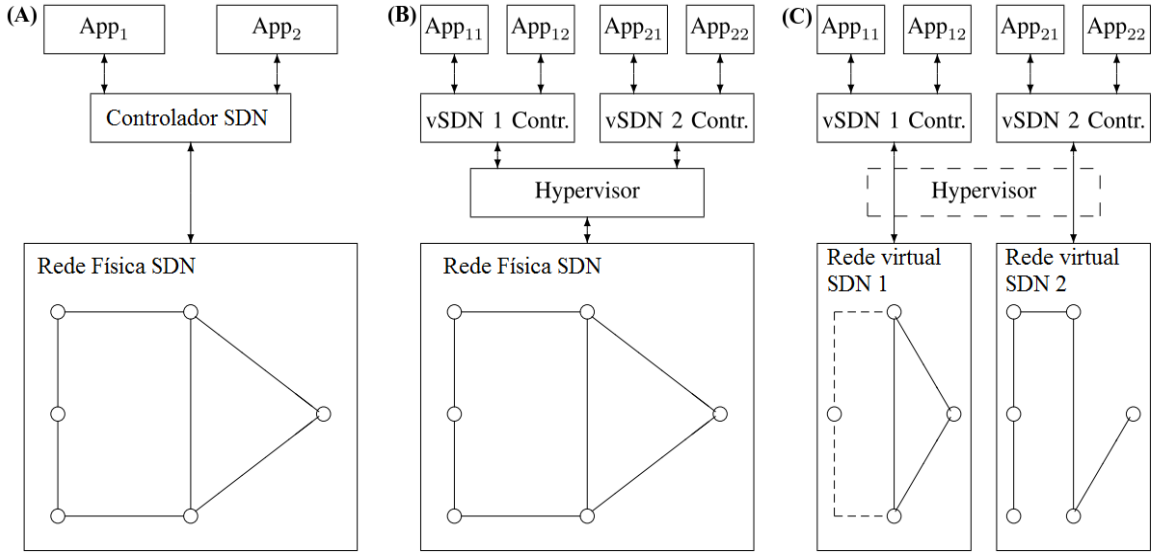


Fonte: Adaptado de (BOZAKOV; PAPADIMITRIOU, 2014).

Através da técnica de fatiar os recursos da rede, cada vSDN criada pelo *hypervisor* também é chamada de *slice*. Embora as definições de *slices* variam de acordo com o *hypervisor*, (GUTZ et al., 2012) definem que a maioria das fatias consistem no seguinte: 1) uma topologia contendo um conjunto da rede e recursos computacionais; 2) um mapeamento entre esses recursos e infraestrutura física subjacente; e 3) uma agregação de políticas de acesso que definem quais fluxos de rede pode entrar para cada *slice*.

A Figura 12 apresenta diferentes perspectivas sobre a virtualização de redes SDN. Em (a) é ilustrado uma rede SDN sem virtualização, onde o controlador define as comunicações entre as aplicações e a rede física. Já (b) e (c) é apresentado uma rede SDN com recurso de virtualização.

Figura 12 – Diferentes perspectivas em vSDNs.



Fonte: Adaptado de (BLENK et al. 2015).

Em (b) é representado a visão do *hypervisor,* que tem a perspectiva da totalidade da rede, no qual interage diretamente com os comutadores como também com os controladores das vSDN 1 e vSDN 2. Já em (c) é apresentado a visão dos controladores das vSDNs, onde o *hypervisor* passa a percepção de que eles interagem diretamente com o substrato físico de sua rede virtual correspondente, quando na verdade esse papel é desempenhado pelo *hypervisor*. Cada vSDN tem a perspectiva somente dos componentes da sua rede virtual, podendo ter diferentes topologias e níveis de abstração, conforme demonstrado pelas diferentes topologias das vSDNs em (c).

Portanto, nota-se a importância que o *hypervisor* possui em uma infraestrutura SDN com suporte a virtualização. Desta maneira surgem inúmeras possibilidades com a virtualização de redes, como exemplo, criar vSDNs com topologias virtuais e arbitrárias, ou seja, onde o *hypervisor* repassa aos controladores informações de equipamentos e links virtuais, podendo passar aos controladores a perspectivas de topologias totalmente desassociadas do substrato físico. Outro exemplo é disponibilizar vSDNs resilientes a falhas, visto que o *hypervisor* gerencia as comunicações com a infraestrutura de rede, é possível

gerenciar os fluxos e na ocorrência de falhas físicas o *hypervisor* retomar o erro de forma transparente para os controladores.

A partir do uso de virtualização, começaram a surgir novas possibilidades de funcionalidades. Na próxima seção, são apresentados dois diferentes *hypervisors*, com métodos de operação distintos de modo a apresentar seus modos de funcionamento e funcionalidades.

4.1.1 FlowVisor

O *FlowVisor* (FV) foi o primeiro *hypervisor* criado para virtualização e compartilhamento de Redes Definidas por *Software* baseado no protocolo *OpenFlow* (SHERWOOD et al., 2009). Em geral, a principal contribuição do FV foi possibilitar que redes de produção e experimentais possam coexistir simultaneamente sobre a mesma infraestrutura da rede, concentrando-se em mecanismos para isolar o tráfego de rede experimental do tráfego de produção.

O FV é considerado um controlador *OpenFlow* especial, que age como um *proxy* transparente entre dispositivos da rede e os múltiplos controladores. Sua função é virtualizar segmentos da rede, de modo que o mesmo *hardware* (plano de dados) possa ser compartilhado entre múltiplas redes, cada uma com controladores distintos. As mensagens do protocolo *OpenFlow* originadas pelos comutadores e controladores, são interceptadas através do FV. Desta forma os controladores não necessitam de modificações e além disso, o FV intercepta e encaminha as mensagens de forma transparente, tanto para os controladores quanto aos switches *OpenFlow*, fazendo-os acreditar estar se comunicando diretamente sem um intermediário (SHERWOOD et al., 2010).

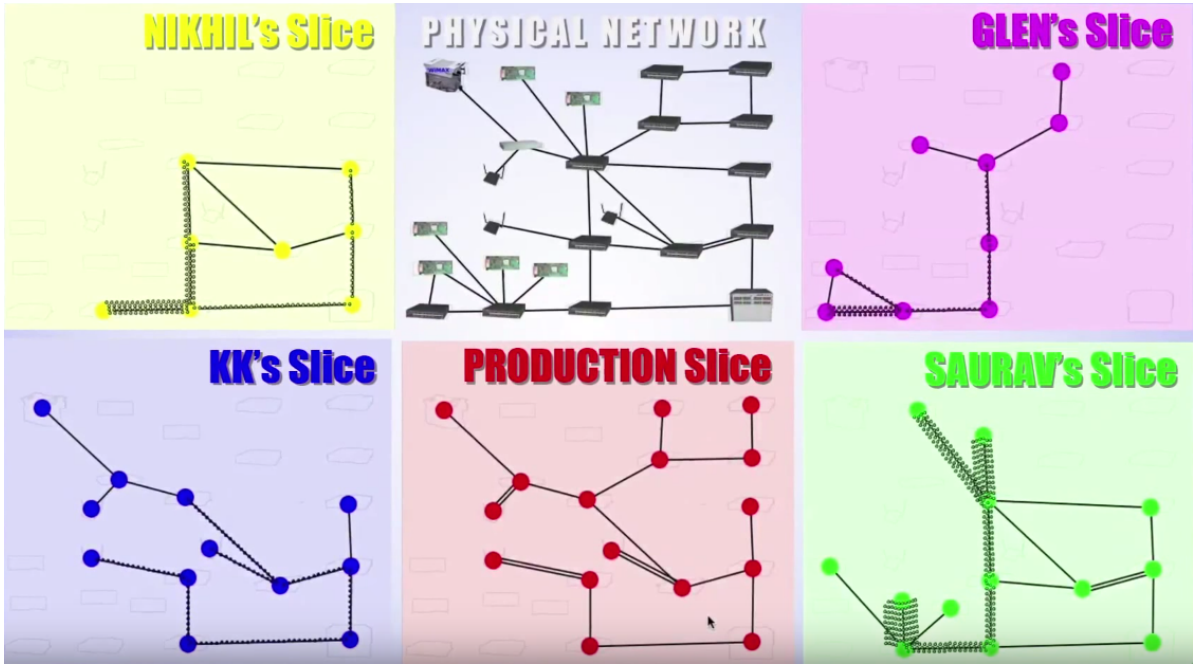
O FV segmenta a rede em *slices*, onde cada um é composto por um conjunto de lógicas de encaminhamento distintas, chamado de *flowspace* (SHERWOOD et al., 2009). O FV aloca para cada *slice* os fluxos que pertencerão a aquela rede virtual, mais especificamente o seu espaço de endereçamento de acordo com os campos do cabeçalho *OpenFlow*, garantindo desta maneira que as redes virtuais não se sobreponham (BLENK et al., 2015). Logo, as redes virtuais podem ser definidas por quaisquer combinações de campos do cabeçalho do protocolo *OpenFlow* 1.0, conforme mostra o Quadro 2.

Quadro 2- Campos do cabeçalho definidos no protocolo *OpenFlow* 1.0

Porta de entrada	ID de VLAN	Ethernet			IP			TCP/UDP	
		Endereço de Origem	Endereço de Destino	Tipo	Endereço de Origem	Endereço de Destino	Protocolo	Origem	Destino

Segundo (SHERWOOD et al., 2009), o FV possibilita que se implemente fatias com um alto nível de granularidade, no que diz respeito à caracterização do *flowspace*. Além disso, o FV garante o isolamento para cada fatia, não permitindo que uma fatia interfira no tráfego de outra. A Figura 13 ilustra o particionamento de uma rede com o FV. Cada *slice* possui sua própria visão da rede, porém com as mesmas características existentes na infraestrutura física. Além da rede de produção, têm-se mais quatro fatias identificadas por “Nikhil” e “KK”, “Saurav” e “Glen”, onde cada uma está vinculado a um controlador distinto.

Figura 13 – Fatiamento de rede do FlowVisor

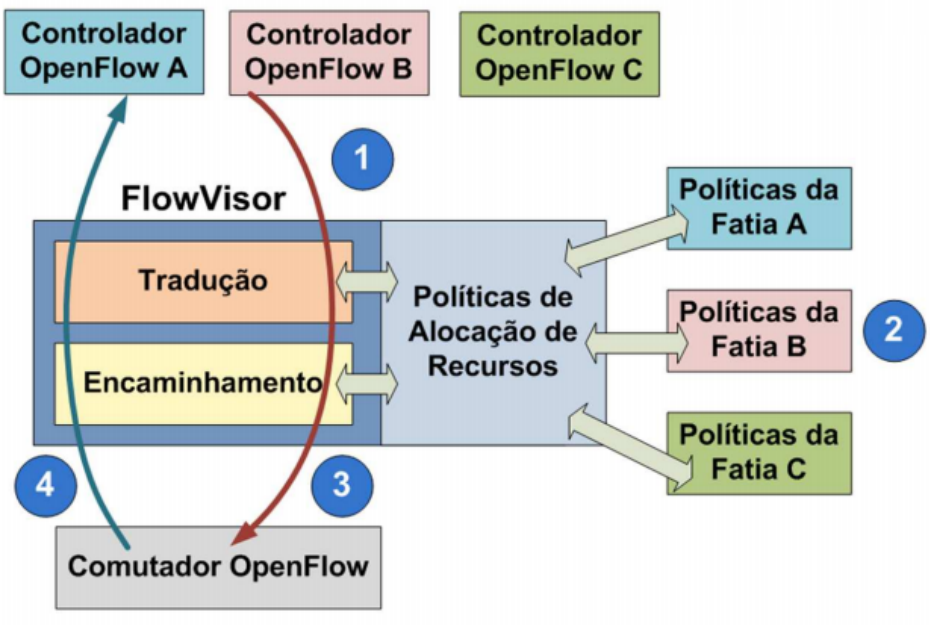


Fonte: (SHERWOOD et al., 2009).

A Figura 14 apresenta a arquitetura e o funcionamento do FV, que desempenha a tarefa de interceptar as mensagens *OpenFlow* mandadas tanto pelos controladores quanto os

switches. Inicialmente o FV processa a mensagem interceptada do controlador com destino ao plano de dados (1). Utilizando a política de *flowspace* definido para aquela fatia (2), o FV reescreve a mensagem do controlador para a fatia específica que compõe o plano de dados (3). As mensagens oriundas dos switches para os controladores (4) são novamente interceptadas pelo FV e reescritas para o respectivo controlador de acordo com a política de *flowspace*. Esse procedimento possibilita que o controle da rede virtual fique limitado ao controlador da rede.

Figura 14 – Arquitetura e funcionamento do FlowVisor.



Fonte: Adaptado de (MATTOS, 2012).

O FV encaminha as mensagens dos switches para os controladores apenas se as mensagens forem compatíveis com as políticas de *flowspace* das fatias. Encontrando uma regra de *flowspace* compatível, o FV encaminha o fluxo para o determinado controlador daquela fatia. O fatiamento do plano de controle é realizado com objetivo de manter cada rede virtual isolada das demais.

As características presentes no *FlowVisor*, como exemplo a virtualização transparente, as formas de isolamento entre as fatias e a sua rica política de definição de *flowspaces*, fazem do FV uma ferramenta extremamente eficiente no que diz respeito à implementação de redes programáveis orientadas a software (FARIAS et al., 2011).

No entanto, ainda que o FV seja capaz de suportar diversos controladores, conferindo uma fatia a cada um deles, identificou-se alguns pontos fracos ainda pouco explorados. Como o caso da falta de suporte para topologias virtuais, segundo (SHERWOOD et al., 2010) o FV não suporta a utilização de topologias virtuais arbitrárias, ou seja, topologias virtuais desassociadas dos componentes da rede física. Logo o FV não permite criar topologias com abstrações da rede física, como exemplo, criar topologias menores com menos ligações e nós de modo a diminuir a complexidade da infraestrutura na perspectiva do controlador. Esta é a principal limitação do FV, de acordo com (SALVADORI et al., 2011), tendo em vista que fica incapaz de fornecer redes com topologias flexíveis e simples de se operar, consequentemente obrigando os pesquisadores a conhecer detalhadamente os componentes da rede física (comutadores, portas, links e *hosts*) para programar seus testes em seus *slices*. Corroboram com esta afirmação (BOZAKOV; PAPADIMITRIOU, 2014) enfatizando que a partir desta fragilidade, o planejamento e instanciação de uma vSDN requer a intervenção substancial do pesquisador, não sendo uma tarefa trivial.

Outro ponto fraco do FV é referente a segmentação das redes virtuais. Cada *slice* recebe uma fatia do *flowspace* da rede, ou seja, um conjunto de métricas dos campos do cabeçalho *OpenFlow*, consequentemente somente um único *slice* tem o controle daquele determinado *flowspace*, não tendo o controle sobre os demais *flowspaces*. Logo, em uma rede com mais de um *slice*, não é possível deter o controle absoluto da totalidade do *flowspace* da rede. Desta forma, (AL-SHABIBI et al., 2014) afirmam que o FV não possibilita uma virtualização completa para as redes virtuais pois não permite o controle absoluto.

Outra fragilidade é referente à recuperação de falhas físicas. O FV não fornece redes virtuais resilientes à ruptura de links na infraestrutura física, portanto na ocorrência de falhas de um enlace que compõe um determinado *slice*, o FV não possui autonomia para definir um caminho alternativo da infraestrutura que não pertença ao domínio do *slice*. Logo, faz-se necessário que o experimentador programe manualmente caminhos redundantes para o seu *slice*, especificando as rotas a serem tomadas caso aconteça uma falha.

Com base nas limitações apresentadas, começaram a surgir diversas iniciativas de *hypervisors* a partir da implementação do FV, com novas funcionalidades ou objetivando sanar as fragilidades supracitadas. Destes, podemos destacar o *ADVisor*, *VeRTIGO*, e o *FlowSpace Firewall*, os quais serão abordados nos trabalhos correlatos.

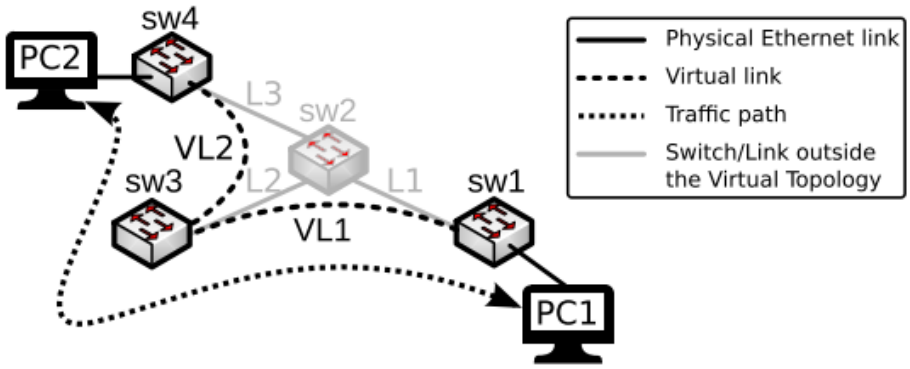
4.1.2 Alternativas ao FlowVisor: Trabalhos correlatos

Pode-se encontrar alguns trabalhos desenvolvidos que abordam as principais características, aplicações e implementações utilizando-se diferentes virtualizadores de redes *OpenFlow*. O FV por sua vez, é considerado atualmente o virtualizador de rede *OpenFlow* mais conhecido e utilizado, destacando-se em pesquisas acadêmicas, redes de produção e *testbeds* experimentais. Além disso, serviu de base para o desenvolvimento de novos *hypervisors*, dentre eles pode-se citar: *FlowSpace Firewall*[3], *ADVisor* (SALVADORI et al., 2011), *VeRTIGO* (CORIN et al., 2012) e *Double-FlowVisors* (YIN et al., 2013).

O *FlowSpace Firewall* é um *hypervisor* semelhante ao FV porém difere nas métricas de *slicing* permitidas, pois possibilita apenas a segmentação das redes por *tag* de VLAN (*Layer* 2) ou por *interfaces* dos switches (*Layer* 1). Visto que não foram implementadas funcionalidades adicionais, continua com a totalidade de limitações do FV.

Já o trabalho de (SALVADORI et al., 2011) apresenta o *ADVisor* (*ADvanced FlowVisor*), com o intuito de superar uma das principais limitações do FV, a utilização de topologias virtuais e arbitrárias sem restrições ao substrato físico da rede, conforme é apresentado na Figura 15. No entanto, a solução proposta pelos autores leva em conta a utilização de *tag* de VLAN para diferenciar *links* e redes virtuais. Portanto, limita-se ao uso de cabeçalhos de VLAN e ainda não possibilita a utilização destas métricas na segmentação das redes, impactando na principal característica do FV, a flexibilidade.

Figura 15 – Slice do ADVisor com topologia virtual.



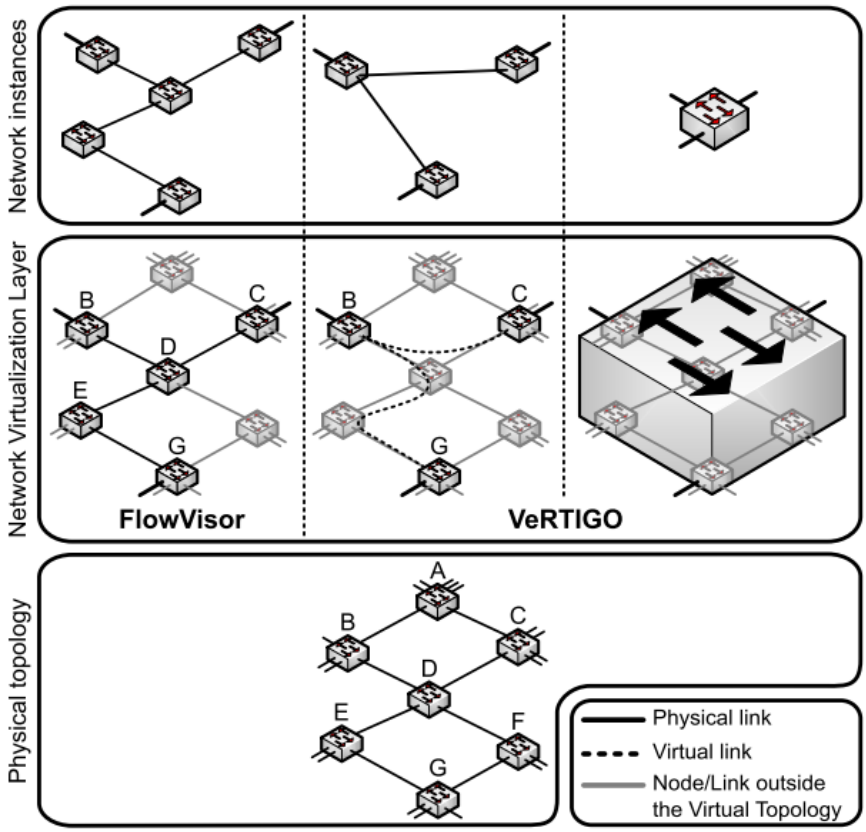
Fonte: [Salvadori et al., 2011].

3 FlowSpace firewall, GlobalNOC – FSFW: FlowSpace Firewall, URL: Acesso dia 23/05/2016).

Em (CORIN et al., 2012) é apresentado o *hypervisor VeRTIGO* (*ViRtual TopologIes Generalization in OpenFlow networks*), que de fato resolve o problema da utilização de topologias virtuais e arbitrárias, sendo possível abstrair toda a infraestrutura física em uma simples instância virtual, conforme mostra a Figura 16. Além disso, a solução apresentada possui a característica de recuperação de falhas na rede física com recursos de *failover* em caso de falhas de *link*, possibilitando a criação de redes virtuais onde o utilizador se concentra exclusivamente nas políticas de encaminhamento e controle de seu *slice*, abstraindo e deixando o controle da camada subjacente para o *VeRTIGO*.

A solução proposta detecta eventos de falhas em *links* utilizando mensagens de *status* da porta dos switches e na descoberta de falhas, a ferramenta recalcula um novo caminho para os links virtuais envolvidos. Pela forma como é realizado o *failover*, acredita-se que o tempo de convergência seja elevado em comparação com o método utilizado pelo OVX, que não necessita de novos cálculos de rota na ocorrência de falhas, pois armazena caminhos alternativos logo no estabelecimento da comunicação entre dois nós. Além disso, a ferramenta não disponibiliza um controle absoluto para cada rede virtual, conforme limitação do FV.

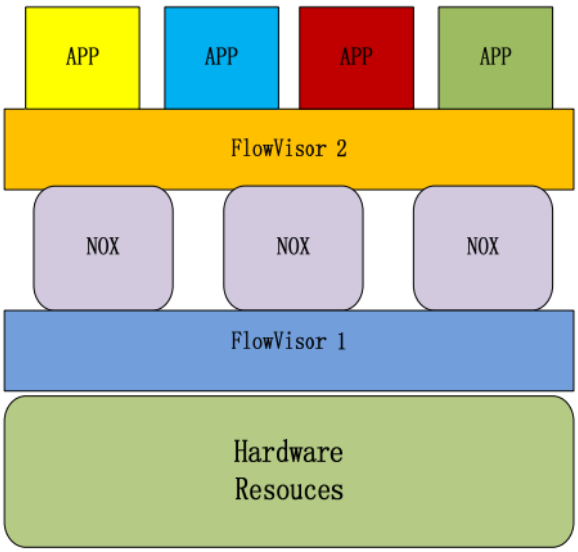
Figura 16 – Virtualização da topologia entre o FV e o VeRTIGO.



Fonte: (CORIN et al., 2012).

Os autores (YIN et al., 2013) apresentam uma plataforma de virtualização baseada em dois *FlowVisors* sobre a mesma infraestrutura, denominada como *Double-FlowVisors* introduzindo o conceito de recursividade em *hypervisors* de rede. Seu objetivo é abstrair os recursos de *hardware* e fornecer um ambiente mais flexível e eficiente para as aplicações nas camadas superiores. A arquitetura é descrita conforme a Figura 19, com dois FVs operando em uma mesma infraestrutura de rede, onde o primeiro opera entre a camada de rede e descreve a topologia física para as camadas acima. O segundo é responsável por calcular, manter as topologias virtuais e gerar a topologia de redes específicas para as aplicações. Para isso acontecer, o *FlowVisor* 2 manda uma requisição ao controlador e este faz a interação com o *FlowVisor* 1 o qual faz o mapeamento da topologia específica para a física. A solução apresentada de fato não possui nenhuma funcionalidade nova em comparação ao FV, portanto continua com as mesmas limitações e fragilidades.

Figura 17 – Composição da arquitetura do Double-FlowVisors.



Fonte: (YIN et al., 2013).

A análise dos trabalhos relacionados na área de virtualizadores de redes *OpenFlow*, demonstram que as fragilidades evidenciadas no FV não foram ainda de fato, solucionadas. O quadro que segue sintetiza as ferramentas desenvolvidas para sanar as fragilidades existentes no FV.

Quadro 3- Comparativo entre os virtualizadores de redes OpenFlow

Virtualizador de rede	**Topologia virtual**	**Flexibilidade na segmentação**	**Resiliência a falhas**	**Controle absoluto**	**Tempo de convergência**
FlowVisor	-	X	-	-	-
FlowSpace Firewall	-	-	-	-	-
ADvisor	X	X*	-	-	-
VeRTIGO	X	X	X	-	Alto
Double-FlowVisors	-	X	-	-	-

Percebe-se que apesar do surgimento de diversas alternativas ao FV, nenhuma delas de fato trouxe uma solução efetiva para as suas fragilidades. Elliot (2015) ao referir-se aos novos

projetos de *testbeds* SDNs, enfatiza para a importância dos investigadores estarem ciente das falhas em *hypervisors* como o *FlowVisor* e/ou desenvolvidos com base nele, para que possam buscar novos métodos e soluções de virtualizadores de rede que consigam fornecer uma virtualização completa da rede.

Nesta perspectiva, a RNP em parceria com o *testbed FIBRE*, realizou uma chamada de trabalhos para incentivar pesquisas e melhorias sobre a rede de experimentação, no qual foi aceito um projeto intitulado de *OpenVirteX* no ambiente *FIBRE*[4], mostrando-se favorável em buscar alternativas às fragilidades existentes na implementação padrão do FV. Tal prerrogativa, evidencia a importância deste trabalho, tendo em vista que ele se insere no âmbito de busca de soluções sobre as redes de experimentação.

4.1.3 OpenVirteX

O *OpenVirteX* (OVX) (AL-SHABIBI et al., 2014) é um *hypervisor* semelhante ao FV, que através da virtualização viabiliza a criação de redes virtuais SDN (vSDNs). O OVX também é considerado um controlador especial e age como um *proxy* sobre o plano de controle, intermediando as comunicações entre os controladores e os dispositivos da infraestrutura de rede.

Os dois *hypervisors* se diferem, tanto no método de funcionamento como em suas características. A primeira diferença é na separação das redes virtuais, enquanto o *FlowVisor* segmenta a rede em *slices* com lógicas de encaminhamento distintas, o OVX cria redes virtuais completas, com controle total do *flowspace* para cada rede virtual criada. Além disso, o OVX permite criar topologias virtuais e arbitrárias possibilitando a abstração de recursos da rede de modo a facilitar na operacionalização por parte do pesquisador.

Al-Shabibi et al. (2014) acreditam que uma infraestrutura de rede SDN ideal seria capaz de fornecer redes virtuais completas onde estariam totalmente sobre controle de seus operadores. No entanto, para isso não basta construir apenas redes virtuais logicamente isoladas, conforme é feito no FV. O mesmo autor define que é necessário fornecer vSDNs com topologias totalmente configuráveis e com funcionalidades semelhantes as conhecidas da virtualização computacional, como a habilidade de criar, deletar, pausar e migrar sobre demanda, características encontradas no OVX.

4 Disponível em: http://fibre.org.br/wp-content/uploads/2015/10/RF-OpenVirteX_CT-FIBRE.pdf

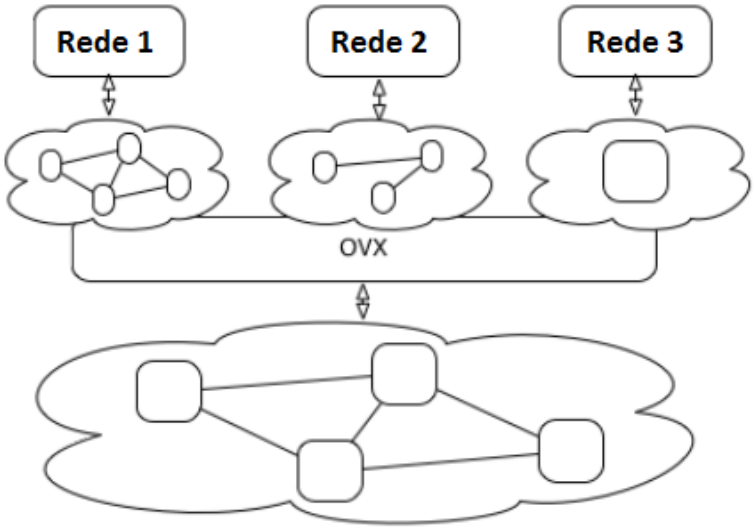
O OVX proporciona aos operadores a ilusão de que se têm o controle total da topologia, endereçamento IP e gestão da infraestrutura, apesar de compartilhá-lo com outros usuários. O OVX foi criado com o objetivo de simplificar e adicionar flexibilidade no processo de provisionamento de redes (AL-SHABIBI et al., 2014). No processo de customização da topologia, um *link* virtual pode ser mapeado para múltiplos *links* físicos e vice-versa, um *switch* virtual pode corresponder a múltiplos *switches* físicos, chamado de *big switch*. Portanto, uma rede física complexa pode ser abstraída para uma topologia virtual mais simples, com menos nós e ligações (BLENK et al., 2015), facilitando a operacionalização por parte dos operadores das vSDNs. Corroboram com esta ideia (AL-SHABIBI et al,. 2014) afirmando que o acoplamento de componentes físicos e virtuais na forma de mapeamento N:1, ou seja, um conjunto de elementos da infraestrutura física mapeados para um componente virtual apresenta um ganho significativo de flexibilidade, possibilitando duas importantes funcionalidades para as vSDNs:

- **Topologia customizável**: vSDNs podem criar topologias virtuais sem referência ao substrato físico, ou seja, não é necessário restringir a sua topologia real. Como exemplo, um *link* virtual pode abranger vários saltos contínuos e switches virtuais podem abstrair partes ou a totalidade de uma rede.
- **Resiliência**: Um *link* ou *switch* virtual podem ser mapeadas em vários componentes físicos para fornecer redundância. Uma ligação virtual resiliente é caracterizada por múltiplos caminhos físicos até o ponto correspondente aos *hosts* finais. Um *switch* virtual que abstrai partes da rede física pode tirar proveito de despedimentos na topologia (por exemplo, vários caminhos) para fornecer vários caminhos entre suas portas.

O OVX mantém representações internas das redes virtuais e física como uma coleção de *switches*, *links* e *hosts*. A representação da topologia física é a base sobre a qual as redes virtuais são mapeadas. Cada rede virtual pode dispor de uma topologia própria, e o OVX armazena o mapeamento entre os elementos da topologia virtual e sua rede física. Conforme a Figura 18, topologias virtuais podem variar de uma cópia exata da rede física (rede 1), ou de um subconjunto de dispositivos da infraestrutura física (rede 2), ou então de apenas um *switch* virtual que corresponde a todos os switches e *links* da rede física (rede 3). Ou seja, os operadores podem criar qualquer tipo de topologia, de acordo com seus interesses. Por exemplo, um operador pode exigir a passagem de tráfego entre dois *hosts* geograficamente

distantes e sem ligação direta, modificando e criando um caminho de ligação entre eles através de um link virtual ou então criando um switch virtual que corresponda a todos switches físicos da rede, ligando cada host nas portas do *switch* virtual.

Figura 18 – Segmentação de redes do OpenVirteX.



Fonte: Adaptado de (Al-Shabibi et al., 2014).

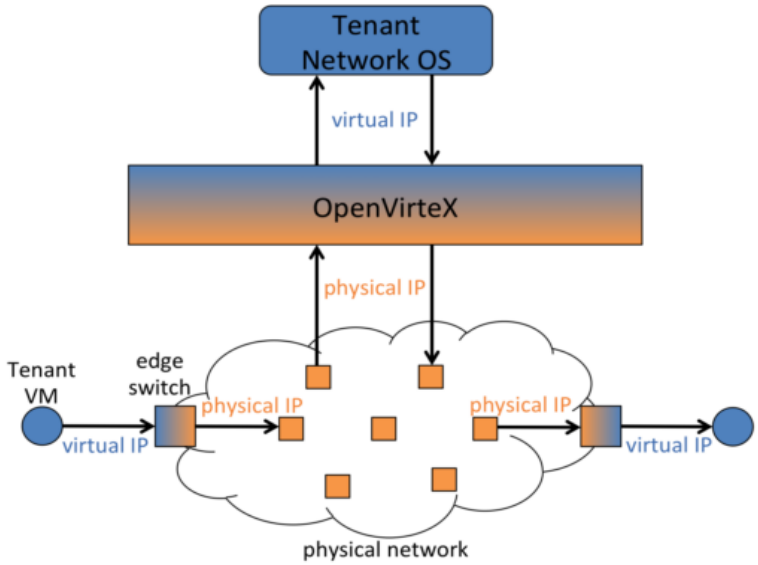
Através da virtualização da topologia física o OVX cria redes virtuais resilientes, em que reage de forma automática a problemas na infraestrutura física, como exemplo de rupturas de *link*, assegurando o transporte contínuo de fluxos nas vSDNs e totalmente transparente ao operador. Logo, essa característica possibilita aos operadores a total abstração da infraestrutura física, sem a necessidade de se preocupar e programar soluções privativas para falhas de conectividade, pois a própria infraestrutura já é resiliente e dispõe essa funcionalidade.

Além da virtualização da topologia, (BLENK et al., 2015) afirma que o OVX soluciona o problema do *flowspace* encontrado no FV, possibilitando para cada vSDN o uso do espaço completo dos campos do cabeçalho *OpenFlow*. A fim de garantir isso, o OVX realiza um mascaramento do endereço IP dos *hosts* pertencentes a rede virtual a fim de não precisar compartilhar as métricas de cabeçalho entre as demais redes. Dessa forma, cada rede virtual tem um endereço de rede único dentro da mesma rede física.

Este processo é feito por switches de borda (*edge switchs*), conforme é visto na Figura 19. São definidos *edge switchs* nas bordas da vSDN cujo possuem a tarefa de atribuir

endereços IP virtuais no início da comunicação e retornar ao IP real quando chegar no edge *switch* de destino. Assim, é possível isolar as vSDNs sem a necessidade de fatiar o *flowspace*, assegurando a totalidade de métricas do cabeçalho *OpenFlow* para cada vSDN. O OVX é responsável pelo mapeamento entre os endereços virtuais e reais e armazena nos *edge* switches estas informações.

Figura 19 – Virtualização de endereçamento do OpenVirteX.



Fonte: (AL-SHABIBI et al., 2014).

Apesar de o OVX trazer soluções, as limitações do FV de virtualização de topologia, controle absoluto para as vSDNs e recuperação de falhas, foi identificado uma fragilidade da qual não é encontrada no FV. Trata-se da flexibilidade na segmentação das redes, principal característica do FV e em virtude do OVX não permite fatiar o *flowspace*, não é possível criar vSDNs tão flexíveis como no FV, que faz por métricas de identificação de fluxos.

Através disso, o OVX não permite associar um mesmo *host* em duas ou mais vSDNs, pois cada rede virtual é identificada pelo seu endereçamento IP virtual, consequentemente um único host não pode responder por dois IPs distintos. Diferente do FV, que um *host* pode pertencer a diversos *slices*, sobre métricas de fluxos diferentes. Então, conclui-se que o OVX não fornece a mesma flexibilidade para as redes virtuais da qual é encontrada no FV.

A partir do estudo desenvolvido, justifica-se a proposta de solução baseada na integração das ferramentas *OpenVirteX* e *FlowVisor. A*credita-se que é possível a partir da integração, viabilizar aos usuários de ambientes de experimentação, redes virtuais com

topologia virtual e arbitrária de fácil manejo, flexíveis e, ainda resilientes, com recuperação automática de falhas de conectividade entre enlaces físicos.

5 RESUMO DAS TEORIAS

Este capítulo apresentou os principais conceitos que abrangem a investigação realizada, quais foram: Redes Definidas por *Software*, Virtualização de Redes *OpenFlow* e Redes de Experimentação *OpenFlow*. No capítulo 2 **Redes definidas por *Software*** destacou-se a definição e caracterização desta tecnologia, enfocando no funcionamento do protocolo *OpenFlow*. No capítulo 3 **Redes de Experimentação *OpenFlow*** apresentou-se os principais *testbeds* em operação, identificando os seus métodos de operação e as ferramentas utilizadas para a virtualização de redes, dentre as quais, destacou-se a preponderância do virtualizador de rede o *FlowVisor*. No capítulo 4 **Virtualização de Redes *OpenFlow,*** o conceito de virtualização de redes é exposto, a partir do contexto de redes definidas por *software*. Destacou-se o conceito de *Hypervisors* de redes, salientando o funcionamento bem como as funcionalidades e limitações do *FlowVisor*. Apresentou-se nesta seção os trabalhos correlatos, os quais sinalizam para o esforço em solucionar as limitações da implementação padrão do *FlowVisor.* A partir da análise empreendida identificou-se que nenhuma das propostas obtiveram êxito na solução das limitações do FV, referendando a justificativa da proposta deste estudo. Na seção 4.1.3 *OpenVirteX*, apresentou-se as características e funcionalidades de um novo *hypervisor,* distinto do *FlowVisor* e dos demais referenciados. Argumenta-se para a possibilidade de integração das duas ferramentas, retomando a hipótese norteadora do estudo, tendo em vista que a integração permitiria unir as funcionalidades de ambas as ferramentas.

6 METODOLOGIA

Para o desenvolvimento deste trabalho, utilizou-se o método de pesquisa hipotético-dedutivo (NEVADO, 2008 apud POPPER, 1994). Para este autor, o método científico em primeiro lugar deve observar a realidade e com base nisso formular um determinado problema. Ou seja, os problemas surgem com a vivência e experiência da realidade prática. Formulado o problema deve-se, então, procurar uma solução para ele, o que caracteriza as hipóteses formuladas pelo pesquisador para sanar a problemática, as quais decorrem das experiências previas ou das teorias já existentes. Outra etapa do método de pesquisa hipotético-dedutivo refere-se aos testes de falseamento onde se testam as proposições levantadas para solucionar o problema por meio de testes que objetivam refutar ou aceitar hipóteses.

A partir da revisão bibliográfica chegou-se a problemática do estudo que é como garantir aos usuários de redes de experimentação *OpenFlow* um ambiente que possibilite criar redes virtuais de baixa complexidade de operação, flexíveis e resiliente a rupturas de *enlaces*. Realizada a definição do problema de pesquisa, formulou-se a hipótese de solução através de referências científicas da área, sendo a validação ou refutação da hipótese decorrente de um processo de experimentação. Ressalta-se que a hipótese definida foi: por meio da integração das ferramentas *OpenVirteX* e *FlowVisor,* é possível viabilizar aos usuários de ambientes de experimentação, redes virtuais com topologia virtual e arbitrária de fácil manejo, flexíveis e ainda resilientes, com recuperação automática de falhas de conectividade entre enlaces físicos.

Como percurso subsequente, passou-se a efetivação da proposta de integração das ferramentas *OpenVirteX* e *FlowVisor*, o que foi denominado como *OpenVisor*. Posteriormente foram realizados testes de falseamento (NEVADO, 2008 apud POPPER, 1994) onde se efetivaram testes direcionados a validar ou refutar a hipótese formulada. Para tanto, foram estabelecidos testes de software do tipo caixa-preta, que caracteriza-se por avaliar as funcionalidades da proposta e não os aspectos internos como, por exemplo, o código fonte. Segundo (MYERS, 2004), para utilizar este método, o sistema deve ser visto como uma caixa preta, onde o objetivo não se direciona a análise detalhada do comportamento interno e da estrutura do *framework*, e sim concentra-se na verificação do funcionamento de acordo com as especificações almejadas no estudo.

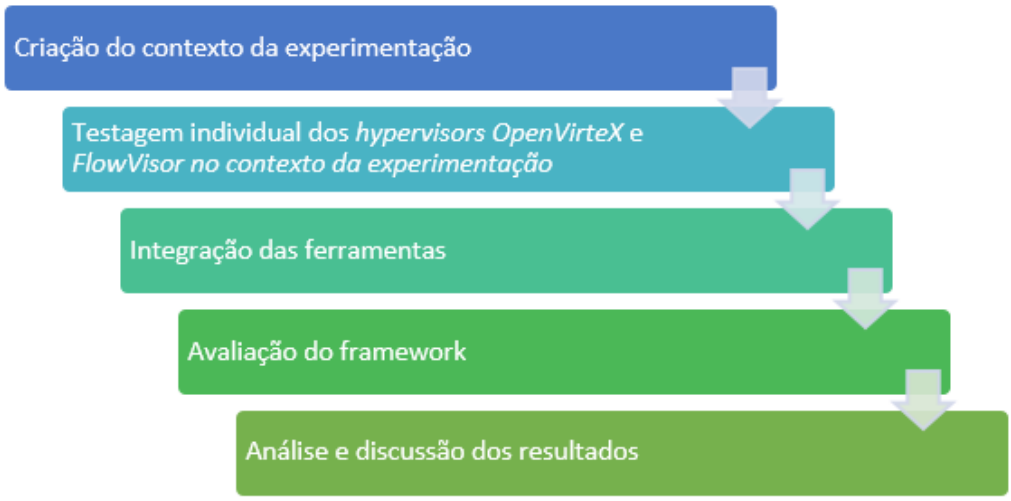
Dentro do Teste de Caixa Preta, foram escolhidos três tipos de teste de Software para a avaliação do *framework*: teste de unidade, de integração e de sistema. Nos testes de unidade foram realizados avaliações individuais de cada uma das funcionalidades dos hypervisors utilizados para a solução proposta. De acordo com (MYERS, 2004), o teste de unidade, também conhecido como teste de módulo, é definido como o processo de teste de procedimentos em um programa.

Já nos testes de integração, pautou-se na ideia de (PRESSMAN, 2011) que descreve técnicas sistemáticas de avaliação do comportamento do sistema com a integração de todas as unidades funcionando conjuntamente. Posteriormente, a realização dos testes de unidade, é realizada a integração dos componentes verificados com o intuito de testá-los de forma conjunta e analisar seu comportamento. Já os testes de sistema relacionam-se à averiguação do sistema como um todo, por meio da realização de diferentes tipos de teste. Segundo (PRESSMAN, 2011), cada um dos testes realizados no sistema tem uma finalidade diferente, mas todos funcionam no sentido de verificar se os elementos do sistema foram integrados adequadamente e executam as funções a eles alocadas.

6.1 PROCEDIMENTOS DA IMPLEMENTAÇÃO:

Nesta seção descreve-se os mecanismos aplicados para o desenvolvimento da proposta, quais foram: criação do contexto da experimentação, testes de unidade dos *hypervisors OpenVirteX* e do *FlowVisor,* integração das ferramentas e realização de testes de integração, avaliação do *Framework* a partir de testes de sistemas e, finalmente a análise e discussões dos resultados. O esquema que segue expressa a organização dos procedimentos:

Figura 20 – Procedimentos do desenvolvimento da proposta



Fonte: Próprio autor.

A seguir são descritas cada uma das etapas para o desenvolvimento da proposta, visando esclarecer de forma detalhada os métodos adotados e a forma de avaliação estabelecida.

6.1.1 Criação do Contexto de Experimentação

Primeiramente foi definido o contexto da experimentação utilizado, delimitando o cenário onde o *framework* proposto foi desenvolvido. O cenário de experimentação foi construído a partir do Mininet[5]. A escolha pelo Mininet se deu em razão deste ser uma ferramenta que possibilita a simulação de Redes Definidas por *Software*, permitindo uma rápida prototipação de uma infraestrutura virtual de rede, através da simulação de *hosts*, switches e *links* virtuais, ideal para os casos de ausência de dispositivos e infraestruturas SDN.

5 Simulador para redes OpenFlow. Disponível em http://mininet.org

6.1.2 Teste Individual dos *Hypervisors*

Criado o ambiente de testes, passou-se para a etapa da realização dos testes de unidade dos *hypervisors OpenVirteX* e do *FlowVisor*, analisando separadamente o funcionamento padrão de ambos. A testagem individual foi realizada com o objetivo de identificar, a partir da experimentação, as características individuais das ferramentas a serem utilizadas, destacando as divergências dos *hypervisors*. Os testes de unidade realizados tomaram como parâmetro as funcionalidades descritas em cada um dos *hypervisors*. Foi tomado como base das testagens, os estudos de (AL-SHABIBI, 2014) e (SHERWOOD et al., 2009), que descrevem as características dos *hypervisors*.

6.1.3 Integração das Ferramentas

A integração contemplou a união do *hypervisor OpenVirteX* e do *FlowVisor*, de modo a garantir as funcionalidades de ambas as ferramentas. Para tanto, foi necessário identificar não apenas o funcionamento individual de cada *hypervisor*, mas especialmente se era possível o funcionamento recursivo entre elas. Nesta direção, o procedimento central da integração esteve focada na recursividade das ferramentas.

6.1.4 Avaliação do Framework

A avaliação do *framework OpenVisor* se deu a partir de testes de integração e de sistema. Nos testes de integração realizou-se averiguações das funcionalidades de ambas as ferramentas utilizadas na integração. Já nos testes de sistema, foram analisados o funcionamento do sistema na tua totalidade. Finalizando, realizou-se testes comparativos com ferramentas já existentes.

- **Testes de integração:** Os testes de integração realizados para averiguar as funcionalidades do *framework OpenVisor* foram: Virtualização de Topologia; Recuperação de Falhas; Flexibilidade na definição das redes; e Controle absoluto da rede virtual. A definição destes testes objetivaram avaliar a presença das características projetadas no objetivo geral do estudo, que contempla as

funcionalidades de redes virtuais flexíveis, com topologia arbitrária e resiliente a falhas físicas.

- **Testes de sistema:** Os testes de sistema realizados para identificar o funcionamento foram: teste de comunicação básica, construção de tabela de fluxos, configuração de um servidor *Web* e transmissão de vídeo via rede. Os testes foram baseados no estudo de (PUPATWIBUL, 2016) onde buscou-se analisar o comportamento do *framework* sobre diferentes tipos de comunicações reais. Realizou-se, ainda, a avaliação do *framework* desenvolvido em comparação com as ferramentas: *FlowVisor*, *OpenVirteX* e *VeRTIGO*. Os critérios foram: quantidade de fluxos, desempenho e tempo de convergência. No que se refere a quantidade de fluxos, baseou-se na norma ISO/IEC 25062:2006[6] que define atributos de qualidade de software, enfocando especificamente na operacionalidade do *framework*. Na métrica de análise desempenho, realizou-se diferentes tipos de testes, tais como os seguintes: taxa de vazão, tempo médio de resposta do primeiro pacote e tempo médio de resposta dos demais pacotes. Tomou-se como base na RFC 2544[7] e os trabalhos de (SCHWARZ, 2014), (PUPATWIBUL, 2016). Por fim, verificou-se o tempo de convergência do *OpenVisor* comparado ao *VeRTIGO*, visto que foi a única ferramenta com a funcionalidade de recuperação de falhas, além do OVX. Tomou-se como base os testes desenvolvidos por (CORIN et al., 2012).

6.1.5 Análise e discussão dos resultados

Esta etapa, contemplou a análise dos resultados, retomando os objetivos do estudo. Seguiu-se a premissa de (NEVADO, 2008 apud POPPER, 1994), referindo-se a constatação se os resultados obtidos nos testes realizados refutam ou validam a hipótese norteadora do estudo.

6 Norma NBR ISO/IEC 25062:2006. Disponível em http://www.abntcatalogo.com.br/norma.aspx?ID=86972. Acesso em 04/10/2016.

7 RFC 2544. Disponível em https://www.ietf.org/rfc/rfc2544.txt. Acesso em 16/09/2016.

7 FRAMEWORK OPENVISOR

7.1 CRIAÇÃO DO CONTEXTO DE EXPERIMENTAÇÃO

Para testar e validar a proposta deste estudo, foi desenvolvido um cenário de experimentação SDN através da ferramenta Mininet, tendo em vista que este possibilita a simulação de Redes Definidas por *Software*. Destaca-se que o *Mininet* foi escolhido pelo *OpenFlow Consortium*, como a forma indicada de se começar a pesquisar o conceito de SDN (CONSORTIUM, 2012a). Dentre suas características, (MININET, 2014) destacam-se as seguintes:

- Fornece um ambiente de simulações barato e simples para o desenvolvimento de aplicações *OpenFlow*;
- Permite que diversos desenvolvedores trabalhem de modo simultâneo, na mesma topologia;
- Suporta testes de regressão em nível de sistema, que são repetíveis e facilmente agrupados;
- Habilita testes de topologias complexas sem a necessidade de uma rede física.

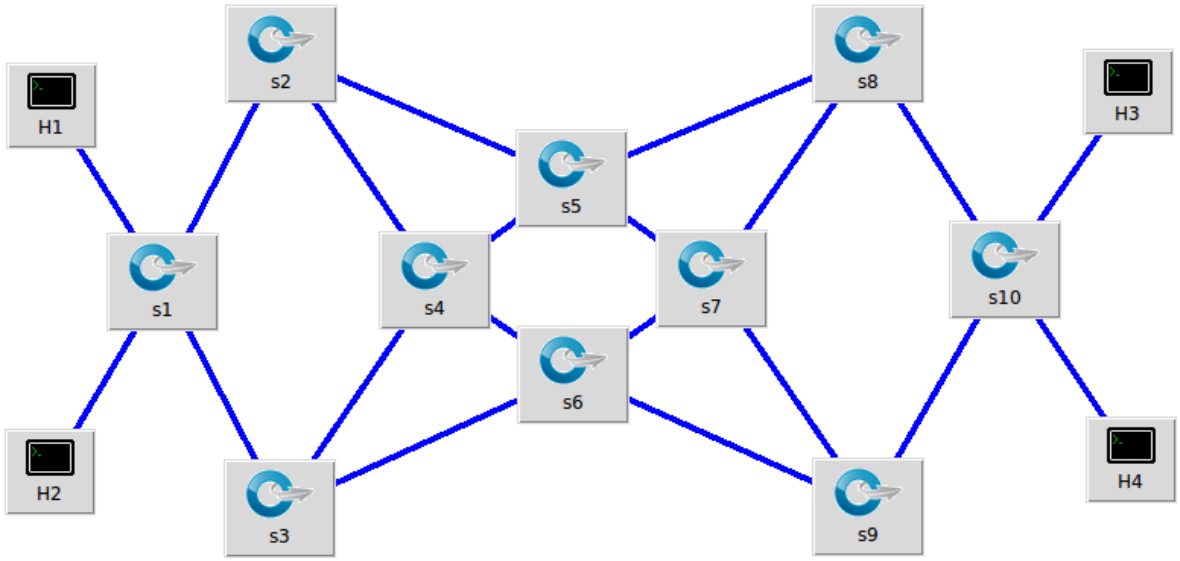
Cada vez mais são utilizados simuladores para SDN, onde atualmente o Mininet é considerado a ferramenta com maior aplicação. Diversas pesquisas utilizam o simulador em busca de validação e experimentação de testes. Entre os trabalhos, citam-se os de (CHOI et al., 2014), (CHEN et al., 2014) (CHOWDHURY et al., 2014), (ADRICHEM, 2014) e (BUENO et al., 2013) que utilizaram o Mininet para avaliar o funcionamento das suas aplicações desenvolvidas. Já no trabalho de (FARIAS et al., 2011), é simulado um *testbed OpenFlow* através do Mininet de modo a apresentar e validar seu funcionamento. O autor (PUPETWIBUL, 2016) utiliza o Mininet em sua tese de doutorado para validar as características e o funcionamento da implementação desenvolvida, analisando através de diferentes testes possíveis de dentro do Mininet, o comportamento da sua aplicação frente a diferentes tipos de comunicações.

Neste trabalho, em virtude da inviabilidade de validação em uma rede real com suporte ao *OpenFlow*, utilizou-se o Mininet para simular e criar uma infraestrutura SDN, composta por switches, *links* e dispositivos finais. Definiu-se, primeiramente os parâmetros e

as configurações, tais como a topologia a ser utilizada, *links*, taxa de transmissão, quantidade de *hosts*. A infraestrutura de rede criada inicialmente foi composta por 4 switches OpenFlow e 4 hosts. Porém, a topologia física da rede foi ampliada e passou-se a ser composta por 10 switches OpenFlow e 4 hosts, nas respectivas ligações conforme é apresentado na Figura 21.

Projetou-se esta topologia com o objetivo de fornecer múltiplos caminhos pela infraestrutura entre os dispositivos finais, visto que uma das características deste trabalho é apresentar um *framework* com a característica de recuperação de falhas.

Figura 21 – Infraestrutura de experimentação utilizada.



Fonte: Próprio autor.

Para tanto, utilizou-se um servidor para hospedar máquinas virtuais (*Virtual Machines* - VMs), tanto para o servidor do Mininet quanto para os demais elementos que compõem este contexto de experimentação. Logo, todos os componentes deste contexto de experimentação foram criados utilizando um servidor Dell PowerEdge T710 com 32GB de memória RAM, 2TB de HD e 2 processadores *Intel Xeon* de 4 núcleos cada, com o sistema operacional *CentOS* versão 6.5 e kernel 2.6.32.

O *Mininet* foi configurado através de uma máquina virtual e determinou-se as seguintes configurações de hardware: 1 processador com 1 núcleo, 4GB de memória RAM e

8GB de HD. A versão utilizada do *Mininet* foi a 2.2.1. Sua implementação utiliza uma API baseada em linguagem de programação *Python*, onde as configurações podem ser definidas através de *scripts* de programação.

Já os *hypervisors* que compõem a solução proposta, o *OpenVirteX* e o *FlowVisor*, também foram implementados via máquinas virtuais no servidor hospedeiro anteriormente detalhado. Para o *FlowVisor* utilizou-se uma máquina virtual com as seguintes configurações: 01 processador com 2 núcleos, 4GB de memória RAM e 8GB de HD. Utilizou-se a versão 1.4 do FV, última disponível até o término desta dissertação. O FV utiliza dois tipos de API, ambas em JSON-RPC, uma destinada para tarefas de leitura de informações e usada para obter configurações e o estado dos dispositivos e a outra é utilizada para criar e configurar as redes virtuais.

Para o *OpenVirteX* foi criado uma VM com as mesmas configurações da VM do FlowVisor: 1 processador com 2 núcleos, 4GB de memória RAM e 8GB de HD. Utilizou-se sua última implementação até o momento, denominada OpenVirteX-0.1-DEV[8]. Da mesma forma que o FV, ele trabalha com duas APIs em JSON-RPC, uma para obter as informações e a outra para operar e configurar as redes virtuais.

Por fim, utilizou-se dois controladores *OpenFlow Floodlight* versão 0.90 através de máquinas virtuais no mesmo servidor físico dos demais. Suas configurações de hardware foram as seguintes: 1 processador com 1 núcleo, 2GB de memória RAM e 8GB de HD. Sua implementação oferece uma API baseada em REST. A escolha pelo *Floodlight* levou em conta que ambos os hypervisors tinham suporte a ele e pela ampla utilização através do FV e do OVX.

7.2 TESTES INDIVIDUAIS DOS HYPERVISORS

Nesta seção é descrito separadamente cada um dos *hypervisors* utilizados no desenvolvimento da solução proposta. Visto que o *framework OpenVisor* é composto pela integração do OVX com o FV, será apresentado separadamente cada um e demonstrado o funcionamento convencional de ambos, objetivando apresentar seu funcionamento em sua forma padrão de operação.

8 Informações retiradas do site oficial http://ovx.onlab.us

Tanto o OVX quanto o FV são denominados como *hypervisors* de redes *OpenFlow*. Desta forma, possuem semelhanças intrínsecas mesmo contendo modos de operação diferentes. Em uma infraestrutura de rede *OpenFlow*, os switches são direcionados para o controlador para que operem de acordo com suas instruções. Já em uma rede SDN com suporte a virtualização, todos os switches são direcionados para o *hypervisor* de rede, agindo na perspectiva dos comutadores como um controlador, dando instruções de operação para os componentes da infraestrutura de rede.

No panorama dos controladores *OpenFlow* (*e.g. Floodlight*, POX, NOX, Ryu), somente o *hypervisor* estabelece conexão ao controlador. Porém o *hypervisor* fica responsável por intermediar as informações provenientes dos switches e, de forma transparente aos controladores. Desta forma, o *hypervisor* pode definir quais informações os controladores terão acesso da infraestrutura física, seja sobre os dispositivos, *links*, portas e *hosts* finais presentes na topologia física, como também sobre topologias e componentes virtuais inexistentes na infraestrutura real, podendo modificar, ou não, as mensagens que transmite aos controladores e/ou aos switches.

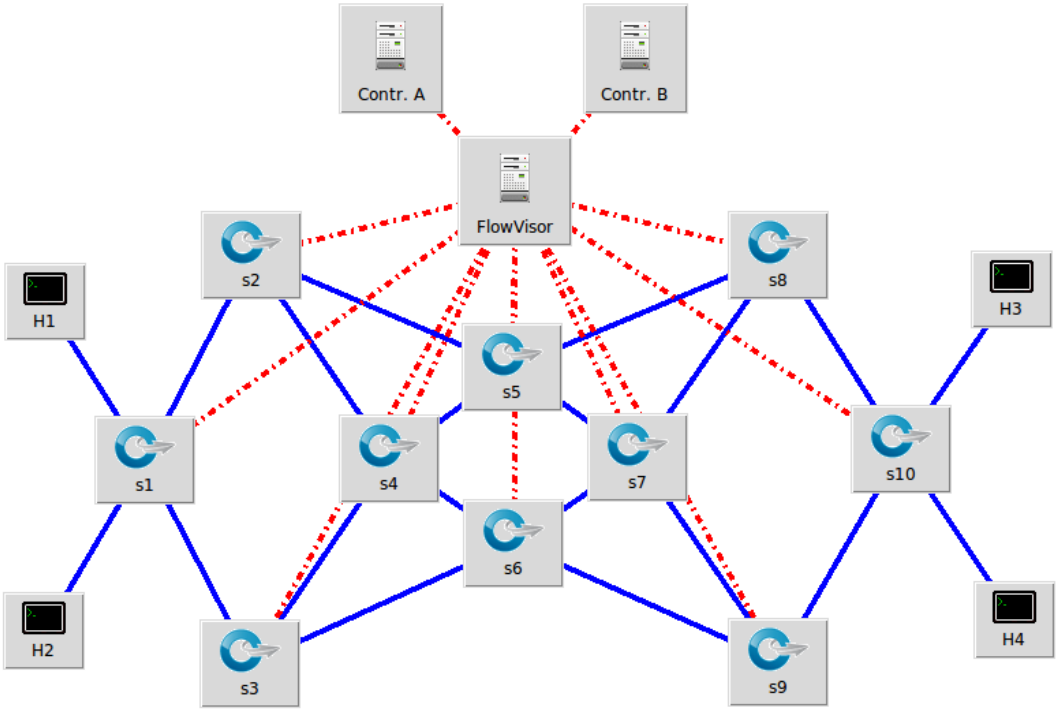
Visto que o controlador recebe apenas a conexão do *hypervisor*, sua perspectiva é de que está diretamente conectado com os dispositivos reais, da mesma forma que em uma rede SDN sem suporte a virtualização. O *hypervisor* assume diferentes papéis em seu funcionamento, sobre a perspectiva dos switches, o *hypervisor* assume o papel de controlador. Já na perspectiva dos controladores *OpenFlow*, o *hypervisor* age como se fosse toda a infraestrutura de rede, repassando as informações dos switches e *hosts*, totalmente transparente ao controlador.

7.2.1 FlowVisor

Com o objetivo de analisar o comportamento do FV em sua forma convencional de funcionamento, foram realizados testes no contexto de experimentação adotado. Utilizou-se o FV comunicando-se diretamente com os switches e com os controladores *OpenFlow*. Foram criados 2 *slices*, sendo que para cada um foi definido uma política de fluxos diferenciada. Também foi designado controladores *OpenFlow* distintos para cada *slice*, neste caso duas instâncias do controlador Floodlight.

Todos os switches foram conectados ao FV, que por sua vez passou a intermediar as comunicações com os controladores. A topologia do experimento foi composta por 2 controladores, 10 switches *OpenFlow* e 4 dispositivos finais (*hosts*). Os switches foram conectados no FV, conforme é apresentado na Figura 22.

Figura 22 – Topologia de experimentação com o FV.



Fonte: Próprio autor.

O FV possui a visão total da infraestrutura de rede pois sua posição permite que se tenha uma perspectiva completa da rede. Inicialmente configura-se os *slices*, definindo quais recursos da infraestrutura serão definidos para cada um, como switches, *links* e *hosts*, criando uma rede virtual customizada e com topologia própria para cada *slice*. Para a configuração de um *slice* é necessário utilizar alguns comandos através da CLI de gerenciamento *fvctl* do FV. O primeiro comando necessário é o "add-slice", cuja sintaxe é a seguinte:

```
fvctl add-slice [options] <nome_slice> <tcp:ip_controlador:porta> <e-mail>
```

Através deste comando é criado um novo slice no FV. Além do comando add-slice é necessário passar algumas informações, como um nome para o novo slice, o IP do controlador *OpenFlow* que gerenciará o *slice* e um e-mail para contato. Para o experimento, foram criados dois *slices*, denominados como “redeA” e “redeB”, atribuindo a gerência para o controlador A e controlador B, respectivamente.

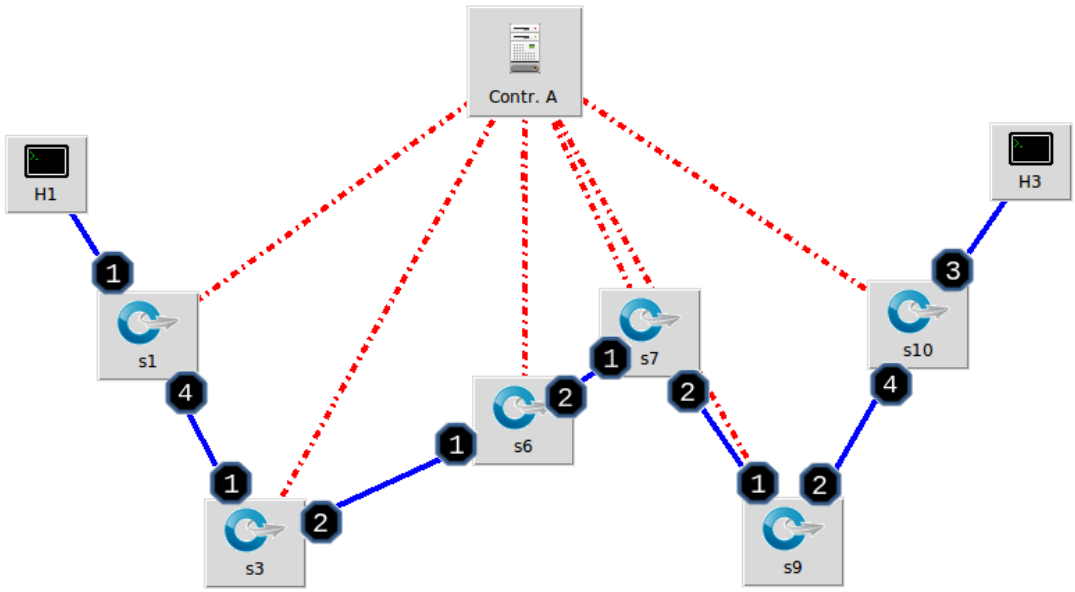
Posteriormente, configurou-se o *flowspace* de ambos *slices*, onde foi definido quais recursos (switches, *links* e hosts) fariam parte de cada slice e também o tipo de fluxos que cada rede virtual iria controlar. Através da definição do *flowspace* é que o FV consegue repassar os fluxos ao controlador correto. Utiliza-se o comando “add-flowspace” cuja sintaxe é a seguinte:

```
fvctl add-flowspace [opções] <nome-flowspace> <dpid> <prioridade> <match> <permissão-slice>
```

O comando “add-flowspace” requer que seja atribuído um nome a cada regra de *flowspace* criada. Além disso, deve-se informar o endereço DPID (*Datapath Identifier*) do *switch* onde será aplicada a regra. Depois, deve-se configurar uma prioridade para cada regra, visto que um fluxo pode atender a múltiplas regras de *flowspace.* A eleição da regra pelo FV se dá pela maior prioridade. O argumento “match” refere-se ao tipo de fluxo, ou seja, quais métricas do cabeçalho *OpenFlow* o fluxo deve ter para ser incluso ao determinado *slice* e consequentemente encaminhado ao controlador do *slice*. É possível escolher um ou mais campos do cabeçalho de fluxos do protocolo *OpenFlow* 1.0. Por fim define-se a permissão do *slice*. Pode-se utilizar os seguintes valores: 1= Delegar, 2= Leitura e 4= Escrita.

Portanto, foram configurados os dois *slices*, com diferentes recursos da rede e padrões de fluxos, consequentemente cada controlador apresenta uma visão diferente sobre a rede. Na perspectiva do controlador A é apresentado uma topologia de rede customizada de acordo com seu *slice*, fazendo-o acreditar que possui a visão completa da infraestrutura e ainda a soberania no controle, desconhecendo a existência de outros controladores, inclusive do FV, e demais componentes da topologia. Isso é evidenciado na Figura 23 onde é apresentado a topologia da rede do *slice* “redeA”, na perspectiva do controlador A.

Figura 23 – Perspectiva global do Controlador A sobre seu *slice*.

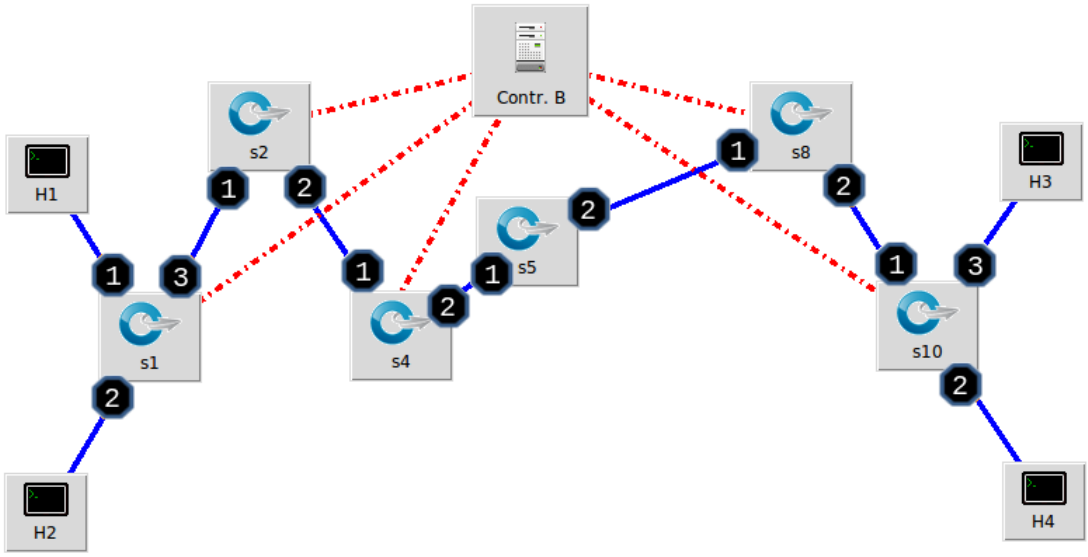


Fonte: Próprio autor.

Nesta rede, foi configurado um padrão de fluxos do tipo protocolo TCP e porta 9999 e foi atribuído a gerência ao controlador A. Logo, qualquer fluxo *OpenFlow* com estas políticas e provenientes dos *hosts* H1 ou H3 são encaminhados para o controlador A. A comunicação entre os dois *hosts* ocorre através dos switches s1, s3, s6, s7, s9 e s10.

Já no segundo *slice* denominado “redeB”, configurou-se um *flowspace* diferente, com componentes e políticas de fluxos distintas da “redeA”, conforme ilustra a Figura 24. Definiu-se que o *slice* seria gerenciado pelo controlador B e controladoria quaisquer tipos de tráfego provenientes dos *hosts* H1, H2, H3 e H4. Os switches atribuídos ao *slice* para viabilizar a comunicação entre os *hosts* foram o s1, s2, s4, s5, s8 e s10. Utilizou-se como métrica do FV as *interfaces* dos switches, ou seja, permitindo quaisquer fluxos passantes na porta do *switch* e então atribuindo ao controlador B.

Figura 24 – Perspectiva global do Controlador B sobre seu *slice*.



Fonte: Próprio autor.

Para a rede B atribuiu-se uma prioridade menor nas regras de *flowspace* das regras da rede A, visto que as políticas de fluxos da rede B poderiam impactar na rede A. No caso da existência de um fluxo com as métricas TCP porta 9999, este tráfego específico seria satisfeito por ambas políticas de *flowspace*, tanto pela rede A que controla exclusivamente tráfego com estas métricas, quanto pela rede B que controla quaisquer métricas, incluindo TCP porta 9999. Logo, com a definição de diferentes prioridades, fluxos com métricas TCP porta 9999 eram encaminhados ao controlador A enquanto os demais eram repassados ao controlador B.

A partir destas configurações, o controlador *OpenFlow* de cada slice já possui total autonomia para gerenciar e operar as comunicações de sua rede, podendo então definir suas próprias políticas de operação de acordo com seus interesses e/ou experimentos. Entretanto, mesmo já criado o *slice*, a comunicação entre os *hosts* deve ser habilitada através do controlador. Visto que o controlador é gerenciado por um usuário, no caso de uma rede de experimentação seja um pesquisador, todo o esforço da configuração das comunicações daqueles componentes passam a ser exclusivamente do usuário.

Para configurar quaisquer comunicações em uma rede SDN, mesmo que seja uma simples comunicação entre dois *hosts*, é necessário conhecer detalhadamente os componentes

da infraestrutura da qual se está operando, para então conseguir criar as regras e inserir nas tabelas de fluxos dos switches.

Para detalhar a complexidade desta tarefa foi configurado a comunicação entre os *hosts* H1 e H3 na redeB criando os fluxos necessários no controlador B. Neste caso, de acordo com a topologia do *slice* rede B, exposto na Figura 24, foram necessários adicionar 12 regras, das quais o usuário teve que levar em conta as seguintes informações: topologia da rede, switches, *links*, *hosts*, interfaces de conexão entre os switches, *mac-address* dos *hosts*, DPID dos switches e ainda a lógica de processamento das informações pelos switches, ou seja, o caminho pelo qual os fluxos percorrerão tanto de ida como de volta. O quadro 4 representa as regras de fluxos criados para habilitar a comunicação entre os *hosts* H1 e H3.

Quadro 4- Regras de fluxos instalados nos comutadores do slice redeB

Regra	**Tráfego**	**Switch**	**Campos do cabeçalho**	**Ação**
1	Partida	s1	in_port:1	out_port:3
2	Partida	s2	in_port:1	out_port:2
3	Partida	s4	in_port:1	out_port:2
4	Partida	s5	in_port:1	out_port:2
5	Partida	s8	in_port:1	out_port:2
6	Partida	s10	"in_port:1", "eth_src":"00:00:00:00:00:11", "eth_dst":"00:00:00:00:00:13",	out_port:3
7	Retorno	s10	in_port:3	out_port:1
8	Retorno	s8	in_port:2	out_port:1
9	Retorno	s5	in_port:2	out_port:1
10	Retorno	s4	in_port:2	out_port:1
11	Retorno	s2	in_port:2	out_port:1
12	Retorno	s1	"in_port:3", "eth_src":"00:00:00:00:00:13", "eth_dst":"00:00:00:00:00:11",	out_port:1

Cada fluxo apresentado no quadro 4 possui determinadas métricas do cabeçalho que são analisadas e na ocorrência de uma comunicação correlata, o *switch* atribui a ação definida por aquela regra. Neste caso, as métricas utilizadas foram genéricas, com exceção da regra 6 e 12, definidas para qualquer tráfego de entrada nas portas dos switches, com a ação de encaminhar para uma porta de saída.

Portanto, o controlador B adicionou estas regras em cada *switch* da topologia da rede B, permitindo que os hosts consigam se comunicar. Até a regra de número 6 representa a comunicação de partida do H1 ao H3, onde a sexta regra contêm as informações de *mac-address* de origem (H1) e destino (H3). Já a partir da regra 7 é representado o retorno da comunicação, partindo do H3 para o H1, onde regra de número 12 contêm novamente informações de *mac-address*, porém agora com origem sendo o H3 e destino o H1. Nota-se que para as regras 6 e 12, as quais representam os *switchs* diretamente ligado aos *hosts*, são utilizados o endereço MAC dos *hosts*, para que, havendo outras comunicações para outros *hosts*, a comunicação consiga chegar no destino corretamente.

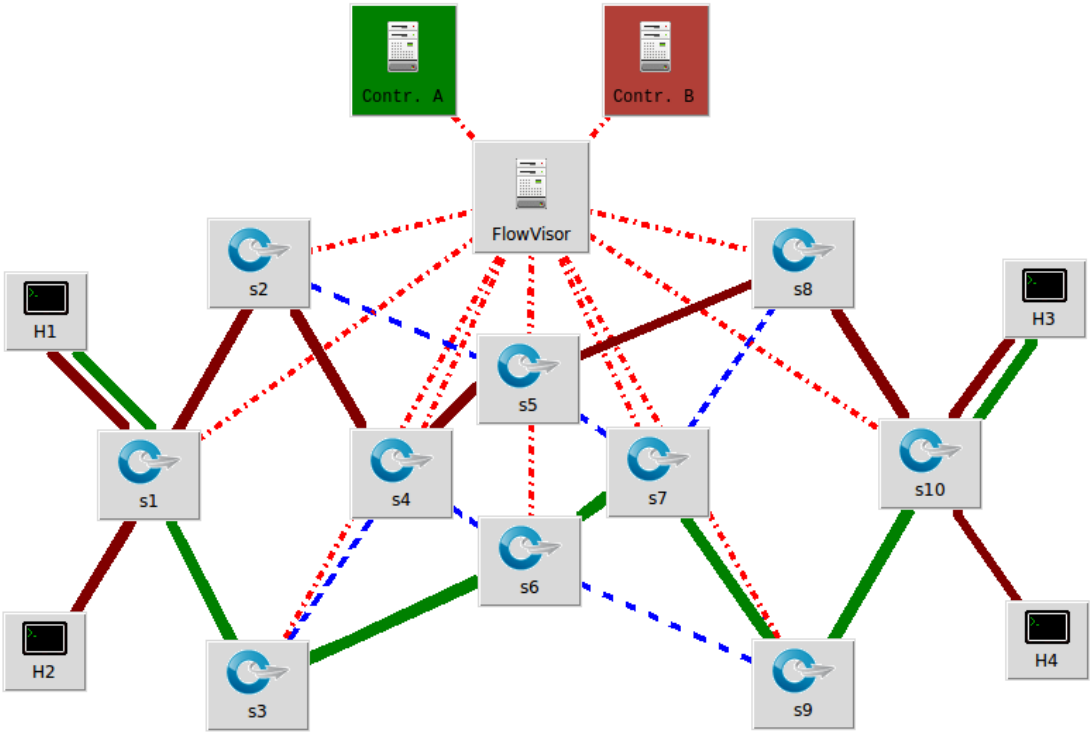
Estes 12 fluxos estabelecem a comunicação apenas entre os *hosts* H1 e H3. Os demais *hosts* presentes no *slice* permanecem sem nenhum tipo de comunicação, necessitando-se adicionar novos fluxos. Portanto, percebe-se a dificuldade em se configurar uma rede SDN, onde o nível de dificuldade aumenta de acordo com o tamanho da topologia, ou seja, quanto mais dispositivos e *hosts*, maior é o número de fluxos e informações necessárias para a configuração de experimentos, aumentando a complexidade de operacionalização e configuração, visto que o FV não age como facilitador deste processo e obriga os usuários a operar sobre as características reais da rede.

Ademais, simulou-se uma falha de conectividade nos dois *slices* existentes, de modo a verificar seu comportamento perante a falhas. Notou-se que em ambas redes a conexão foi perdida e o FV não conseguiu restabelecer a comunicação mudando a rota do tráfego. Portanto, evidenciou-se que o FV não opera com recuperação de falhas, sendo necessário programar rotas secundárias e por seguinte, necessitando que o usuário configure um sistema de *failover* no seu controlador *OpenFlow*.

Por fim, analisando a visão completa da infraestrutura de rede, nota-se que o FV é o único detentor da perspectiva global e real da rede, enquanto os controladores apenas veem as suas redes virtuais. Conforme a Figura 25, é possível verificar a visão de cada controlador e a

perspectiva global, vista apenas pelo FV. Alguns enlaces dos quais estão sinalizados em azul com linhas pontilhadas são desconhecidos e inutilizados pelos controladores.

Figura 25 – Infraestrutura de rede com o FV.



Fonte: Próprio autor.

Portanto, percebe-se que o FV permite segmentar uma infraestrutura com um alto nível de flexibilidade, através da configuração do *flowspace* por métricas de identificação de fluxos *OpenFlow*. No entanto, o experimento evidenciou suas limitações, a primeira delas é referente a falta de virtualização de topologia, fazendo o FV impactar na complexidade de operação dos testes na perspectiva do pesquisador, pois implica em fazer os usuários operar sobre a infraestrutura real da rede, onde a dificuldade aumenta de acordo com o tamanho da rede. Além disso, cabe ressaltar que na ocorrência de falhas nos componentes físicos de ambos os *slices*, a comunicação é perdida, mesmo existindo diversas rotas alternativas na infraestrutura física, pois a implementação padrão do FV não possui suporta a recuperação de falhas.

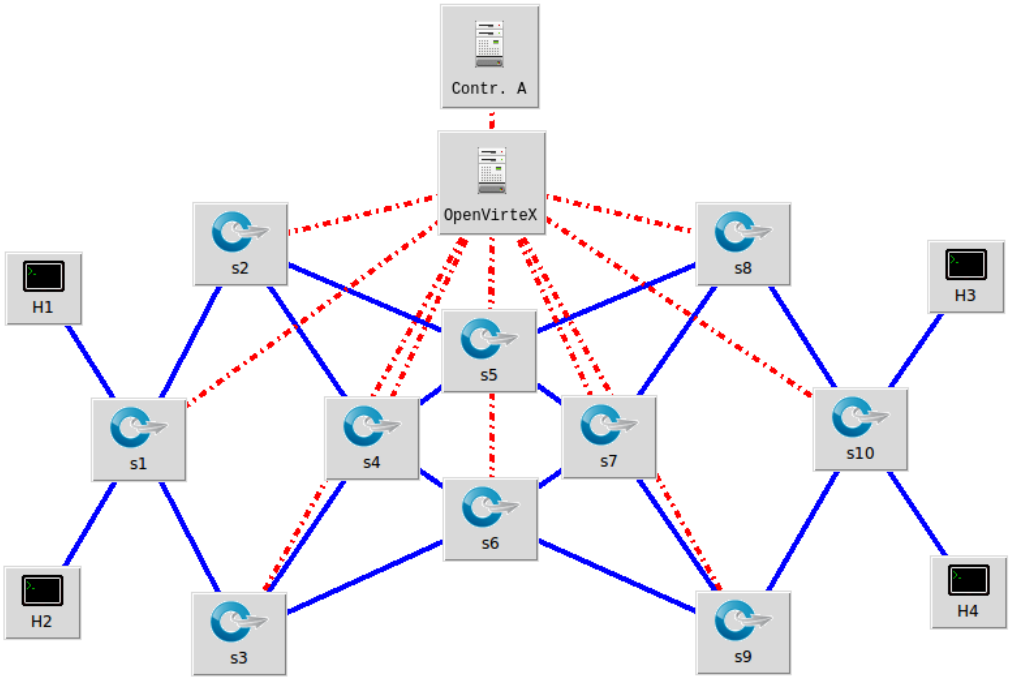
Outro ponto analisado neste experimento refere-se ao compartilhamento do *flowspace*. Para tanto, cada *slice* possui uma fatia do *flowspace* da rede, consequentemente nenhuma das duas redes possui o controle total da rede, não permitindo aos *slices* o controle absoluto da rede, mas sim apenas as comunicações definidas na política de *flowspace.*

7.2.2 OpenVirteX

Já a implementação do OVX trouxe funcionalidades inexistentes no FV e nos demais *hypervisors* citados neste trabalho. O *framework* desenvolvido neste estudo utiliza em sua arquitetura o OVX controlando todas as comunicações da infraestrutura da rede. Portanto, todas funções inerentes as tarefas na topologia física são de responsabilidade do OVX. Desta forma, realizou-se um teste somente com o OVX operando no cenário de experimentação descrito na seção 4.1, de forma a apresentar seu funcionamento convencional, objetivando identificar as diferenças e diversidades de operação em relação ao FV.

Diferente do experimento do FV, foi criado apenas uma rede virtual e utilizado uma única instância do controlador Floodlight, denominado Controlador A, atribuindo os quatro *hosts* H1, H2, H3 e H4 na rede. Utilizou-se apenas uma rede virtual pois diferente do FV, o OVX não permite que um único *host* pertença a mais de uma rede, em virtude do seu método de segmentação de redes diferente do FV. A figura 26 apresenta a visão geral da topologia através da perspectiva do OVX.

Figura 26 – Infraestrutura de rede com o OVX.



Fonte: Próprio autor.

De modo a minimizar a complexidade de operação na perspectiva do usuário, definiu-se a utilização de uma topologia virtual e arbitrária, visto que o OVX permite criar topologias desprezando as limitações da infraestrutura física, podendo criar ambientes menos complexos e fáceis de se operar. Para apresentar esta funcionalidade aos controladores, o OVX modifica as mensagens LLDP (*Link Layer Discovery Protocol*) provenientes dos controladores. Estas mensagens são utilizadas para o descobrimento dos componentes da infraestrutura de rede. Em particular, quando uma mensagem de LLDP chega a um *switch* virtual com uma certa porta de saída especificada, o OVX entende de fato para onde o controlador deseja encaminhar o fluxo, e portanto, forja um pacote de resposta LLDP e envia de volta para o controlador, criando a ilusão de um *link* real para o controlador em questão.

Desta forma, o número de pacotes LLDP que trafegam nos *switchs* permanece constante, não importando a quantidade de redes virtuais existentes, visto que o OVX intermedeia esse processo e não repassa aos switches reais tráfegos LLDP vindos dos controladores. Por conseguinte, é possível apresentar qualquer topologia virtual para os controladores *OpenFlow*.

Semelhante ao FV, o OVX também possui uma CLI de configuração para a execução de comandos, denominada *ovxctl*, que se comunica com o OVX através de sua API JSON. Para a configuração de uma rede virtual com topologia arbitrária, foi necessário os seguintes passos. Inicialmente utiliza-se o comando “createNetwork”:

```
python ovxctl.py [opções] createNetwork <tcp:ip_controlador:porta> <endereço de rede> <máscara da rede>
```

Através deste comando é criado uma rede virtual e atribuído para um controlador. O endereço de rede e máscara referem-se ao endereçamento do qual a rede virtual irá utilizar para trafegar pela infraestrutura física de rede, forma utilizada pelo OVX para garantir o controle absoluto da rede sem a necessidade de compartilhar o *flowspace*. Após executar o comando, é criado um número identificador da rede virtual, denominado como *tenant_ID* que é usado posteriormente nas configurações. O próximo passo é configurar as características da rede virtual.

É possível criar switches virtuais idênticos aos reais ou então criar um “*big switch*”, nomenclatura dada para um *switch* virtual que corresponde a mais de um comutador físico. Através de switches virtuais é possível abstrair topologias imensas e complexas para infraestruturas mais simples, com menos ligações e equipamentos, diminuindo a complexidade de operação na perspectiva do usuário nas redes virtuais. Para isso, é necessário criar os switches virtuais através do comando “createSwitch” apresentado a seguir:

```
python ovxctl.py [opções] createSwitch <tenant_ID> <DPID-switch físico>
```

Neste caso, criou-se um *big switch* referenciando-se a totalidade da infraestrutura de rede, todos *switches* físicos e *links* existentes. Executado o comando *createSwitch*, é criado o *switch* virtual e é gerado um DPID próprio para o dispositivo virtual, conforme é visto a seguir.

```
Virtual switch has been created (tenant_id 1, switch_id 00:a4:23:05:00:00:00:01)
```

Depois, é necessário adicionar as portas virtuais. As portas virtuais são utilizadas para conectar *hosts* e/ou *enlaces* com outros equipamentos. No caso de enlaces, é possível criar

links arbitrários, ou seja, *enlaces* inexistentes na topologia real. Já para os *hosts*, o OVX não impõe limitações de quantidade de portas virtuais para cada *switch*, logo é possível conectar quantos *hosts* desejar. Neste experimento definiu-se um *switch* com todos os *hosts* conectados em suas portas, H1, H2, H3 e H4. Para isso, utilizou-se o comando *createPort*, conforme a seguir:

```
python ovxctl.py [opções] createPort <tenant_ID> <DPID-switch físico> <porta física>
```

Após criar as quatro portas virtuais, o OVX retorna uma mensagem mostrando as informações criadas. É apresentado o número da porta virtual criada e em qual *switch* virtual ela foi adicionada.

```
Virtual port has been created (tenant_id 1, switch_id 00:a4:23:05:00:00:00:01, port_id 2)
```

Após a configuração das portas virtuais, é necessário conectar os *hosts*. Para isso é preciso informar em qual *switch* virtual o *host* vai conectar, a porta virtual e o endereço MAC do *host*, conforme o comando a seguir:

```
python ovxctl.py -n connectHost 1 00:a4:23:05:00:00:00:01 4 00:00:00:00:00:31
```

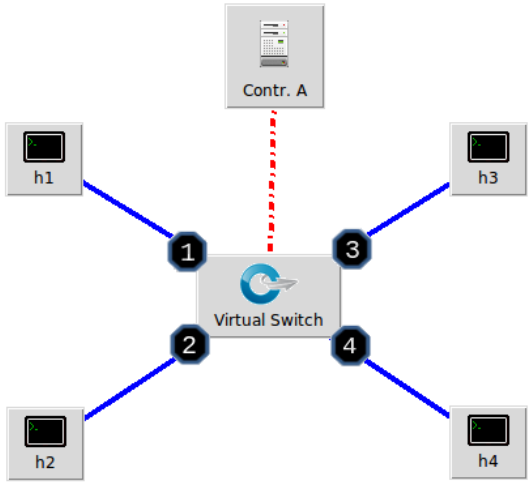
Neste exemplo, adicionou-se um *host* na rede virtual com identificador “1” (*tenant_ID*), no *switch* virtual com DPID 00:a4:23:05:00:00:00:01, na porta virtual “4” e o *host* cujo endereço MAC é o 00:00:00:00:00:31.

Por fim, basta iniciar a rede virtual criada, através do comando a seguir:

```
python ovxctl.py -n startNetwork 1
```

A partir de então, a rede está criada e o OVX transmite as informações configuradas para o controlador definido. Neste experimento o controlador da rede virtual 1, Controlador A, recebe as informações e sua perspectiva da rede pode ser vista na Figura 27.

Figura 27 – Perspectiva global do Controlador A sobre sua rede virtual.



Fonte: Próprio autor.

Nesta rede, foi criado uma topologia arbitrária sem nenhuma referência as características físicas da topologia real, como *links* e switches físicos. Foi criado 01 *switch* virtual e adicionado os *hosts* H1, H2, H3 e H4 nas portas virtuais 1, 2, 3 e 4, respectivamente. Através da perspectiva do Controlador A, nota-se que a topologia da rede controlada pelo usuário é significativamente simples, visto desconhecer a infraestrutura física e suas ligações reais. Portanto, as configurações por parte do usuário se tornam mais simples e exigem um menor número de fluxos.

Para a mesma comunicação criada no experimento anterior, onde o Controlador A criou uma comunicação entre H1 e H3 e foram necessários 12 fluxos, neste caso é necessário apenas 2 fluxos, onde a primeira regra define que as comunicações provenientes do H1 (porta 1) devem ser encaminhadas para o H3 (porta 3) e o segundo fluxo estabelecendo o caminho de volta (H3 – H1), conforme é demonstrado no Quadro 5.

Quadro 5- Fluxos de encaminhamento para comunicação entre H1 – H3.

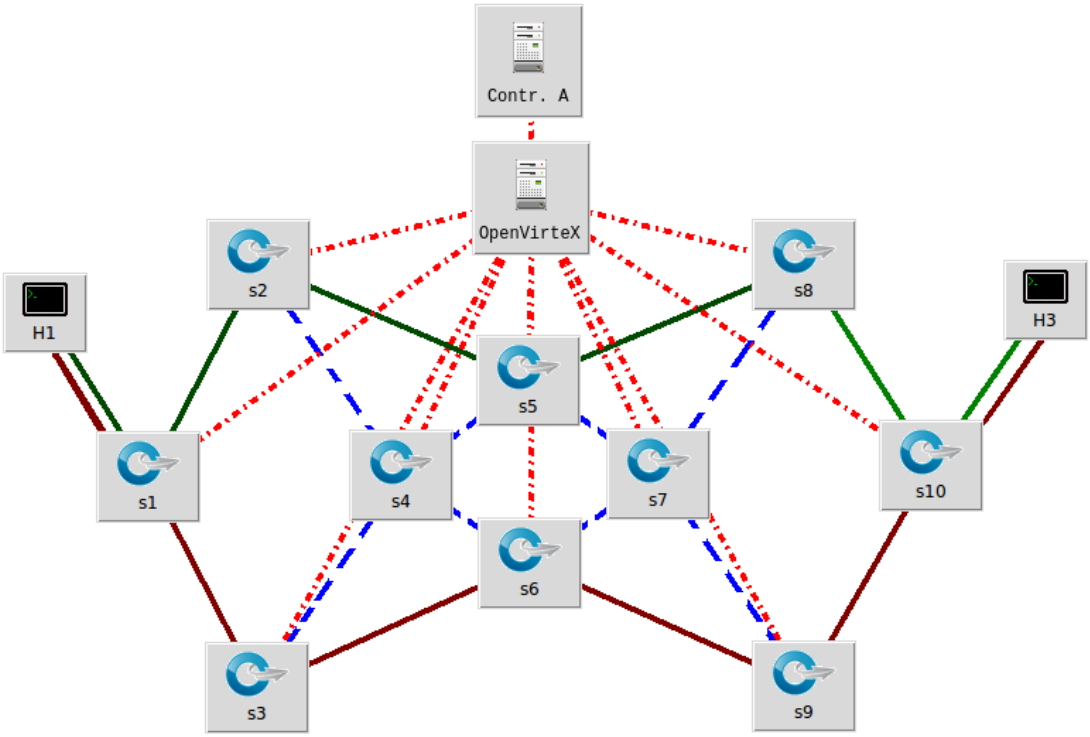
Regra	Switch	Campos do cabeçalho	Ação
1	Switch Virtual	in_port:1	out_port:3
2	Switch Virtual	in_port:3	out_port:1

Através desta topologia virtual criada, todo processo de configuração e operacionalização da rede, tarefa desempenhada pelo controlador *OpenFlow* o qual é realizado pelo usuário da rede virtual se torna mais simples. O processo de configuração de comunicação entre os *hosts* é menos complexo, pois o controlador tem a visão de uma rede com um *switch* apenas e 4 portas com 4 *hosts* conectados.

Já na infraestrutura física, o OVX controla por quais caminhos a informação percorrerá. Isso é feito através do algoritmo SPF (*Shortest Path First*) para a escolha da melhor rota entre a origem e o destino. Logo após a criação da rede virtual, o OVX define qual a melhor rota para a comunicação entre os *hosts*. Além de definir o melhor caminho, é armazenado ainda uma ou mais rotas secundárias para a comunicação, para o caso de falhar o caminho principal.

Para a comunicação entre o H1 e H3, o OVX definiu a rota na topologia física da comunicação. O melhor caminho foi definido entre os switches s1-s2-s5-s8-s10, grifado na cor verde na Figura 28. Ainda foi definido um caminho alternativo, o s1-s3-s6-s9-s10, em cor marrom na imagem. Na ocorrência de falhas nos enlaces em utilização, o OVX automaticamente faz a convergência para uma rota secundária, restabelecendo a comunicação sem a necessidade de nenhuma ação manual. Ainda, seu tempo de convergência é considerado baixo em virtude de que o OVX não necessita calcular novas rotas no momento da falha, pois já possui internamente rotas secundárias armazenadas.

Figura 28 – Visão global da infraestrutura de rede controlada pelo OVX



Fonte: Próprio autor.

Portanto, através da funcionalidade de virtualização o OVX cria redes virtuais resilientes para os usuários, com *failover* nativo e automático em caso de rupturas de enlaces. Na ocorrência de falhas, o próprio OVX se encarrega de restabelecer a comunicação das redes virtuais automaticamente, alterando o caminho dos fluxos para rotas secundárias.

Diferente do FV, onde é preciso definir qual o caminho por onde os fluxos percorrerão, no OVX isso não é necessário pois o usuário opera diretamente em um cenário virtual. É função do OVX definir os caminhos onde as comunicações acontecerão. A eleição do melhor caminho ocorre com base nas mesmas métricas de avaliação utilizadas pelo protocolo de roteamento dinâmico OSPF (*Open Shortest Path First*).

Logo, todo esse processo é feito internamente pelo OVX e totalmente transparente para o usuário da rede virtual, consequentemente os usuários não necessitam desenvolver soluções para falhas de conectividade, como é o caso da utilização de protocolos de

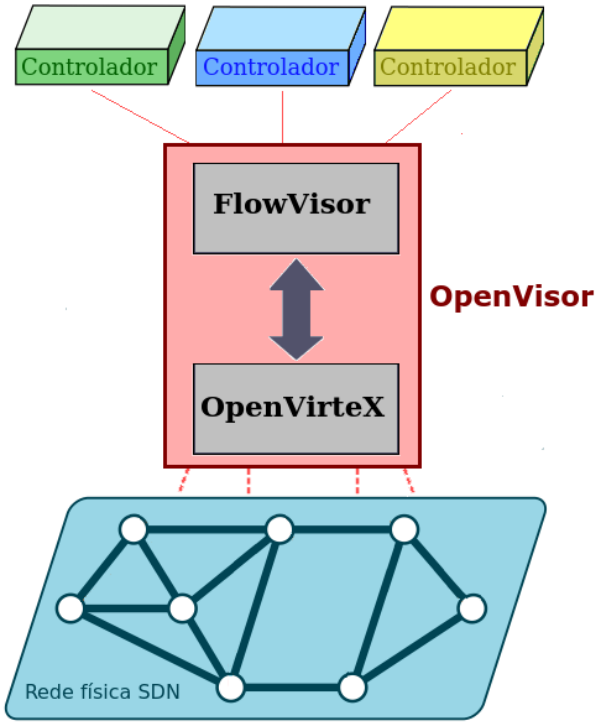
roteamento dinâmicos (OSPF, RIP, EIGRP, etc), pois a própria infraestrutura já possui esta característica de forma nativa.

*D*esta forma, cada rede virtual é identificada por uma faixa de IP única naquela infraestrutura, possibilitando, ainda, a sobreposição de IPs na infraestrutura física por outras redes virtuais.

7.3 INTEGRAÇÃO DAS FERRAMENTAS

A implementação da integração entre as ferramentas *OpenVirteX* e *FlowVisor*, foi denominada de *OpenVisor*. Os procedimentos definidos para o desenvolvimento da solução tomaram como base a utilização do OVX sobre a rede física SDN, comunicando-se diretamente com o FV que intermedeia as comunicações com os controladores *OpenFlow*, conforme é ilustrado na Figura 29.

Figura 29 – Infraestrutura de rede controlada pelo OpenVisor.



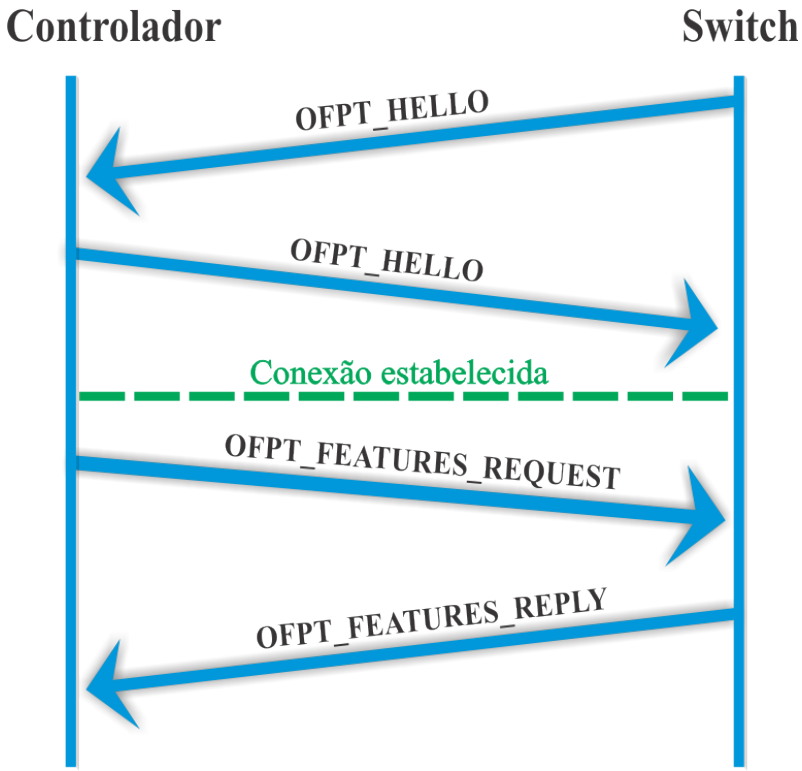
Fonte: Próprio autor.

Tanto o FV como o OVX são ferramentas que foram desenvolvidas com o propósito de intermediar as conexões dos comutadores e dos controladores *OpenFlow*, conforme denominado anteriormente como o funcionamento padrão de um *hypervisor* para Redes Definidas por *Software*. No entanto, de acordo com a documentação do FV e apresentado no trabalho de (YIN et al., 2013), é possível utilizá-lo na forma recursiva, ou seja, com mais de um *hypervisor* na mesma infraestrutura de rede simultaneamente. Desta forma, constatou-se que o FV não necessitaria de modificações internas para a realização da integração proposta. Em contrapartida, o OVX em sua implementação padrão não possui tal característica de recursividade e testes iniciais confirmaram que não era possível a utilização de outros *hypervisors* conectados ao OVX.

Portanto, para o desenvolvimento do *framework OpenVisor* foi necessário modificar e implementar a funcionalidade de recursividade no OVX, onde verificou-se que o OVX não conseguia estabelecer a conexão com o FV, pois identificava-o como um controlador *OpenFlow* convencional e, reconhecendo que não se tratava de um controlador, terminava a conexão acusando recebimento de mensagens ilegais, fechando aquele canal de comunicação com aquele “controlador”. Para compreender melhor este processo, foi necessário entender o processo estabelecimento de conexão do OVX com os controladores e, o comportamento do FV ao se conectar com os switches.

O processo de estabelecimento de um controlador *OpenFlow* convencional se dá da seguinte maneira: Inicialmente ocorre o processo “*3-way-handshake*” para estabelecimento da comunicação TCP o qual o protocolo *OpenFlow* opera. Então o *switch*, logo que conecta na porta TCP do controlador, envia uma mensagem do tipo OFPT_HELLO, conforme é ilustrado na Figura 30. O controlador após receber esta mensagem, envia também um OFPT_HELLO contendo a mensagem de retorno e ainda solicita informações sobre a versão do *OpenFlow* utilizada.

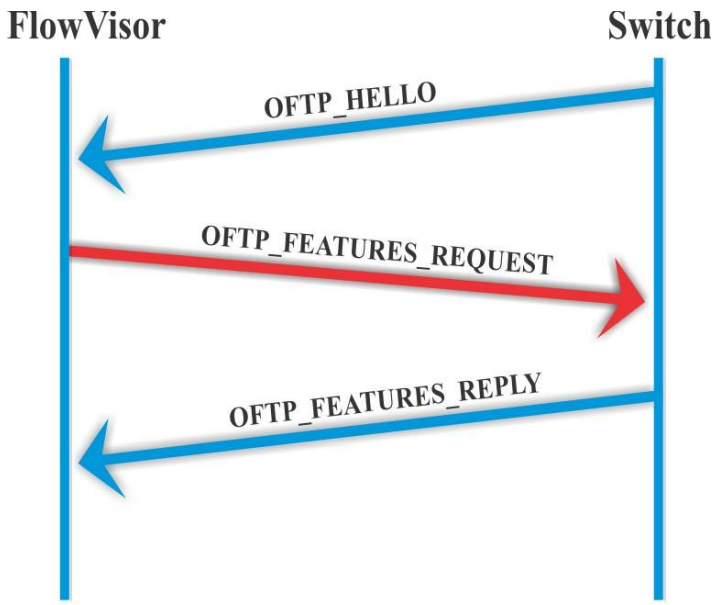
Figura 30 – Controlador OpenFlow estabelecendo conexão com o switch.



Fonte: Próprio autor.

Depois da conexão estabelecida com sucesso, o controlador envia uma mensagem OFPT_FEATURES_REQUEST. Por padrão, essa mensagem contém apenas informações no cabeçalho *OpenFlow* e não possui nada em seu corpo. O *switch* recebe a mensagem e responde com um OFPT_FEATURES_REPLY, enviando informações como seu DPID e suas capacidades. No entanto, analisando o comportamento do FV ao estabelecer a conexão com os switches, identificou-se que este processo diferenciava-se em alguns pontos. Inicialmente notou-se que o FV não enviava uma mensagem de OFPT_HELLO de retorno, conforme é visto na Figura 31, o que em teoria deveria fazer o *switch* não estabelecer a conexão com o FV.

Figura 31 – FlowVisor estabelecendo conexão com o switch.



Fonte: Próprio autor.

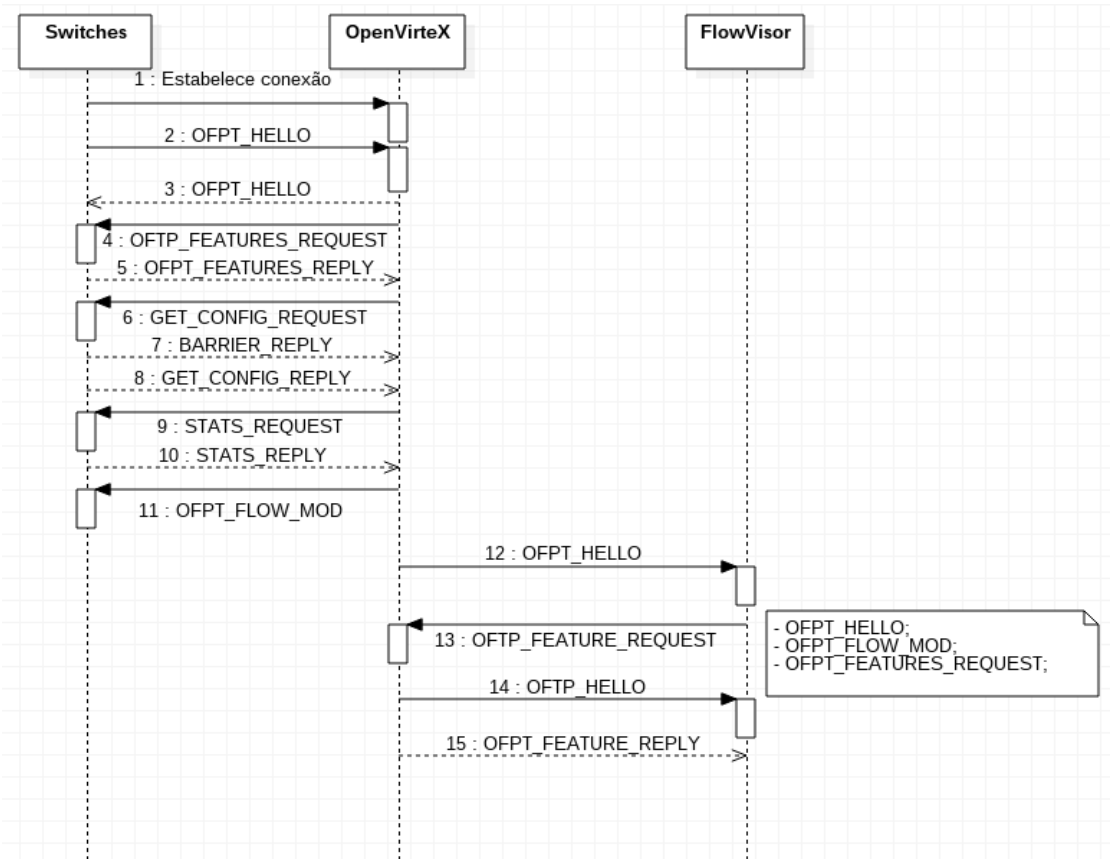
Analisando detalhadamente este processo, identificou-se que o FV ao receber a mensagem de "*Hello*" do *switch*, envia uma mensagem OFPT_FEATURES_REQUEST contendo três mensagens dentro, uma OFPT_HELLO, OFPT_FLOW_MOD e OFPT_FEATURES_REQUEST. Esta característica é prevista nas especificações do protocolo *OpenFlow*.

Essa mensagem *OpenFlow* composta por outras três, envia as seguintes informações e solicitações ao *switch*: Primeiro é estabelecido a comunicação *OpenFlow* com o *switch* através do "*hello*" de retorno; então é enviado uma instrução do tipo "*flow delete*" na mensagem OFPT_FLOW_MOD, solicitando ao *switch* apagar quaisquer regras armazenadas em suas tabelas de fluxos; e por fim a mensagem OFPT_FEATURES_REQUEST solicitando informações sobre suas capacidades, versão do *OpenFlow* suportada, entre outras.

Analisando o processo de estabelecimento de conexão do OVX com os controladores em sua implementação padrão, percebeu-se que o mesmo iniciava a comunicação enviando

uma mensagem OFPT_HELLO do tipo “*send_hello*” e ficava aguardando outra do tipo “*recv_hello*”, da mesma forma que para estabelecer a comunicação *OpenFlow*. No caso de não receber o OFPT_HELLO de retorno, é reencaminhado a mensagem inicial. A comunicação do OVX com os switches e o controlador, que neste caso é o FV, pode ser vista no diagrama de sequência apresentado na Figura 32.

Figura 32 – Diagrama de sequência do OVX estabelecendo conexão com os switches e o FV



Fonte: Próprio autor.

Nota-se que a comunicação do OVX com os switches acontece normalmente, estabelecendo a comunicação e ocorrendo a troca de mensagens corretamente. Porém quando o OVX tenta estabelecer conexão com o FV, a comunicação não é estabelecida pois o OVX não recebe a informação da qual necessita.

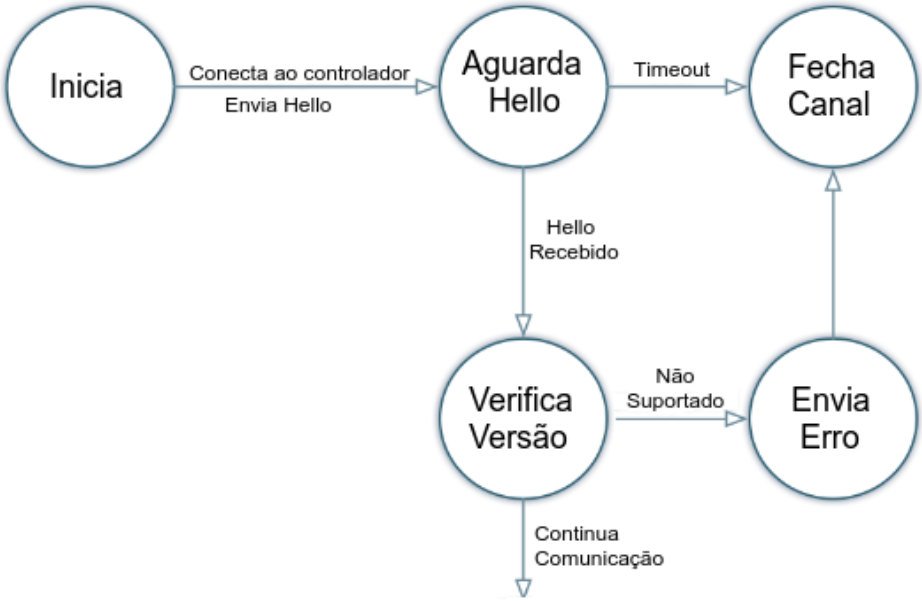
Desta forma, identificou-se que a falha de conectividade entre o OVX e o FV ocorria logo no estabelecimento da conexão, pois o FV enviava instruções de operação através do

OFPT_FEATURES_REQUEST antes mesmo de terminar o processo de estabelecimento de conexão OpenFlow. Para contemplar este objetivo, existiam duas alternativas: Adicionar a funcionalidade de recursividade no OVX modificando seu código-fonte para faze-lo entender as mensagens do FV para estabelecer a conexão entre ambos; ou então modificar a forma de operação do FV no estabelecimento de conexão com os switches.

Escolheu-se a primeira, adicionar a funcionalidade no OVX pelas seguintes razões: o OVX possui uma documentação bastante completa sobre sua arquitetura e modo de funcionamento, já o FV não. E também em virtude do FV não possuir documentação oficial, logo não saberia o impacto gerado com a modificação no estabelecimento de conexão com os switches, diferente do OVX que seu código-fonte era detalhado sobre seu funcionamento.

Para tanto, foi necessário modificar a classe *ControllerChannelHandler.java* responsável por este processo, ilustrada pelo fluxograma da Figura 33. A classe define o estabelecimento da conexão associando o estado de ativo com os controladores. No caso do OVX não receber a mensagem correta de retorno do controlador, o método *illegalMessageReceived* é chamado pela classe que por sua vez fecha o canal de comunicação com o controlador. Logo mensagens diferentes da esperada (*recv_hello*) eram consideradas ilegais a conexão com o controlador era finalizada.

Figura 33 – Fluxograma do estabelecimento de conexão do OVX com os controladores.



Fonte: Próprio autor.

Então foi necessário modificar a classe onde era definido que mensagens diferentes da esperada (*recv_hello*) eram consideradas ilegais. Definiu-se que mensagens do tipo OFPT_FEATURES_REQUEST deveriam ser analisadas antes de ser consideradas ilegais. Logo além de mensagens OFTP_HELLO o OVX passou a aceitar, com algumas condições, mensagens do tipo OFPT_FEATURES_REQUEST, conforme mostra o pseudocódigo do Algoritmo 1.

Algoritmo 1: Pseudocódigo do algoritmo de estabelecimento de conexão OVX-FV

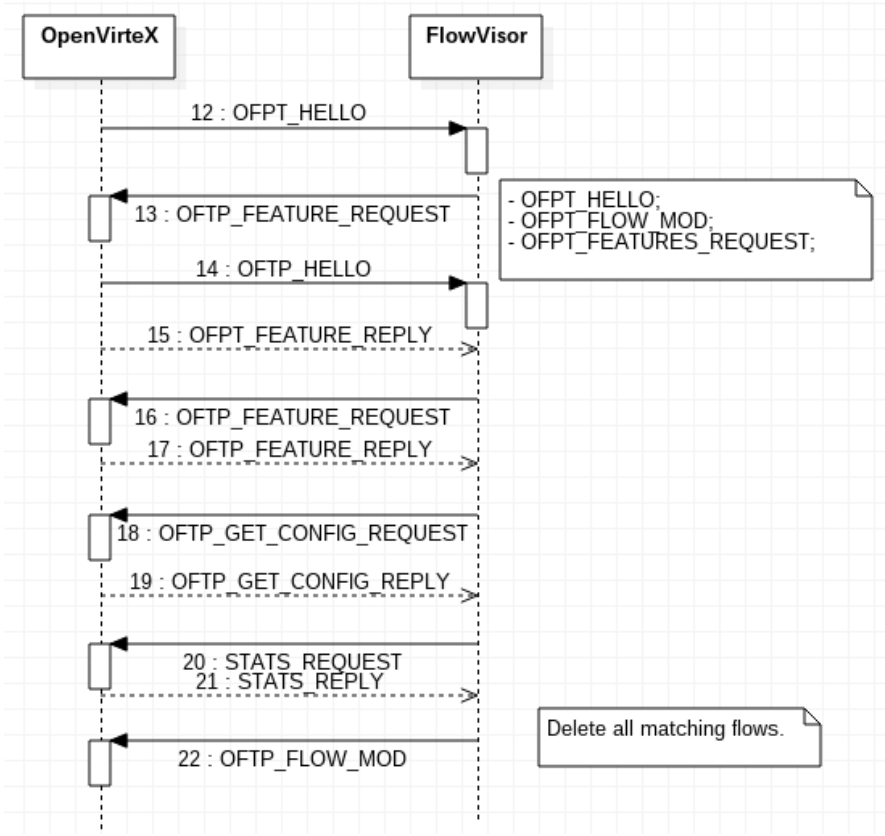
Inicialização: Inicialmente, o OVX estabelece conexão apenas com controladores que enviam a mensagem do tipo OFPT_HELLO. Definiu-se que quaisquer mensagens diferente do tipo OFHello e OFFeaturesRequest são consideradas ilegais e o OVX fecha a conexão com o controlador. M é uma constante que indica a mensagem OpenFlow recebida.

```
início
    OFMessage M;
    enquanto Handshake == 0 faça
        M ←— mensagem_OpenFlow;
        se M ==OF_HELLO então
            Handshake ←— 1;
            Define estado como OPFT_FEATURES_REQUEST;
        fim
        se (M ==OF_FEATURES_REQUEST e
         corpo_mensagem ==OF_HELLO e OFFlowMod) então
            Defina o estado do canal como
             OFTP_FEATURES_REPLY
            Handshake ←— 1;
        fim
        se M! = (OF_FEATURES_REQUEST e OF_HELLO) então
            Chamar método illegalMessageReceived() fim
        fim
        fim
```

Verificando que a mensagem recebida seja um OFPT_HELLO, a comunicação é continuada normalmente e o *handshake* é estabelecido. Porém recebendo uma mensagem do tipo OFPT_FEATURES_REQUEST em vez de aplicar o método *illegalMessageReceived*, conforme sua implementação padrão, é realizado uma análise no corpo da mensagem para verificar se existem outras mensagens *OpenFlow*, especificamente o OFTP_HELLO e o OFPT_FLOWMOD. Atendendo estas condições a comunicação não é finalizada e o *handshake* é definido forçadamente como estabelecido, onde a comunicação continua ativa até ser processado as informações recebidas. Depois é enviado ao controlador a mensagem de OFPT_FEATURES_REPLY. Não atendendo as condições implementadas na mensagem OFPT_FEATURES_REQUEST, o método *illegalMessageReceived* considera aquela mensagem como ilegal e finaliza a comunicação com aquele controlador.

Após a alteração na classe *ControllerChannelHandler.java*, o OVX modificado conseguiu estabelecer conexão com o FV mesmo sem, de fato, receber o *OFPT_HELLO* de retorno, conforme mostra o diagrama de sequência na Figura 33. No início o OVX recebe uma mensagem de retorno diferente da "*recv_hello*" esperada e sua implementação padrão define um reenvio do "*hello"* inicial. Enquanto isso a mensagem OFTP_FEATURES_REQUEST proveniente do FV é analisada pela implementação detalhada no Algoritmo 1.

Figura 34 – Diagrama de sequência do OVX modificado.



Fonte: Próprio autor.

Logo após o OVX enviar o segundo "*send_hello"* a mensagem recebida do FV já foi examinada e consequentemente o *handshake* é estabelecido. A partir disso, o OVX e o FV

passam a se comunicar normalmente, iniciando a troca de mensagens com o *OFPT_FEATURES_REPLY*, resposta a mensagem do FV.

Portanto, foi possível a integração do FV a partir da modificação realizada no OVX, cujo foi nomeado *OpenVirteX* Recursivo (OVX^R) em virtude das funcionalidades criadas após a implementação. A partir da integração finalizada, passou-se a realização de um conjunto de testes de validação da solução proposta.

8 AVALIAÇÃO DO FRAMEWORK

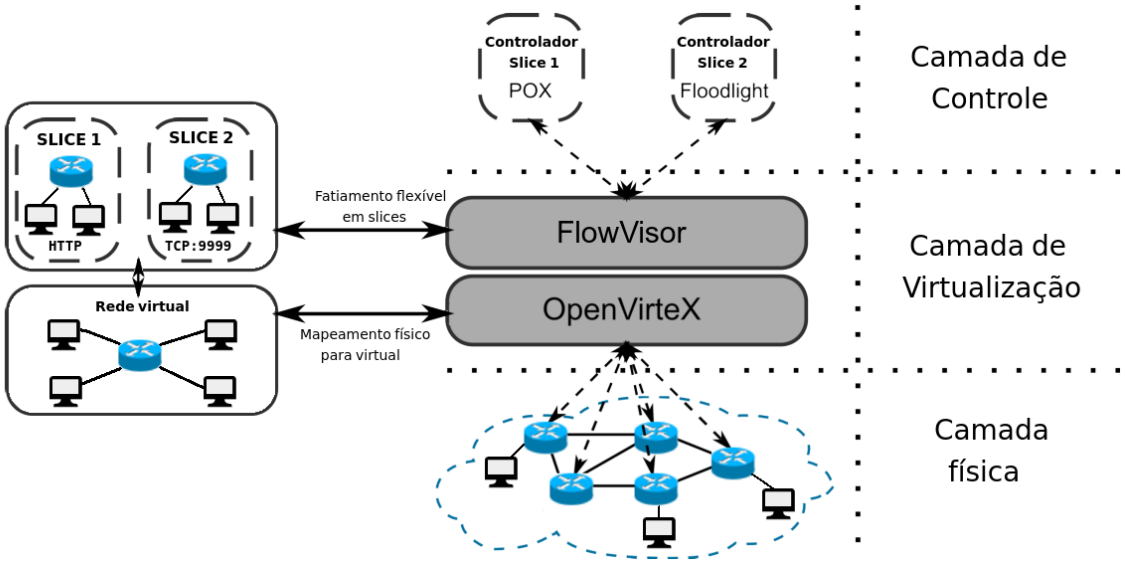
Neste capítulo serão descritos os experimentos realizados para verificar e validar as funcionalidades esperadas no *framework* proposto. Analisa-se, primeiramente o funcionamento inicial da proposta e, posteriormente apresenta-se o conjunto de testes realizados para avaliar o comportamento da ferramenta. A avaliação foi divida em três categorias, conforme é apresentado a seguir:

- **Testes de funcionamento:** Estes testes objetivaram demonstrar o funcionamento do *OpenVisor* com a pretensão de analisar o comportamento do *framework* sobre diferentes tipos de comunicações reais. Os testes realizados para identificar o funcionamento foram: teste de comunicação básica, construção de tabela de fluxos, configuração de um servidor web e transmissão de vídeo via rede através do Media Player. Os testes foram baseados no estudo de (PUPATWIBUL, 2016) onde buscou-se analisar o comportamento do *framework* sobre diferentes tipos de comunicações reais.
- **Testes de funcionalidades:** Esta categoria de testes teve o objetivo de averiguar se a integração proposta trouxe a união das funcionalidades das ferramentas utilizadas. Para tanto, realizou-se os seguintes testes*:* Virtualização de Topologia; Recuperação de Falhas; Flexibilidade na definição das redes; e Controle absoluto da rede virtual. Os testes realizados tomaram como parâmetro as funcionalidades descritas em cada um dos *hypervisors*, no qual baseou-se nos estudos de (AL-SHABIBI, 2014) e (SHERWOOD et al., 2009), que descrevem as características dos *hypervisors*.
- **Comparações**: nesta etapa, buscou-se avaliar o *framework* desenvolvido comparando com as seguintes ferramentas: *FlowVisor*, *OpenVirteX* e *VeRTIGO*. Foram utilizados como métricas de análise os seguintes critérios: quantidade de fluxos, de modo a verificar a complexidade de operação do *OpenVisor*; desempenho objetivando analisar diferenças consideráveis com as demais ferramentas; e tempo para a recuperação de falhas, de modo a avaliar o tempo gasto na convergência de rotas. Utilizou-se como referência as normas RFC 9126, 2544 e os trabalhos de (SCHWARZ, 2014), (PUPATWIBUL, 2016) e (CORIN et al., 2012).

A integração contemplou a união do *hypervisor OpenVirteX* e do *FlowVisor*, primando por garantir a funcionalidade de ambas as ferramentas. A recursividade das ferramentas foi o

desafio da implementação da proposta, exigindo adicionar a funcionalidade de recursividade sobre o OVX, de modo a tornar possível utilizá-lo junto com o FV sobre a mesma infraestrutura de rede, objetivando a soma das características de ambos. Portanto, definiu-se a utilização do OVX^{R} agindo no controle da infraestrutura física. Já o FV foi definido na camada superior ao OVX^{R}, conforme ilustra a Figura 35.

Figura 35 – Processo de conexão entre o OVX e o controlador.



Fonte: Adaptado de (Al-Shabibi et al., 2014).

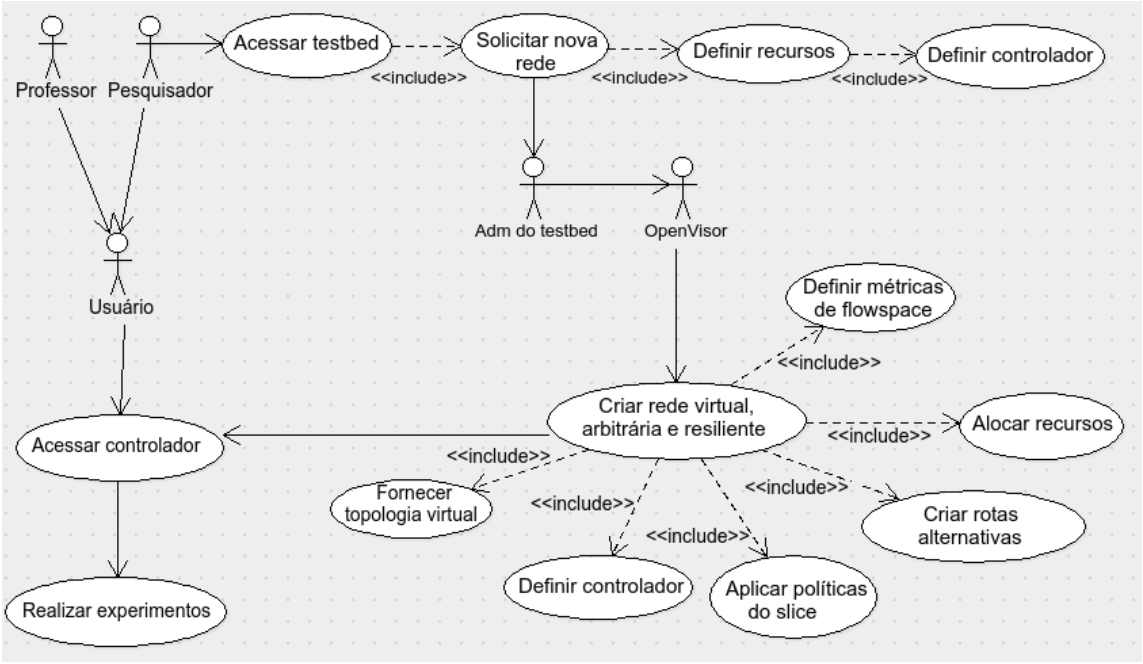
O OVX^{R} no controle de toda camada física da infraestrutura, cria uma rede virtual e arbitrária com um *switch* virtual que faz referência a todos dispositivos e *links* da infraestrutura física. Todos os *hosts* são conectados a este *switch* virtual e é definido como controlador desta rede virtual o FV. Dessa maneira, o OVX cria uma rede com topologia virtual e arbitrária, de baixa complexidade de operação e repassa o controle para o FV.

O FV, por sua vez, passa a operar diretamente sobre a rede virtual criada pelo OVX. Considerando que o OVX possui a funcionalidade de criar redes virtuais resilientes, a rede cujo o FV controla passa a possuir a característica nativa de recuperação de falhas. Logo,

tanto o FV quanto as camadas superiores passam a operar sobre uma rede virtual, arbitrária e resiliente, desconhecendo a infraestrutura física o qual operam.

O papel do FV no *OpenVisor* é adicionar a flexibilidade na segmentação dos slices, característica não existente no OVX. Através do FV, é possível segmentar a rede criada pelo OVX, criando *slices* com alta flexibilidade, através da definição de métricas do cabeçalho *OpenFlow* para cada rede.

Figura 36 – Diagrama de casos de uso do OpenVisor.



Com relação ao funcionamento do *OpenVisor*, o diagrama de atividades representado na Figura 36 descreve o modo de funcionamento do ambiente no processo de definição de uma nova rede virtual. O diagrama inicia com a solicitação do pesquisador por uma rede virtual, onde é repassada ao administrador do *OpenVisor*, que cria a rede com base nas informações passadas pelo usuário. Fonte: Próprio autor.

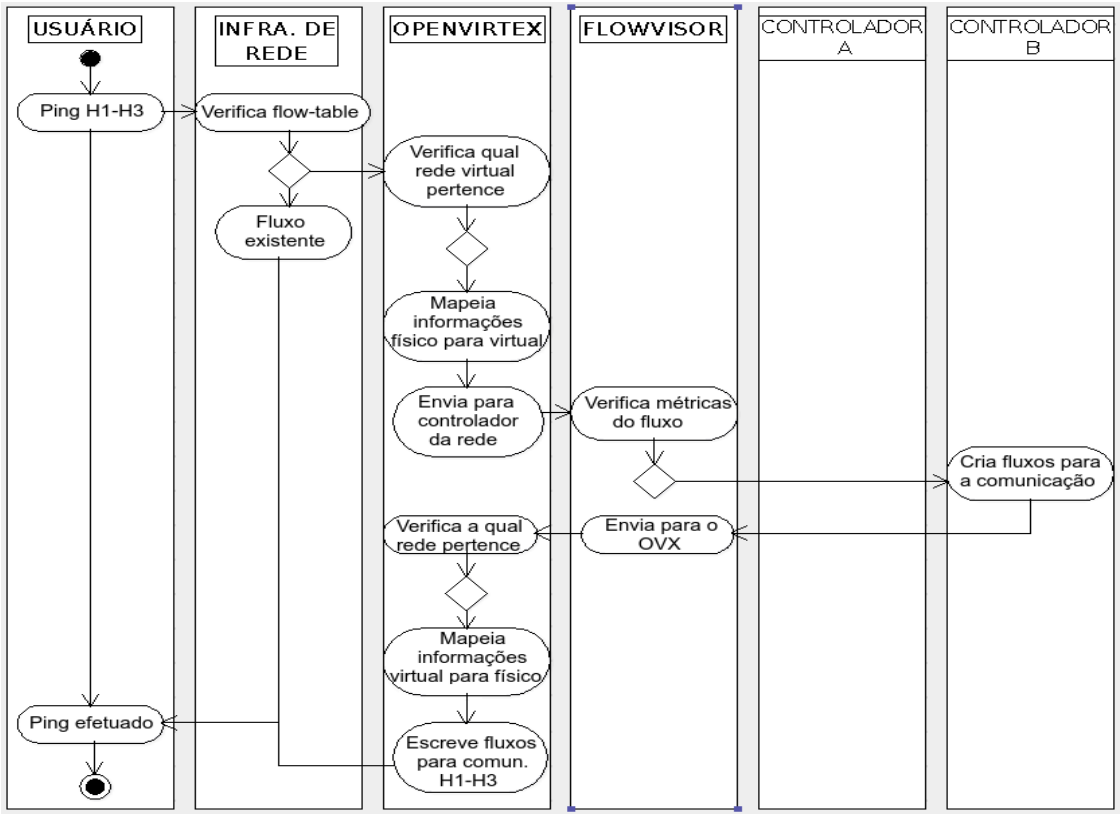
O diagrama mostra o processo de solicitação de uma nova rede por parte do usuário (pesquisador ou professor). Este, deve solicitar uma nova rede virtual e definir quais recursos, serão atribuídos para esta rede e quais métricas do *flowspace* serão utilizadas. Feito isso, o

usuário deve passar o endereço do controlador que controlará a rede virtual. Destaca-se ainda no diagrama, as características que são fornecidas ao usuário, a partir da ferramenta criada.

Todas essas informações são passadas para o administrar do *testbed* que então, cria uma nova rede virtual, com base nas informações passadas pelo usuário. É criado uma rede com um *switch* virtual, igual para todas as redes virtuais existentes, diferenciando-se apenas em relação aos *hosts* e a política de *flowspace*. Desta forma, o usuário passa a atuar sobre uma topologia virtual desconhecendo totalmente a infraestrutura física, operando sobre uma rede que possui nativamente recursos de recuperação de conectividade na ocorrência de falhas na topologia física.

Criado a rede virtual, o usuário já pode realizar seus experimentos através do manuseio do controlador *OpenFlow*. Através do controlador, o usuário pode configurar as comunicações entre os *hosts* com uma topologia simples e de fácil operacionalização. O diagrama de atividades representado na Figura 37 descreve um exemplo de comunicação entre dois *hosts* no *OpenVisor.*

Figura 37 – Diagrama de atividades do OpenVisor.



Fonte: Próprio autor.

A comunicação é iniciada pelo usuário através de um dos *hosts* da rede virtual. Neste caso foi iniciado uma comunicação do tipo *ping* entre os *hosts* H1 e H3. Essa comunicação chega nos switches da rede física, denominado no diagrama como infraestrutura física, que por sua vez verificam se aquela determinada comunicação possui uma regra de ação em suas tabela de fluxos. Caso possuam, o *switch* simplesmente atribui a ação a qual a regra define e efetua o *ping* em questão. Caso este padrão de fluxo não possua uma ação estabelecida nos comutadores, então o *switch* envia ao seu controlador, que neste caso é o OVX, em busca de instruções em como operar aquele determina tráfego. Nesta situação, o OVX recebe o fluxo e verifica a qual rede virtual pertence aquele fluxo, para então repassar ao controlador responsável por aquela rede virtual. Antes de encaminhar ao controlador, o OVX realiza um mapeamento das informações físicas para os atributos virtuais da topologia virtual e arbitrária criada. Visto que o controlador da rede virtual criada pelo OVX é o FV, todos os fluxos da

infraestrutura física da rede são repassados ao FV sobre a perspectiva da rede virtual e arbitrária criada pelo OVX.

O FV então verifica a qual *slice* o fluxo pertence, através de suas métricas do cabeçalho e envia ao controlador *OpenFlow* do rede criada pelo FV. Caso o fluxo não pertença a nenhum *slice*, o FV descarta a comunicação. Desta forma, os controladores *OpenFlow* só recebem exatamente os fluxos ao qual controlam e que estão configurados em seu *flowspace*. Encontrado a qual *slice* o pacote pertence, o FV envia ao controlador do *slice*, neste caso, o Controlador B.

Chegado o fluxo ao controlador, é tarefa do usuário, cujo gerencia o controlador, definir as políticas daquele tráfego de acordo com seus interesses, criando assim as regras a serem instaladas na infraestrutura de rede. Estas informações são encaminhadas ao FV que repassa para o OVX. O OVX, novamente verifica a que rede virtual pertence e faz conversão das características virtuais para as físicas. Após, escreve os fluxos necessários para possibilitar a comunicação entre os hosts H1-H3, conforme instrução determinada pelo Controlador B.

A partir da integração do OVX e do FV, concluiu-se no *framework OpenVisor* proposto, resultando numa ferramenta com funcionamento aparentemente adequado e viável em um contexto de ambientes de experimentação *OpenFlow*.

8.1 TESTES DE FUNCIONAMENTO DO OPENVISOR

Para validar o funcionamento de uma rede com o *OpenVisor*, um conjunto de testes foram realizados, com base no estudo de (PUPATWIBUL, 2016), a fim de analisar o comportamento do *framework* desenvolvido em um ambiente simulado (Mininet) porém com diferentes tipos de aplicações reais e práticas. Os experimentos realizados foram os seguintes: comunicação básica viabilizada pelo *OpenVisor*; Construção da tabela de fluxos: a utilização do comando *ping* entre os *hosts* e verificação da comunicação pelo protocolo ARP; criação de um servidor *web* onde um cliente envia requisições de acesso; transferência de vídeo pela rede entre dois *hosts*.

8.1.1 Teste de Comunicação Básica

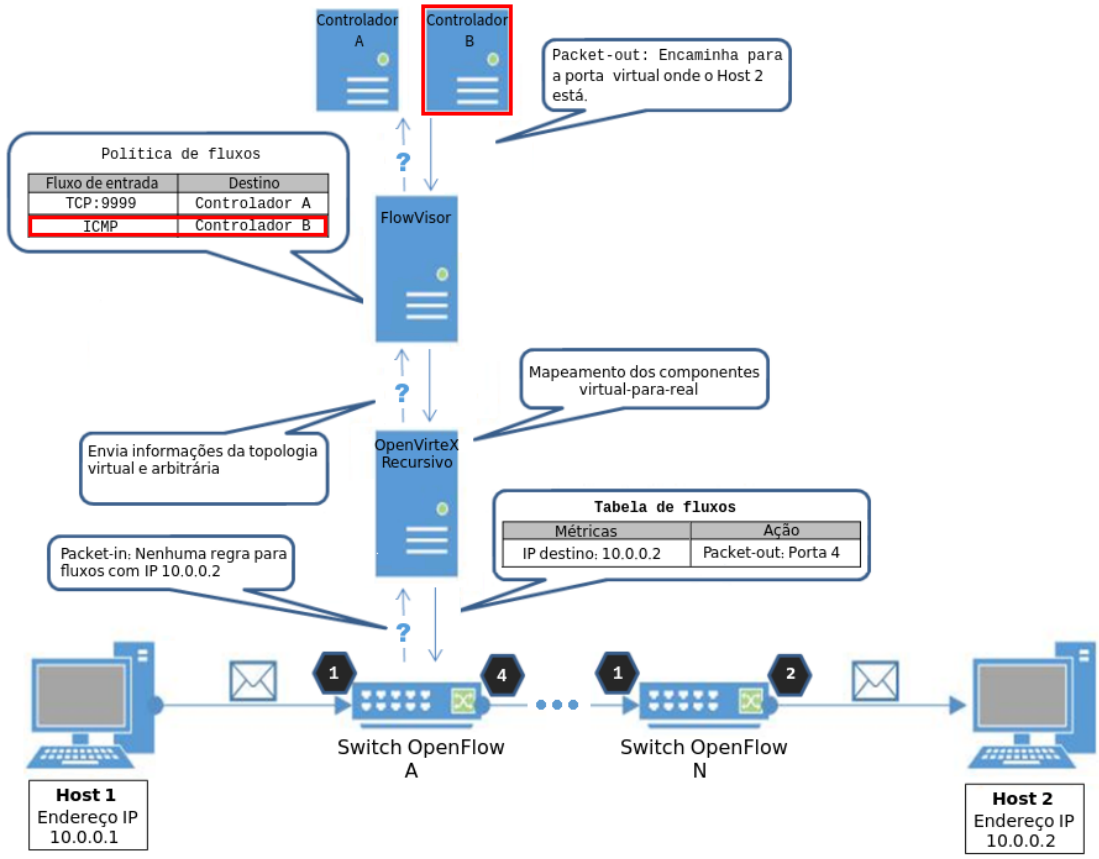
O teste que segue, objetiva demonstrar como as comunicações ocorrem em uma infraestrutura de rede controlada pelo *OpenVisor*, desde a saída do fluxo do *host* de origem até o destino. Considerou-se uma comunicação de um *ping* do *Host* 1 para o *Host* 2, independente da infraestrutura de rede.

A comunicação inicia-se do *Host* 1 para o *Host* 2, passando pelos switches A até o N, referenciando-se a uma rede composta por N switches. Quando o pacote chega ao *switch* A, que não tem nenhuma entrada de fluxo, ele envia uma mensagem de *OFPT_Packet-in* para seu controlador, que neste caso é o *framework OpenVisor*, mais especificamente a instância do OVX^R, em busca de instruções para operar com aquele fluxo.

O OVX^R recebe a mensagem e verifica a qual rede virtual pertence, de acordo com o IP virtual encontrado no cabeçalho da mensagem. Posteriormente ele realiza o mapeamento dos componentes físico-para-virtual e, então envia o *OFPT_Packet-in* para o controlador da rede virtual e arbitraria, neste caso o FV, que então repassa a mensagem para o controlador de acordo com as métricas do cabeçalho daquele fluxo.

A figura 38 expressa o detalhamento da comunicação efetuada:

Figura 38 – Comunicação entre H1 e H2 através do OpenVisor.



Fonte: autoria própria

Como se evidencia na figura, a informação chega até o controlador B, grifado em vermelho, que então determina qual ação a infraestrutura de rede (switches) deve tomar para este tipo de fluxo. Neste exemplo, o controlador define que o fluxo deverá percorrer até a *interface* onde o *Host* 2 está conectado na topologia virtual e arbitrária conforme sua perspectiva, respondendo com uma mensagem de *OFPT_Packet-out* para o FV.

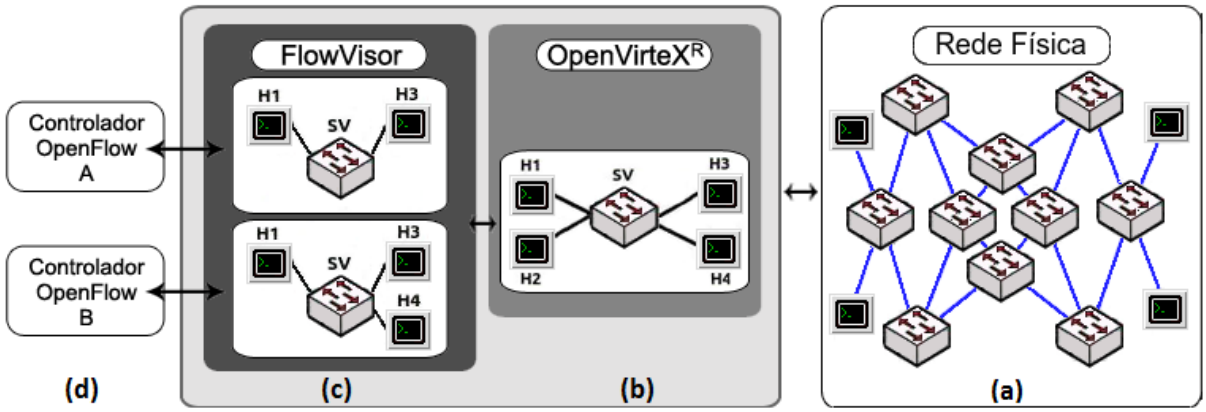
O FV encaminha o pacote até o OVX^{R} que verifica que é uma mensagem *OFPT_Packet-out* cuja ação é encaminhar o fluxo até o *Host* 2, porém na perspectiva da topologia virtual. Portanto, ele realiza a conversão novamente, porém agora inversa, da topologia virtual para a infraestrutura física e cria as regras de fluxos necessárias nos switches da rede para permitir a comunicação entre os *hosts* em questão.

8.1.2 Construção da tabela de fluxos

A fim de mostrar o processo de instalação de fluxos nos comutadores e a comunicação intermediada pelo *OpenVisor* até chegar aos controladores *OpenFlow*, um teste foi executado onde o *host* 1 realiza uma comunicação através do utilitário *ping* com o *host* 3, utilizando, nesta experimentação um cenário específico. O processo é apresentado, a partir dos pacotes de dados e de controle que são enviados para cada *host*, iniciando por uma comunicação *ARP request/reply* e posteriormente os fluxos ICMP *echo_request/reply* entre as duas extremidades.

Utilizou-se o cenário de experimentação criado para implementação do protótipo, contendo 10 switches e 4 *hosts*, os quais foram conectados ao OVX^{R}, conforme é ilustrado na Figura 39 em (a).

Figura 39 – Cenário de experimentação com o *framework OpenVisor*.



Fonte: Próprio autor.

Utilizou-se o OVX^{R} para dispor da virtualização e abstração da infraestrutura física, criando uma rede virtual composta por uma topologia arbitrária de apenas um único *switch* virtual denominado “SV” (b), cujo abstrai todos os switches *OpenFlow* da rede física, bem como seus *enlaces*. Ainda, são conectados os quatro *hosts* nesta rede virtual. Definida a rede virtual, foi configurado o FV como controlador desta rede que então recebe as informações do OVX^{R} e possui apenas a perspectiva da rede virtual, visualizando apenas o *switch* virtual e os quatro *hosts*, desconhecendo totalmente os componentes da rede física. Sua função é

segmentar a rede virtual criada (b) em *slices* adicionando a flexibilidade no fatiamento, atribuindo a cada rede um espaço de endereçamento de *flowspace*. É criado então dois *slices*, um contendo os *hosts* H1 e H3 e o outro com o H1, H3 e H4 (c), ambos definidos para controladores *OpenFlow* diferentes (d). Foram definidos políticas de *flowspace* distintas para ambos *slices*, onde o primeiro foi configurado que somente tráfego do protocolo TCP com porta 9999 e proveniente dos *hosts* H1 e H3. Já o segundo *slice* foi configurado para quaisquer fluxos provenientes dos hosts H1, H3 e H4.

O primeiro recebeu a prioridade “100”, enquanto o segundo foi definido como prioridade “1”. Portanto, um fluxo entre H1 e H3 assumindo as métricas TCP porta 9999 poderia ser repassado para quaisquer *slices*, visto que se enquadra nas políticas de ambos. Porém neste caso a eleição do *slice* é feita através da prioridade mais alta que no caso é para o primeiro *slice*, consequentemente para o controlador A.

Por fim, os controladores são conectados ao FV que intermedeia as comunicações com o OVX^{R}. Cada controlador é de responsabilidade do usuário do *testbed* e o gerenciamento e a utilização fica de acordo com seus critérios e requisitos.

A análise foi realizada seguindo a sequência lógica de uma comunicação sobre o contexto de SDN, que contemplou primeiramente a identificação dos dados, a partir do número de sequência dos pacotes enviados entre os *hosts*. O segundo aspecto a ser analisado definiu-se pacotes de controle, referindo-se ao número de sequência de pacotes de controle enviados entre o *OpenVisor* e os controladores. Identificou-se, ainda a origem e o destino do pacote, bem como a descrição deste. O Quadro 6 detalha os elementos expressos:

Quadro 6- Métricas de análise da comunicação.

Dados	**Pacotes de controle**	**Origem**	**Destino**	**Conteúdo**
Número de sequência dos pacotes enviados entre os hosts.	Número de sequência de pacotes de controle enviados entre o *OpenVisor* e os controladores	Origem do pacote	Destino do pacote	Descrição do conteúdo do pacote.

Neste experimento, quando o comando *ping* é executado do *host* 1 para o *host* 3, o mecanismo de encaminhamento para pacotes com destino desconhecido é demonstrado pelo processo de “*ARP request*” a seguir:

- **ARP request**

Quadro 7- Início da comunicação ARP Request

Dados	**Pacotes de controle**	**Origem**	**Destino**	**Conteúdo**
1		*Host* 1	*Broadcast*	ARP: Quem é o 10.0.0.3?

Uma solicitação “ARP *Request*” é realizada pelo *host* 1 com destino para o endereço FF:FF:FF:FF:FF:FF, objetivando descobrir o endereço MAC do IP 10.0.0.3 através de um *broadcast* para rede dentro daquele domínio.

Quadro 8- Switch encaminha o fluxo com o ARP Request ao OVX^{R}

Dados	**Pacotes de controle**	**Origem**	**Destino**	**Conteúdo**
	1a	*Switch* 1	*OpenVirteX*^{R}	*OFTP_Packet-In*: Pacote 1 encapsulado (ARP *request*)

Então, chegando no comutador este fluxo, o *Switch* 1 verifica que não possui regras da tabela de fluxos e não sabe como encaminhar esse pacote. Ele então envia um *packet-in* que contém o ARP-*request* encapsulado para o OVX^{R}.

Quadro 9- OVX^{R} encaminha o ARP Request ao FV

Dados	**Pacotes de controle**	**Origem**	**Destino**	**Conteúdo**
	1b	*OpenVirteX*^{R}	*FlowVisor*	*OFTP_Packet-In*: Pacote 1 encapsulado (ARP *request*)

O OVX^{R}, apenas intermediando a comunicação, recebe o fluxo e encaminha para o controlador da rede virtual, no caso o FV.

Quadro 10- FV encaminha o ARP Request ao Controlador B

Dados	**Pacotes de controle**	**Origem**	**Destino**	**Conteúdo**
	1c	*FlowVisor*	Controlador B	*OFTP_Packet-In*: Pacote 1 encapsulado (ARP *request*)

Da mesma forma que o OVX^{R}, o FV apenas intermedeia a comunicação, portanto, recebe e encaminha para o controlador *OpenFlow* cujo responde por aquele *flowspace*. Neste caso, foi encaminhado o *ARP request* encapsulado para o controlador B.

Quadro 11- Controlador B define a política de encaminhamento para o fluxo ARP Request

Dados	**Pacotes de controle**	**Origem**	**Destino**	**Conteúdo**
2		Controlador B	*FlowVisor*	ARP: Quem é o 10.0.0.3?

O controlador B envia um *packet-out* para o FV, que em sua perspectiva é a infraestrutura de rede, com a ação de enviar o *ARP request* para todas as portas do *switch*, com exceção da porta de ingresso do fluxo.

Quadro 12- FV encaminha a mensagem do Controlador B ao OVX^{R}

Dados	**Pacotes de controle**	**Origem**	**Destino**	**Conteúdo**
	2a	*FlowVisor*	*OpenVirteX^{R}*	*OFPT_Packet-out*: Pacote 2 encapsulado (ARP *request*)

O FV novamente apenas encaminha a mensagem de *packet-out* do controlador B para o OVX^{R}.

- **ARP reply**

Quadro 13- OVX^{R} envia o ARP Reply para o FV

Dados	**Pacotes de controle**	**Origem**	**Destino**	**Conteúdo**
3		OpenVirteX^{R}	FlowVisor	ARP: Eu sou o 10.0.0.3 e meu MAC é o 00-00-00-00-00-13.

Chegado no OVX^{R} o fluxo, ele mesmo responde a solicitação se identificando como sendo o *Host* 3 e informando o endereço MAC, visto que ele é o responsável por todas as comunicações com a infraestrutura física e detêm o conhecimento da totalidade da rede, tanto no aspecto virtual quanto real. Desta forma, o OVX^{R} não realiza o *broadcast* solicitado pelo controlador para encontrar o MAC do *host* 3.

Quadro 14- FV encaminha o ARP *Reply* ao Controlador B

Dados	**Pacotes de controle**	**Origem**	**Destino**	**Conteúdo**
	3a	*FlowVisor*	Controlador B	*OFTP_Packet-In*: Pacote 3 encapsulado (ARP *reply*)

O FV novamente apenas encaminha o fluxo encapsulado ao controlador B contendo o *ARP-reply* enviada pelo OVX^{R}.

Quadro 15- Controlador B define a política de encaminhamento para fluxos ARP

Dados	**Pacotes de controle**	**Origem**	**Destino**	**Conteúdo**
4		Controlador B	*FlowVisor*	*OFPT_Flow-mod*: Prot: ARP, Src: 00-00-00-00-00-11, Dst: 00-00-00-00-00-13, IN = 1 → OUT = 3

O controlador B após descoberta da localização do *host* 3, envia uma instrução ao *switch* para instalar um novo fluxo e como encaminhar pacotes similares a este. Logo, pacotes com endereço MAC de origem 00-00-00-00-00-11 e destino 00-00-00-00-00-13, com fluxo de entrada pela porta 1 deve ser encaminhados para a porta 3.

Quadro 16- FV encaminha o fluxo do Controlador B para o OVX^{R}

Dados	**Pacotes de controle**	**Origem**	**Destino**	**Conteúdo**
	5a	*FlowVisor*	*OpenVirteX^{R}*	*OFPT_Packet-out*: Pacote 4 encapsulado (*Flow-mod*)

O FV envia a solicitação 2c do controlador B para o OVX^{R}. O *host* 1 já pode preencher sua tabela ARP após ter recebido a resposta do ARP-*request*, consequentemente já pode enviar o pedido de ICMP *echo request*.

- **ICMP echo request**

Quadro 17- Início da comunicação ICMP echo request entre H1 e H3

Dados	**Pacotes de controle**	**Origem**	**Destino**	**Conteúdo**
1		*Host* 1	*Host* 3	ICMP *echo request*

Host 1 envia um pacote ICMP *echo request* para o *host* 3.

Quadro 18- Fluxo chega no *Switch* 1 e encaminha ao OVX^{R}

Dados	**Pacotes de controle**	**Origem**	**Destino**	**Conteúdo**
	1a	*Switch* 1	*OpenVirteX*^{R}	*OFPT Packet-in*: Pacote 1 encapsulado (ICMP *echo request*)

Chegando no *switch* 1 o fluxo, o comutador não sabe como encaminhar este pacote e envia-o na como *packet-in* ao OVX^{R}.

Quadro 19- OVX^{R} encaminha o fluxo para o FV

Dados	**Pacotes de controle**	**Origem**	**Destino**	**Conteúdo**
	2a	*OpenVirteX*^{R}	*FlowVisor*	*OFPT_Packet-in*: Pacote 1 encapsulado (ICMP *echo request*)

O OVXR encaminha ao FV a mensagem encapsulada de ICMP *echo request*.

Quadro 20- FV encaminha o fluxo para o Controlador B

Dados	**Pacotes de controle**	**Origem**	**Destino**	**Conteúdo**
	3a	*FlowVisor*	Controlador B	*OFPT_Packet-in*: Pacote 1 encapsulado (ICMP *echo request*)

O FV encaminha ao controlador B a mensagem encapsulada.

Quadro 21- Controlador B define a política de encaminhamento para fluxos ICMP

Dados	**Pacotes de controle**	**Origem**	**Destino**	**Conteúdo**
4		Controlador B	*FlowVisor*	*OFPT_Flow-mod*: Prot: ICMP Src: 00-00-00-00-00-11, Dst: 00-00-00-00-00-

				13, IN = 3 → OUT = 1

Novamente, o controlador B solicita a instalação de um novo fluxo e como encaminhar tráfegos similares a este, porém agora com protocolo ICMP.

Quadro 22- FV encaminha o fluxo do Controlador B para o OVX^{R}

Dados	**Pacotes de controle**	**Origem**	**Destino**	**Conteúdo**
	4a	*FlowVisor*	*OpenVirteX^{R}*	*OFPT Packet-out*: Pacote 4 encapsulado (ICMP *echo request*)

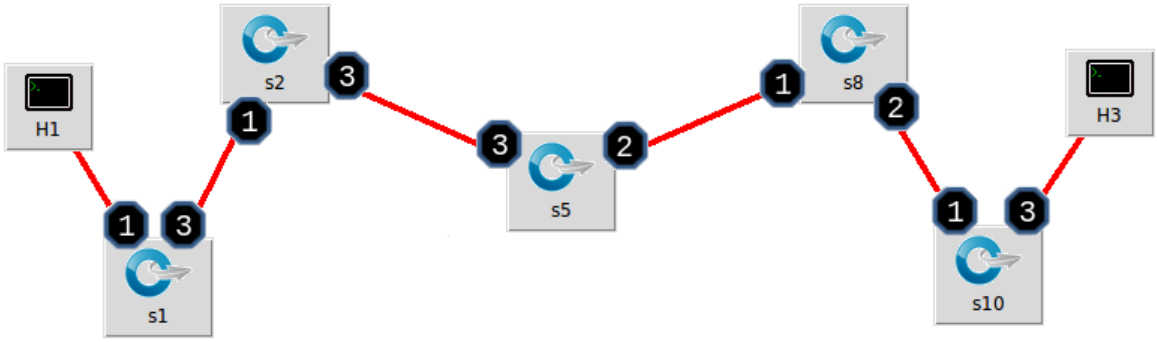
Novamente, o controlador B solicita a instalação de um novo fluxo e como encaminhar tráfegos similares a este, porém agora com protocolo ICMP.

Quadro 23- O OVX^{R} cria os fluxos nos switches da rede física

Dados	**Pacotes de controle**	**Origem**	**Destino**	**Conteúdo**
	4b	*OpenVirteX^{R}*	*Switch* 1	*OFPT_Flow-mod*: Prot: ICMP Src: 00-00-00-00-00-11, Dst: 00-00-00-00-00-13, IN = 1 → OUT = 3
	4b	*OpenVirteX^{R}*	*Switch* 2	*OFPT_Flow-mod*: Prot: ICMP Src: 00-00-00-00-00-11, Dst: 00-00-00-00-00-13, IN = 1 → OUT = 3
	4b	*OpenVirteX^{R}*	*Switch* 5	*OFPT_Flow-mod*: Prot: ICMP Src: 00-00-00-00-00-11, Dst: 00-00-00-00-00-13, IN = 3 → OUT = 2
	4b	*OpenVirteX^{R}*	*Switch* 8	*OFPT_Flow-mod*: Prot: ICMP Src: 00-00-00-00-00-11, Dst: 00-00-00-00-00-13, IN = 1 → OUT = 2
	4b	*OpenVirteX^{R}*	*Switch* 10	*OFPT_Flow-mod*: Prot: ICMP Src: 00-00-00-00-00-11, Dst: 00-00-00-00-00-13, IN = 1 → OUT = 2

Chegando a informação ao OVX^{R}, ele então instala os fluxos necessários para habilitar a comunicação na rede física entre o *Host* 1 e o *Host* 3, conforme ilustra a figura 40.

Figura 40 – Rota na infraestrutura física da rede pelo qual a comunicação entre H1 e H3 foi realizada.



Fonte: Próprio autor.

- **ICMP echo reply**

Por fim, é realizado o ICMP *echo reply*, referindo-se ao retorno da comunicação entre os *hosts*. A comunicação ocorre partindo do *Host* 3 ao *Host* 1, portanto quando o pacote chega no primeiro *switch* (S10), o comutador não sabe como encaminhar aquele fluxo e então realiza os mesmos passos da comunicação anterior.

Quadro 24- Comunicação ICMP echo reply

Dados	Pacotes de controle	Origem	Destino	Conteúdo
1		*Host* 3	*Host* 1	ICMP *echo reply*
	1a	*Switch* 10	*OpenVirteX*^{R}	OFPT_*Packet-in*: (ICMP echo reply)
	1b	*OpenVirteX*^{R}	*FlowVisor*	OFPT_*Packet-in*: (ICMP echo reply)
	1c	*FlowVisor*	Controlador B	OFPT_*Packet-in*: (ICMP echo reply)
2		Controlador B	*FlowVisor*	OFPT_*Flow-mod*: Prot: ICMP Src: 00-00-00-00-00-13, Dst: 00-00-00-00-00-11, IN = 3 → OUT = 1
	2a	*FlowVisor*	*OpenVirteX*^{R}	OFPT_*Packet-out*: Pacote 4

				encapsulado (ICMP echo reply)
	2a	*OpenVirteX*^{R}	*Switches* 10, 8 5, 2 e 1	Escreve fluxos nos switches para habilitar a comunicação entre H3 → H1.
3		*Switch* 1	*Host* 1	ICMP *echo reply*

Após isso, a comunicação entre o *Host* 1 e o *Host* 3 é estabelecida, consequentemente os pacotes seguintes, desta comunicação, serão enviados diretamente ao *host* de destino e não será mais necessário passar pelo controlador.

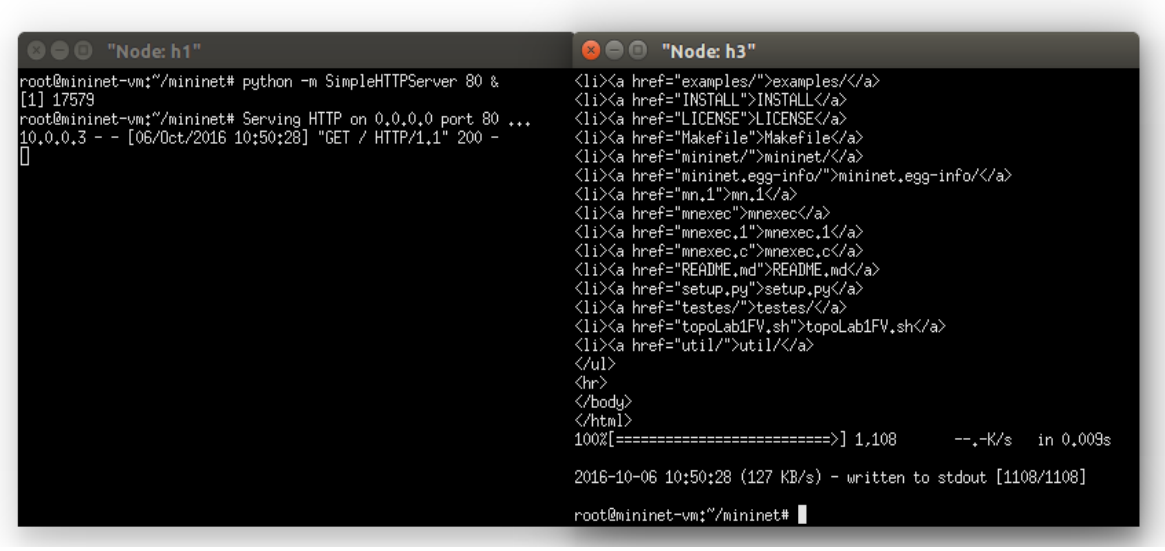
8.1.3 Configurando um servidor Web

Ping não é o único comando no *Mininet* que pode ser executado em um *host*. *Hosts* virtuais podem executar qualquer comando ou programa que está disponível para o sistema subjacente Linux, ou do sistema operacional na máquina virtual. Esta subseção cria um servidor *web* HTTP simples no *host* 1 e faz uma requisição do *host* 3 com base na topologia de experimentação apresentada na Figura 33. Os comandos a seguir apresentados na caixa de texto representam o início do servidor *Web* no H1 e a requisição de conexão HTTP do H3.

```
mininet@mininet-vm:~$ h1 python -m SimpleHTTPServer 80 &
mininet@mininet-vm:~$ h3 wget -O - h1
```

A figura 41 demonstra como isso é visto na tela. O CLI do *Mininet* permite aos usuários o controle e gerenciamento da totalidade da rede simulada a partir do *console*.

Figura 41 – Imagem da tela da comunicação HTTP com o servidor Web



Fonte: Próprio autor.

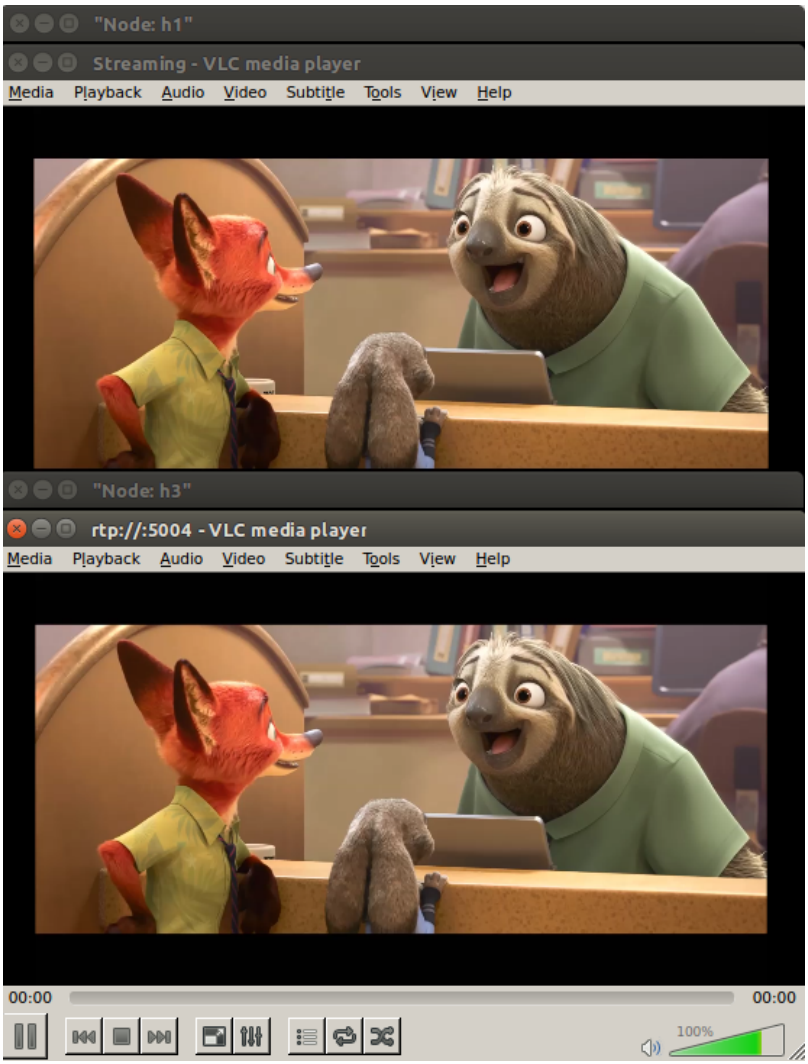
Percebe-se que a comunicação funciona normalmente, evidenciando o correto funcionamento para este tipo de comunicação.

8.1.4 Transmissão de vídeo via rede através do VLC *Media Player*

Uma sessão de *streaming* foi realizada com o objetivo de demonstrar que o *OpenVisor* pode gerenciar e suportar aplicações sensíveis a falhas. O VLC é um *player open-source* multimídia e multi-plataforma. O VLC foi utilizado para transmitir um arquivo de vídeo local do *host* 1 para o *host* 3. Configurou-se o H1 como o servidor do *stream*, responsável por transmitir o vídeo pela rede e o H3 sendo o cliente que recebe a transmissão. Este experimento foi baseado na topologia de rede apresentada na Figura 20. A transmissão ocorre através do protocolo RTP (*Real time protocol*) na porta 5004 com o encapsulamento de um vídeo MP4.

Figura 42 mostra como ele é visto na tela, tanto no servidor quanto pelo cliente. Nesta configuração, o *OpenVisor* pode adicionar os respectivos fluxos que permitem ao cliente para receber o vídeo com sucesso e sem perda qualidade impactante.

Figura 42 – Imagem da tela da seção VLC de *Streaming* de vídeo.



Fonte: Próprio autor.

Para verificar as entradas de fluxo nos comutadores, o comando *dpctl dump-flows* é executado e a caixa de texto a seguir apresenta os fluxos ativos do *switch* s1:

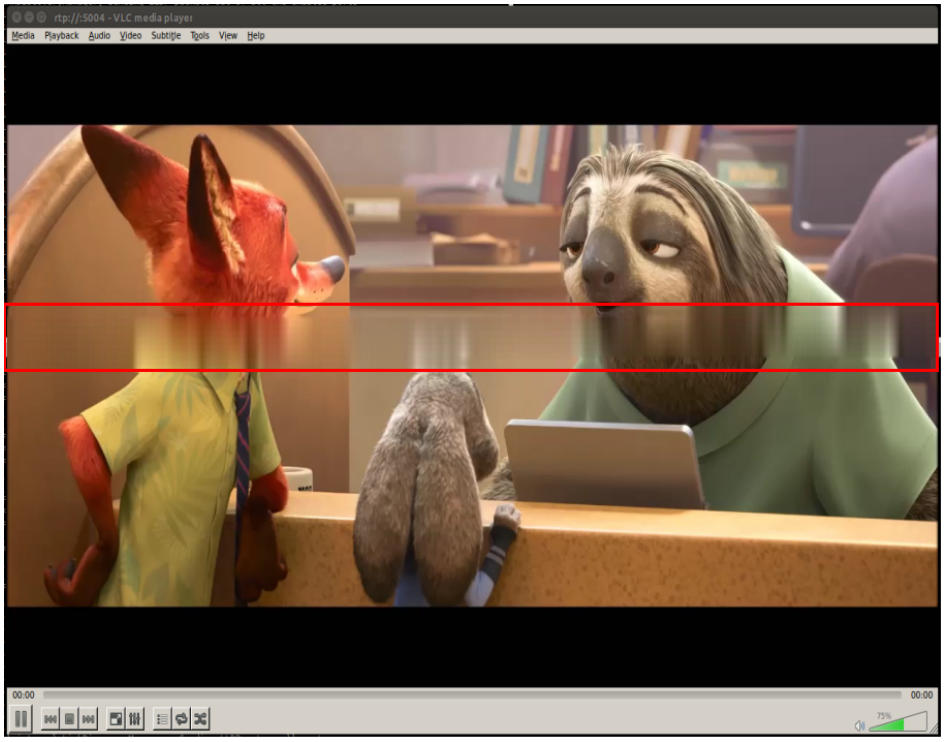
```
*** s1 ----------------------------------------------------------------------------------------------------
cookie=0x10000000b,duration=15.863s,table=0,n_packets=1041,n_bytes=1426170,
idle_timeout=5,idle_age=0,priority=1,udp,in_port=1,dl_src=00:00:00:00:00:31,
dl_dst=00:00:00:00:00:33,nw_src=10.0.0.1,nw_dst=10.0.0.3,tp_src=44142,tp_dst=5004
actions=mod_nw_dst:1.0.0.2,mod_nw_src:1.0.0.1,output:3
```

A comunicação ocorre normalmente, onde pode-se visualizar um fluxo ativo no s1 saindo do H1, cujo endereço IP é 10.0.0.1 com destino ao endereço IP 10.0.0.3 que representa

o H3. Ainda pode-se visualizar a conversão do endereço IP do *hosts* para um endereço virtual na rede 1.0.0.0/16.

Realizou-se também uma simulação de falha na infraestrutura física no momento em que o *stream* estava ocorrendo. A rota pelo qual o OVX^{R} definiu para realizar esta comunicação foi entre os comutadores s1-s2-s5-s8-s10, portanto simulou-se uma falha entre os enlaces s1-s2 no meio da transmissão do vídeo entre o H1 e H3. Identificou-se que o *streaming* não é interrompido, pois o OVX^{R} rapidamente altera para uma rota secundária através do s1-s3-s6-s9-s10. Notou-se apenas uma pequena perda de qualidade no vídeo no exato momento da convergência, conforme é visualizado na figura 43.

Figura 43 – Momento da simulação da falha na *stream* de vídeo.



Fonte: Próprio autor.

É perceptível que o *OpenVisor* torna as redes resilientes que na ocorrência de falhas se torna quase que imperceptível para o usuário, mesmo com tráfego sensível como é o caso de uma transmissão de vídeo e em tempo real. Convém destacar que o experimento foi realizado sobre uma infraestrutura de rede simulada e os resultados podem divergir em contextos reais.

8.2 TESTES DE FUNCIONALIDADES DO OPENVISOR

Os experimentos apresentados nesta seção objetivam avaliar as funcionalidades existentes após a integração das ferramentas OVX e FV. Busca-se analisar se as funcionalidades de ambas ferramentas foram unificadas e apresentam-se no mesmo cenário de testes. Foi tomado como base das testagens, os estudos de (AL-SHABIBI, 2014) e (SHERWOOD et al., 2009), que descrevem as características dos *hypervisors*. Para tanto foram realizados 4 experimentos, sendo estes: Virtualização de Topologia; Recuperação de Falhas; Flexibilidade na Definição de Redes; e Controle Absoluto das Redes Virtuais.

8.2.1 Virtualização de topologia

Para validar a característica de virtualização de topologia, vista na configuração apresentada na Figura 38, é apresentado as visões de cada componente do contexto de experimentação. Esta característica e todo processo de mapeamento físico para o virtual e vice-versa dos componentes é desempenhado pelo OVX^R.

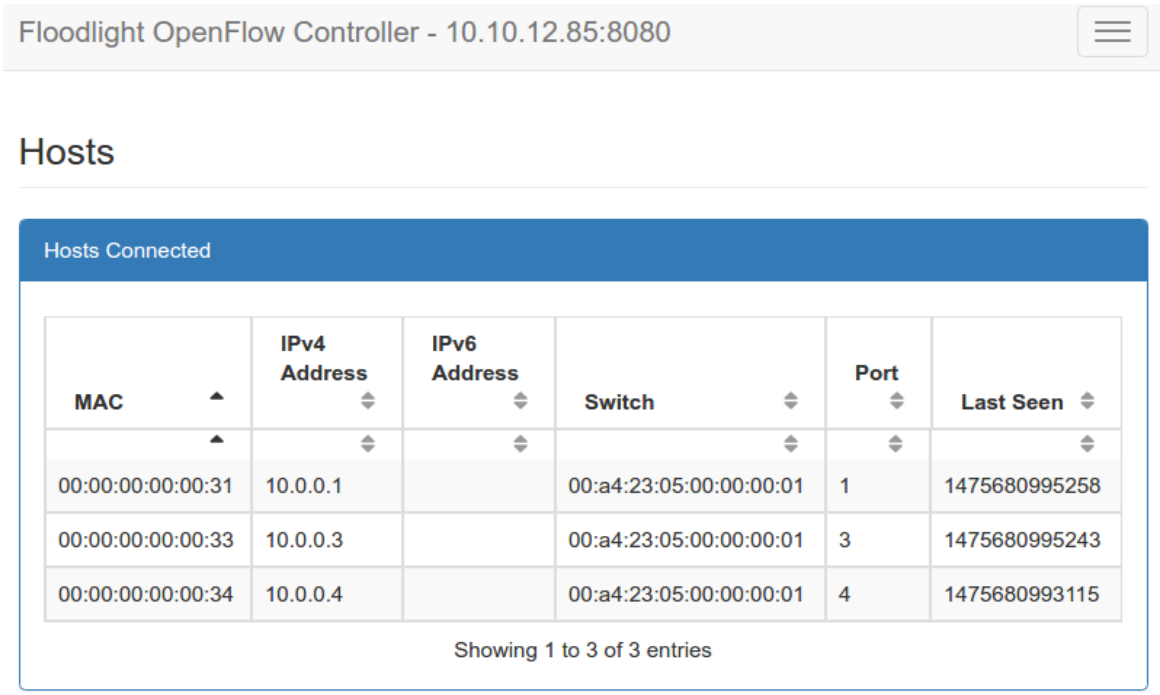
Logo após criar a rede virtual com topologia arbitrária, o OVX^R envia as informações ao FV. O FV então passa a operar exclusivamente na rede virtual, desconhecendo totalmente a infraestrutura física e suas características. Através do comando "*list-datapaths*" no FV é possível verificar os comutadores existentes conectados a ele, conforme é apresentado a seguir:

```
flowvisor@flowvisor-KVM:~$ fvctl list-datapaths
  Connected switches:
    1 : 00:a4:23:05:00:00:00:01
```

É importante notar que seja qual for a quantidade de equipamentos e componentes da rede física, a topologia virtual e arbitrária criada pelo OVX será sempre a mesma, representada por apenas um *switch* virtual. Visto que a perspectiva do FV seja da topologia virtual, as redes criadas a partir dele também terão esta mesma perspectiva, diferenciando-se apenas nos *hosts*. Enquanto que a primeira rede criada pelo FV, gerenciada pelo controlador A visualizará um único *switch* com os *hosts* H1 e H3, a outra rede cujo controlador B gerencia,

tem a mesma perspectiva porém com os *hosts* H1, H3 e H4. A Figura 44 apresenta a perspectiva do controlador B através de sua interface gráfica de gerenciamento.

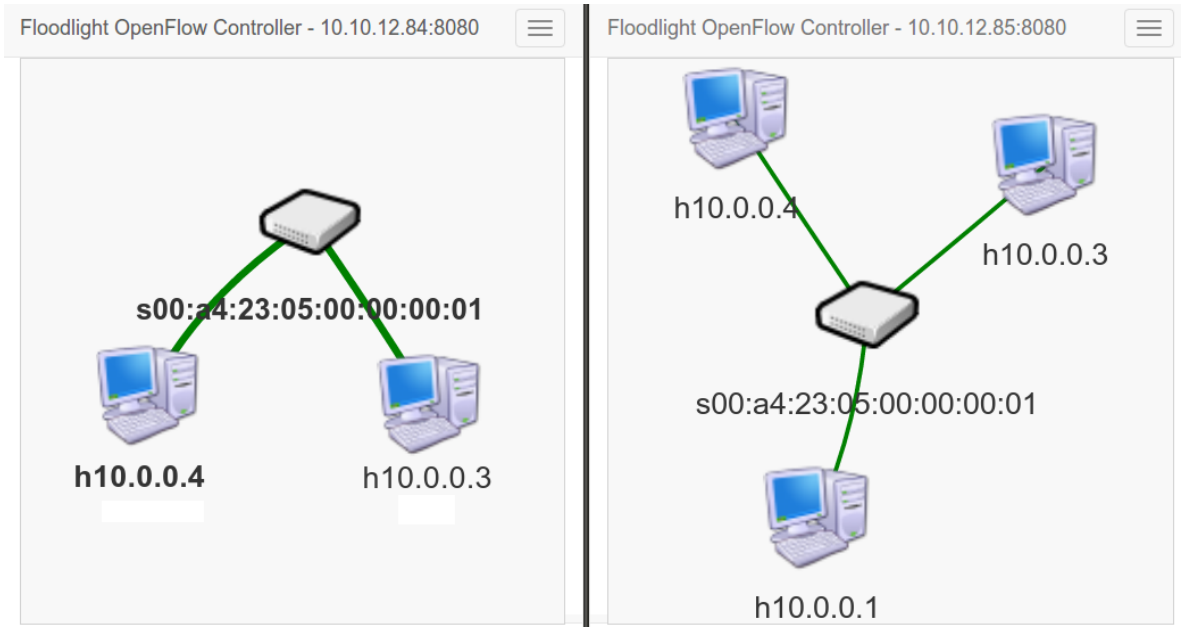
Figura 44 – Visão geral da rede na perspectiva do Controlador B.



Fonte: Próprio autor.

Percebe-se que o controlador identifica um *switch OpenFlow* e três *hosts* conectados, H1, H3 e H4 nas respectivas portas 1, 3 e 4. Este *switch*, cujo DPID é 00:a4:23:05:00:00:00:01 é o *switch* virtual criado pelo OVX e repassado ao FV. Ainda através da *interface* de gerenciamento dos controladores, é verifica-se a topologia da rede, conforme é apresentado na Figura 45, onde é possível verificar as perspectivas de ambas as redes.

Figura 45 – Topologia da rede nas perspectivas dos controladores A e B.



Fonte: Próprio autor.

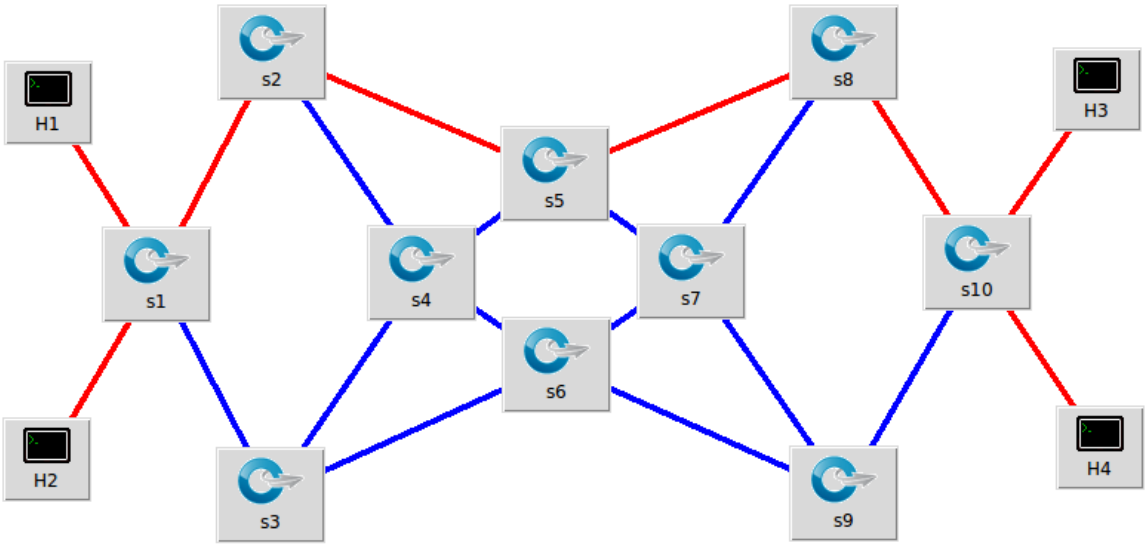
Portanto, esta topologia simplifica a operacionalização na perspectiva do usuário, onde em vez de operar sobre uma infraestrutura com diversos dispositivos e complexa, opera em uma rede com um *switch* e os *hosts* que serão usados nos experimentos. Desta forma, conclui-se que a funcionalidade de virtualização de topologia e arbitrariedade está presente após a integração proposta

8.2.2 Recuperação de falhas

Para validar a funcionalidade de recuperação de falhas, utilizou-se diferentes velocidades nos *links* da rede física, de modo a se ter uma melhor rota composta pela maior taxa de transmissão e demais rotas alternativas para a comunicação entre os *hosts*. Modificou-se no *Mininet* a taxa dos *enlaces* para 1000Mbits e 100Mbits da topologia de experimentação criada. Alterou-se as cores dos links para a melhor compreensão da modificação.

Definiu-se que os enlaces em vermelho teriam a taxa de 1000Mbits enquanto os demais *enlaces* representados na cor azul 100Mbits, conforme é apresentado na Figura 46. Desta forma, objetivando verificar o cálculo de rotas do OVX^{R}, que neste caso deveria escolher como rota primária o caminho com maior banda, grifado em vermelho.

Figura 46 – Rota principal escolhida pelo OVX^{R}.



Fonte: Próprio autor.

Para este experimento utilizou-se a ferramenta *Iperf* (IPERF, 2014), inclusa no *Mininet* para gerar tráfego entre os *hosts* H1 e H3, sob gerência do controlador A, em intervalos de 0.5 segundos. O controlador por ter a visão somente da topologia virtual, desconhece o caminho real que o tráfego percorre. Neste caso, o OVX^{R} é o responsável por definir as comunicações e os caminhos dos fluxos na infraestrutura física.

Inicialmente, o OVX^{R}, define a rota que o fluxo percorrerá na rede física. No entanto, logo que o OVX^{R} realiza o cálculo de rotas entre os *hosts* da rede, ele ainda armazena uma rota secundária internamente para que na ocorrência de falhas na rota principal, não seja

necessário realizar um novo cálculo, conforme é mostrado em seu *debug*, ilustrado na caixa de texto a seguir:

12:07:18.368 [qtp337165520-77] INFO OVXBigSwitch - **Add route for big-switch 00:a4:23:05:00:00:00:01 between ports (1,3) with priority: 64** and path: [00:00:00:00:00:00:00:01/3-00:00:00:00:00:00:00:02/1, 00:00:00:00:00:00:00:02/3-00:00:00:00:00:00:00:05/3, 00:00:00:00:00:00:00:05/2-00:00:00:00:00:00:00:08/1, 00:00:00:00:00:00:00:08/2-00:00:00:00:00:00:00:0a/1]

12:07:18.379 [qtp337165520-77] INFO OVXBigSwitch - **Add route for big-switch 00:a4:23:05:00:00:00:01 between ports (3,1) with priority: 64** and path: [00:00:00:00:00:00:00:0a/1-00:00:00:00:00:00:00:08/2, 00:00:00:00:00:00:00:08/1-00:00:00:00:00:00:00:05/2, 00:00:00:00:00:00:00:05/3-00:00:00:00:00:00:00:02/3, 00:00:00:00:00:00:00:02/1-00:00:00:00:00:00:00:01/3]

12:07:18.381 [qtp337165520-77] INFO OVXBigSwitch - **Add backup route for big-switch 00:a4:23:05:00:00:00:01 between ports (1,3) with priority: 63** and path: [00:00:00:00:00:00:00:01/4-00:00:00:00:00:00:00:03/1, 00:00:00:00:00:00:00:03/2-00:00:00:00:00:00:00:06/1, 00:00:00:00:00:00:00:06/4-00:00:00:00:00:00:00:09/3, 00:00:00:00:00:00:00:09/2-00:00:00:00:00:00:00:0a/4]

12:07:18.381 [qtp337165520-77] INFO OVXBigSwitch - **Add backup route for big-switch 00:a4:23:05:00:00:00:01 between ports (3,1) with priority: 63** and path: [00:00:00:00:00:00:00:0a/4-00:00:00:00:00:00:00:09/2, 00:00:00:00:00:00:00:09/3-00:00:00:00:00:00:00:06/4, 00:00:00:00:00:00:00:06/1-00:00:00:00:00:00:00:03/2, 00:00:00:00:00:00:00:03/1-00:00:00:00:00:00:00:01/4]

Neste caso, o OVX^{R} cria um mapeamento entre as portas virtuais existentes que referem-se aos *hosts* da rede. Como pode-se visualizar, foram criadas duas rotas de ida e volta entre as portas virtuais 1 e 3, as quais estão conectados os *hosts* H1 e H3. A rota principal foi estabelecida no caminho s1-s2-s5-s8-s10 cuja prioridade foi definida como 64. Já a rota secundária, com prioridade menor que a principal passa pelos dispositivos s1-s3-s6-s9-s10. Na ocorrência de falhas nos *enlaces* da rota principal, o OVX^{R} altera os fluxos para a rota secundária sem a necessidade de um recálculo para restabelecer a comunicação.

Iniciada a comunicação entre H1 e H3, o OVX^{R} define a comunicação utilizando o caminho definido como rota principal (s1-s2-s5-s8-s10), conforme é evidenciado pelo comando *dpctl dump-flows* no *Mininet* na caixa de texto a seguir, para verificar fluxos ativos nos switches da infraestrutura, podendo verificar de fato por onde o tráfego está passando.

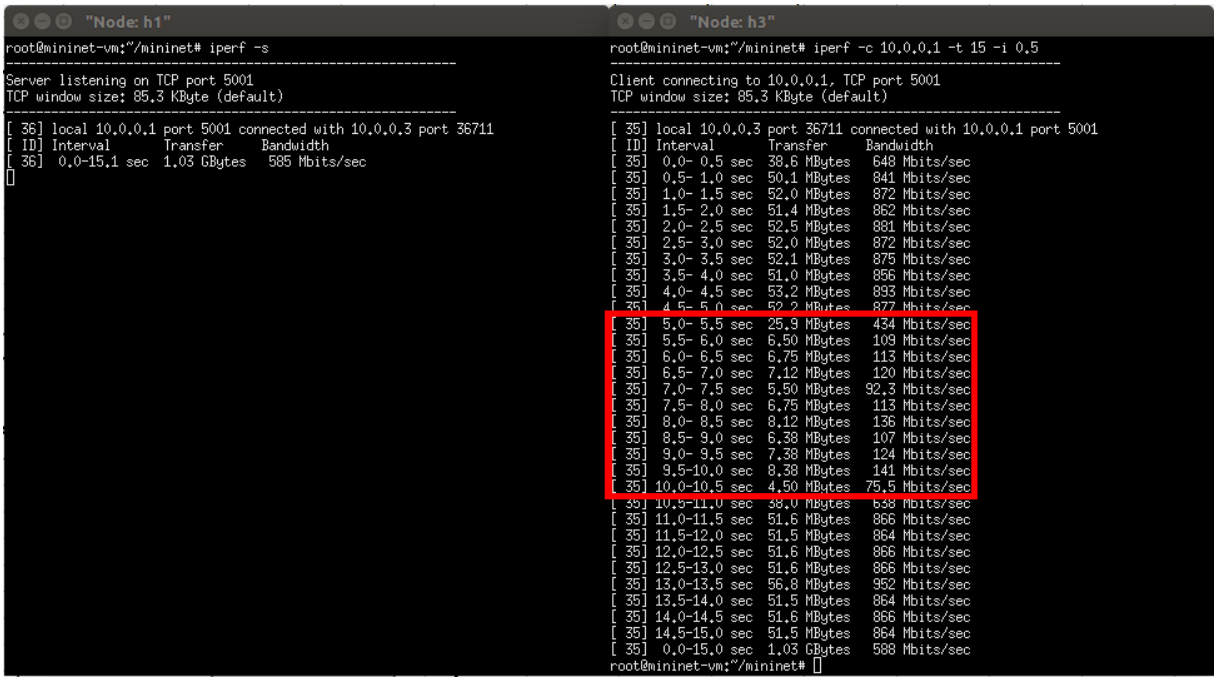
```
mininet> dpctl dump-flows
*** s1 ----------
cookie=0x100000004,duration=6.392s,table=0,n_packets=6,n_bytes=532,idle_timeout=5,idle_age=1,
priority=0,in_port=1,vlan_tci=0x0000,dl_src=00:00:00:00:00:31,dl_dst=00:00:00:00:00:33
actions=mod_nw_dst:1.0.0.2,mod_nw_src:1.0.0.1,output:3
*** s2 ----------
cookie=0x100000004,duration=6.404s,table=0,n_packets=6,n_bytes=532,idle_timeout=5,idle_age=1,
priority=0,in_port=1,vlan_tci=0x0000,dl_src=00:00:00:00:00:31,dl_dst=00:00:00:00:00:33
actions=output:3
*** s3 ----------
*** s4 ----------
*** s5 ----------
cookie=0x100000004,duration=6.419s,table=0,n_packets=6,n_bytes=532,idle_timeout=5,idle_age=1,
priority=0,in_port=3,vlan_tci=0x0000,dl_src=00:00:00:00:00:31,dl_dst=00:00:00:00:00:33
actions=output:2
*** s6 ----------
*** s7 ----------
*** s8 ----------
cookie=0x100000004,duration=6.434s,table=0,n_packets=6,n_bytes=532,idle_timeout=5,idle_age=1,
priority=0,in_port=1,vlan_tci=0x0000,dl_src=00:00:00:00:00:31,dl_dst=00:00:00:00:00:33
actions=output:2
*** s9 ----------
*** s10 ----------
cookie=0x100000004,duration=6.453s,table=0,n_packets=6,n_bytes=532,idle_timeout=5,idle_age=1,
priority=0,in_port=1,vlan_tci=0x0000,dl_src=00:00:00:00:00:31,dl_dst=00:00:00:00:00:33
actions=mod_nw_src:10.0.0.1,mod_nw_dst:10.0.0.3,output:3
```

Nota-se que apenas os switches da rota principal possuem fluxos ativos, confirmando que a rota principal foi definida na comunicação entre H1 e H3. Depois de 5 segundos, simulou-se uma falha no *enlace* entre os switches s1-s2, de modo a forçar o sistema de *failover* do OVX^{R}, que logo após receber a informação da infraestrutura que houve uma falha de conectividade, estabeleceu a rota secundária entre H1 e H3 definida anteriormente (s1-s3-s6-s9-s10), conforme pode-se ver novamente com o *dpclt dump-flows*:

```
mininet> dpctl dump-flows
*** s1 ----------
cookie=0x100000014,duration=6.49s, table=0,n_packets=6, n_bytes=532,idle_timeout=5,idle_age=1,
priority=0,in_port=1,vlan_tci=0x0000,dl_src=00:00:00:00:00:31,dl_dst=00:00:00:00:00:33
actions=mod_nw_dst:1.0.0.2,mod_nw_src:1.0.0.1,output:4
*** s2 ----------
*** s3 ----------
cookie=0x100000014,duration=6.507s,table=0,n_packets=6,n_bytes=532,idle_timeout=5,idle_age=1,
priority=0,in_port=1,vlan_tci=0x0000,dl_src=00:00:00:00:00:31,dl_dst=00:00:00:00:00:33
actions=output:2
*** s4 ----------
*** s5 ----------
*** s6 ----------
cookie=0x100000014, duration=6.523s,table=0,n_packets=6,n_bytes=532,idle_timeout=5, idle_age=1,
priority=0,in_port=1,vlan_tci=0x0000,dl_src=00:00:00:00:00:31,dl_dst=00:00:00:00:00:33
actions=output:4
*** s7 ----------
*** s8 ----------
*** s9 ----------
cookie=0x100000014,duration=6.539s,table=0,n_packets=6,n_bytes=532,idle_timeout=5, idle_age=1,
priority=0,in_port=3,vlan_tci=0x0000,dl_src=00:00:00:00:00:31,dl_dst=00:00:00:00:00:33
actions=output:2
*** s10 ----------
cookie=0x100000014,duration=6.545s,table=0,n_packets=6,n_bytes=532,idle_timeout=5,idle_age=1,
priority=0,in_port=4,vlan_tci=0x0000,dl_src=00:00:00:00:00:31,dl_dst=00:00:00:00:00:33
actions=mod_nw_src:10.0.0.1,mod_nw_dst:10.0.0.3,output:3
```

Em virtude das duas rotas possuírem larguras de banda diferentes, é possível visualizar o *failover* do OVX^{R} através da taxa de vazão do experimento, conforme ilustra a Figura 47. Inicialmente a comunicação ocorre pelo caminho principal, cuja taxa de transimssão é de 1000Mbps e no instante 5 é simulada a falha, onde é possível identificar que a vazão cai bruscamente até aproximadamente 100Mbps, cuja é a banda da rota secundária escolhida pelo OVX^{R}. No instante 10 foi restabelecido o enlace s1-s2 e a rota principal voltou a funcionar. Logo o *OpenVisor* altera novamente o caminho dos fluxos, conforme é possível visualizar a vazão aumentar consideravelmente.

Figura 47 – Rota principal na infraestrutura de testes.



Fonte: Próprio autor.

Logo, evidenciou-se o funcionamento da característica de recuperação de falhas na operacionalização do *OpenVisor*, concluindo que o *framework* de fato fornece redes virtuais resilientes aos usuários.

8.2.3 Flexibilidade na definição de redes

Através da característica de flexibilidade proveniente do FV é possível anular a fragilidade do OVX de não permitir que um único *host* pertença a mais de uma rede virtual. Conforme é visto na caixa de texto a seguir, o OVX informa a seguinte mensagem de erro na tentativa de associar um mesmo *host* em mais de uma rede virtual.

```
root@ovx-vm:/home/ovx/OpenVirteX/utils#
python ovxctl.py -n connectHost 2 00:a4:23:05:00:00:00:01 1 00:00:00:00:00:31

{u'jsonrpc': u'2.0', u'id': u'ovxctl', u'error': {u'message': u'ConnectHost:
The specified MAC address is already in use: 00:00:00:00:00:31', u'code': -32602}}
```

Conforme apresentado, o erro mostra que não é possível associar o *host* com o *mac-address* 00:00:00:00:00:31, correspondente ao H1 pois ele já está sendo utilizado em outra rede virtual. Para suprir esta fragilidade o *OpenVisor* designa a tarefa de segmentação e definição das redes virtuais ao FV, desta forma permitindo a associação de um *host* em diversos *slices* e possibilitando criar redes virtuais flexíveis.

Definiu-se duas redes virtuais com os mesmos *hosts*, porém com políticas de *flowspace* distintas, uma abrangendo tráfego TCP porta 9999 proveniente dos *hosts* H1 e H3 e a outra contemplando qualquer tráfego distinto a esse e de origem os *hosts* H1, H3 e H4.

Isso é possível pois quando um *host* pertence a mais de uma rede, o FV a partir das políticas de cada *slice*, determina para qual controlador será encaminhado os fluxos, podendo ora as comunicações daquele *host* ser processadas por um controlador, ora por outro. Desta forma os *slices* trabalham de forma isolada dos demais, mesmo utilizando *hosts* em comum, pois é de responsabilidade do FV encaminhar os fluxos para o controlador correto, de acordo com o *flowspace* de cada rede virtual.

Pode-se verificar os *flowspaces* configurados no FV, conforme é ilustrado na caixa de texto a seguir através do comando *list-flowspace*. O primeiro *flowspace* pertence ao *slice* “redeB”, cujo refere-se para qualquer tráfego entre H1 e H3, mais especificamente, todo tráfego de entrada/saída na porta 1 (*in_port*:1) do *switch* com DPID 00:a4:23:05:00:00:00:01 pertencerá a rede denominada como redeB e consequentemente deverá ser encaminhada para o controlador em questão.

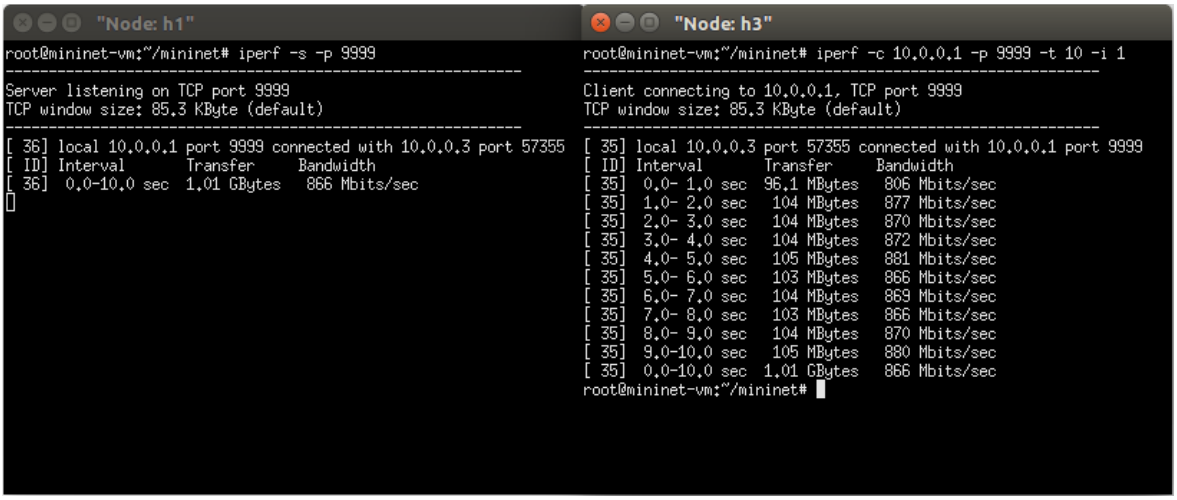
```
flowvisor@flowvisor-KVM:~$ fvctl list-flowspace
Configured Flow entries:
{"name": "s1-p1", "slice-action": [{"slice-name": "redeB"}], "priority": 1,
"dpid": "00:a4:23:05:00:00:00:01",
"match": {"in_port": 1}}

{"name": "s1-p1-src", "slice-action": [{"slice-name": "redeA"}], "priority": 100,
"dpid":"00:a4:23:05:00:00:00:01",
"match": {"dl_type": 2048, "nw_proto": 6, "in_port": 1" tp_src": 9999}}
```

Porém, o outro *flowspace* configurado refere-se a rede virtual denominada “redeA”, cuja política de *flowspace* é exclusivamente para fluxos IPv4 (*dl_type*:2048) com protocolo de transporte TCP (*nw_proto*:6) na porta 9999 (*tp_src*), provenientes dos *hosts* H1, H2 e H4.

Portanto, o FV recebendo fluxos com estas métricas, encaminha para o controlador B, que gerencia esta rede virtual.

Ressalta-se que para todas as regras de *flowspaces* deve-se atribuir uma prioridade. A partir da prioridade é definido o *flowspace* escolhido pelo FV, no caso de um fluxo satisfazer as métricas de mais de um *slice*. Neste caso, o FV atribui o fluxo para o *flowspace* com maior prioridade. Visto que a política de fluxos da “redeA” poderia ser satisfeita nas regras de *flowspace* da “redeB”, verificou-se para qual controlador o FV enviava os fluxos, a fim de validar se estava desempenhando a função corretamente. Portanto, fluxos com métricas TCP porta 9999 eram satisfeitas por ambas as métricas das redes, porém o FV deveria encaminhar ao Controlador A o fluxo em virtude da prioridade maior. A seguir é apresentado a comunicação entre H1 e H3 através do *Iperf* gerando tráfego TCP na porta 9999.

Figura 48 – Comunicação entre os hosts H1 e H3 através da “rede A”.

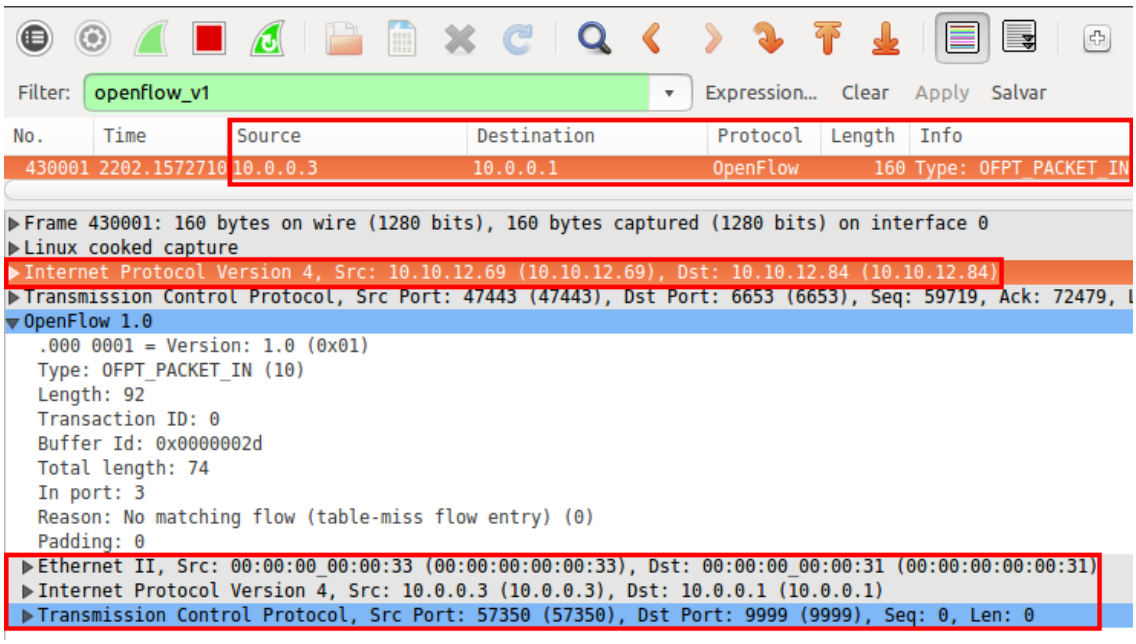


Fonte: Próprio autor.

Percebe-se que a comunicação utilizando a porta 9999 entre os *hosts* H1 e H3 ocorre normalmente, porém para identificar para qual controlador *OpenFlow* o FV encaminha o fluxo foi necessário analisar a comunicação utilizando a ferramenta *Wireshark*, conforme a figura 48 ilustra. Nota-se que a comunicação TCP, cuja possui o encapsulamento do tráfego *OpenFlow*, possui como IP de origem o 10.10.12.69, referente ao FV e de destino o

10.10.12.84, o endereço IP do controlador A. Dentro do fluxo *OpenFlow*, percebe-se que a comunicação é proveniente do H3 (10.0.0.3) com destino ao H1 (10.0.0.1).

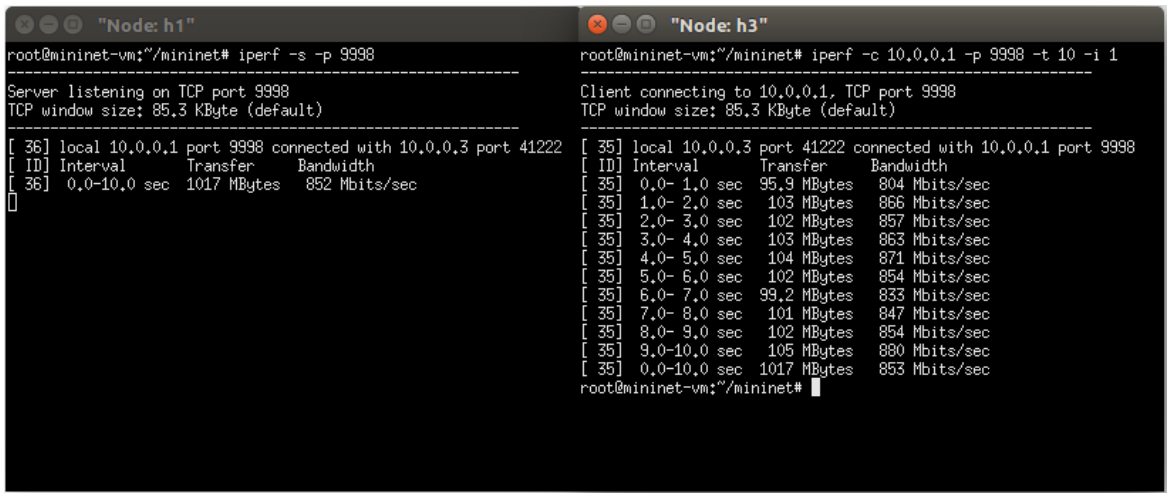
Figura 49 – *Wireshark* comunicação H1 e H3 através da “rede A”.



Fonte: Próprio autor.

Nota-se que o FV age corretamente enviando os fluxos ao *slice* adequado, com base nas métricas definidas anteriormente. Realizou-se uma nova comunicação entre H1 e H3, porém utilizando como métricas um fluxo TCP porta 9998, conforme a Figura 49, com o intuito de visualizar o FV encaminhando esta comunicação para o controlador da “redeB”, visto que seu *flowspace* controla quaisquer fluxos, desde que diferentes da “redeA”.

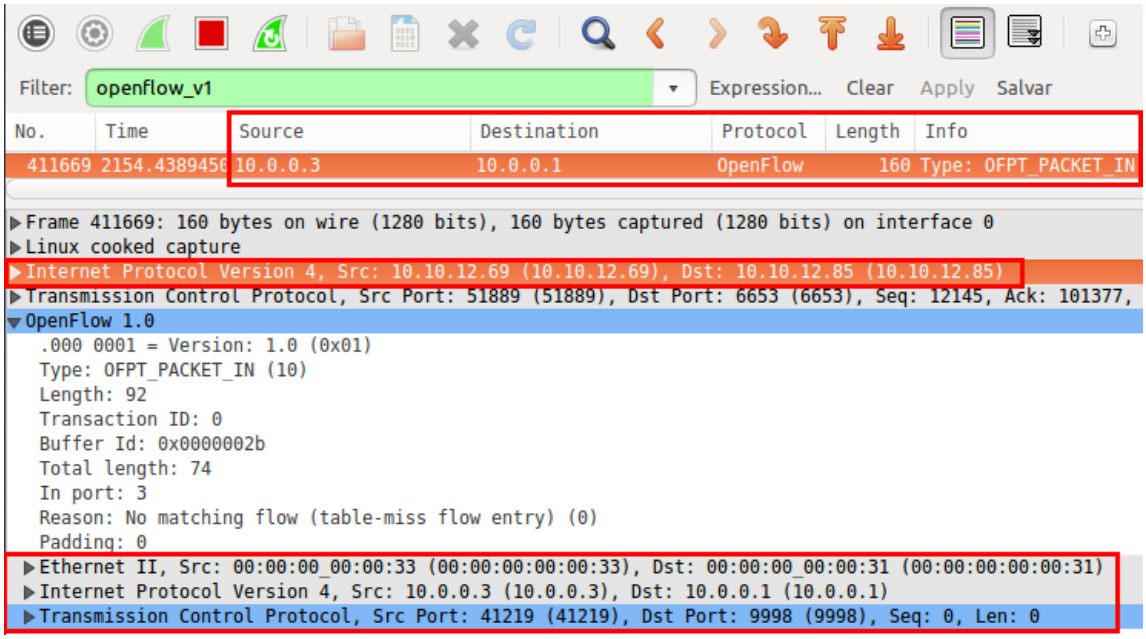
Figura 50 – Comunicação entre os hosts H1 e H3 através da “rede B”.



Fonte: Próprio autor.

Da mesma forma que anteriormente, nota-se que a comunicação ocorre normalmente e somente através da análise com o *Wireshark* é possível identificar para qual controlador o FV encaminha. Percebe-se que o FV (10.10.12.69) encaminha ao endereço IP 10.10.12.85, referente ao controlador B, a comunicação *OpenFlow* entre H3 (10.0.0.3) com destino ao H1 (10.0.0.1), conforme é apresentado na Figura 51.

Figura 51 – *Wireshark* Comunicação entre os hosts H1 e H3 através da “rede B”.



Fonte: Próprio autor.

Logo, percebe-se que a flexibilização na segmentação das redes acontece normalmente após a integração, onde de acordo com os tipos de fluxos da rede, as comunicações são encaminhadas ora para um controlador, ora para outro.

8.2.4 Controle absoluto da rede virtual

O controle absoluto da rede virtual é característica proveniente do OVX, que a partir da forma pelo qual segmenta as redes possibilita que cada rede virtual possa controlar a totalidade do *flowspace*. Porém os testes demonstraram resultados diferentes do qual era esperado previamente.

Para analisar este processo, é preciso visualizar como o OVX fornece o controle absoluto para as redes virtuais. Para isso foi analisado uma comunicação entre H1 e H3 com o *Mininet*. Utilizando o comando *dpctl dump-flows* no *Mininet* pode-se visualizar os fluxos passando pelos switches, onde é possível identificar que o fluxo logo que sai do H1 e chega

no *switch* s1, realiza um procedimento de conversão do endereço IP real para um IP virtual, fazendo com que seja mascarado o endereço IP verdadeiro do *host* H1, conforme é possível visualizar na caixa de texto a seguir:

```
mininet> dpctl dump-flows
*** s1 ----------------------------------------------------------------------------------------------------------------
cookie=0x100000014,duration=6.49s, table=0,n_packets=6, n_bytes=532,idle_timeout=5,idle_age=1,
priority=0,in_port=1,vlan_tci=0x0000,dl_src=00:00:00:00:00:31,dl_dst=00:00:00:00:00:33
actions=mod_nw_dst:1.0.0.2,mod_nw_src:1.0.0.1,output:4
        …
        …
        …
*** s10 ---------------------------------------------------------------------------------------------------------------
cookie=0x100000014,duration=6.545s,table=0,n_packets=6,n_bytes=532,idle_timeout=5,idle_age=1,
priority=0,in_port=4,vlan_tci=0x0000,dl_src=00:00:00:00:00:31,dl_dst=00:00:00:00:00:33
actions=mod_nw_src:10.0.0.1,mod_nw_dst:10.0.0.3,output:3
```

Quando a informação chega no s10, *switch* onde o *host* de destino está conectado, é realizado novamente a modificação do endereço IP, porém do endereço IP virtual para o real e então são restabelecidos os endereços IPs reais dos *hosts* H1 e H3. Desta forma, quaisquer fluxos provenientes desta rede virtual serão mascarados para endereços virtuais da rede 1.0.0.0/16 e trafegarão pela rede com endereços únicos naquela infraestrutura. No caso de existir outras redes virtuais criadas pelo OVX, ocorrerão o mesmo processo, serão mascarados os IPs reais dos *hosts* por endereços virtuais e únicos para que sejam isolados e não conflitem entre as redes existentes.

Entende-se que este procedimento é realizado para possibilitar o controle absoluto para as redes virtuais, pois para cada rede virtual criada no OVX é designado uma faixa de IP única na totalidade de redes. Desta forma, cada rede virtual pode controlar a totalidade de fluxos entre os *hosts* pertencentes a rede, não sendo necessário compartilhar o *flowspace*, conforme ocorre no FV.

Visto que a proposta deste estudo leva em conta o FV controlador a rede criada pelo OVX, utilizou-se apenas uma rede virtual criada pelo OVX, contendo um *switch* virtual e quatro *hosts*, conforme visto na Figura 39 (b). O FV detêm o controle absoluto da rede, podendo operar e controlar quaisquer fluxos entre os *hosts* H1, H2, H3 e H4. No entanto, visto que o papel do FV é adicionar flexibilidade na segmentação das redes virtuais, identificou-se que as redes fornecidas aos usuários cujo eram criadas pelo FV, não disponibilizariam o controle total do *flowspace*, pois o FV segmenta o *flowspace* total da rede

através da definição de métricas do cabeçalho *OpenFlow* para cada uma delas, permitindo o controle aos usuários apenas da política de fluxos de sua rede.

Por outro lado, em virtude da característica de controle absoluto, o OVX não permite a flexibilidade na segmentação das redes e a associação de um mesmo *host* em várias redes virtuais. Desta forma, observou-se que disponibilizar o controle total da rede para o usuário tem o impacto de fragilizar a flexibilidade do ambiente de experimentação.

8.3 COMPARAÇÕES

Nesta seção apresenta-se os testes realizados objetivando comparar, a partir dos critérios de: quantidade de fluxos, desempenho, tempo de convergência, o *framework* proposto e as ferramentas FV, OVX e *VeRTIGO*. Para os testes referentes a quantidade de fluxos foi comparado o número de fluxos exigido do usuário para operacionalizar seus testes. Testou-se em dois cenários diferentes, em uma infraestrutura SDN com funcionalidade de virtualização da topologia e outra sem esta característica. Para tanto, neste teste comparou-se o *OpenVisor* com o *FlowVisor*, visto que a falta da característica de virtualização de topologia compete a operacionalização sobre o contexto de rede real, que por sua vez pode se tornar uma tarefa de alta complexidade.

Na métrica de desempenho, a comparação desenvolvida foi tanto com o OVX quanto com o FV, onde realizou-se diferentes tipos de testes, almejando identificar diferenças significativas através do cenário de experimentação utilizado. Levou-se em conta os seguintes aspectos: taxa de vazão, tempo médio de resposta do primeiro pacote e tempo médio de resposta dos demais pacotes. Tomou-se como base a norma RFC 2544[9] e os trabalhos de (SCHWARZ, 2014), (PUPATWIBUL, 2016).

Por fim, verificou-se o tempo de convergência do *OpenVisor* comparado ao *VeRTIGO*, visto que foi a única ferramenta com a funcionalidade de recuperação de falhas, além do OVX, de modo a verificar qual método de *failover* entre elas possui uma convergência mais rápida e menos significativa para o usuário. Tomou-se como base os testes desenvolvidos por (CORIN et al., 2012).

9 RFC 2544. Disponível em https://www.ietf.org/rfc/rfc2544.txt. Acesso em 16/09/2016.

8.3.1 Quantidade de Fluxos

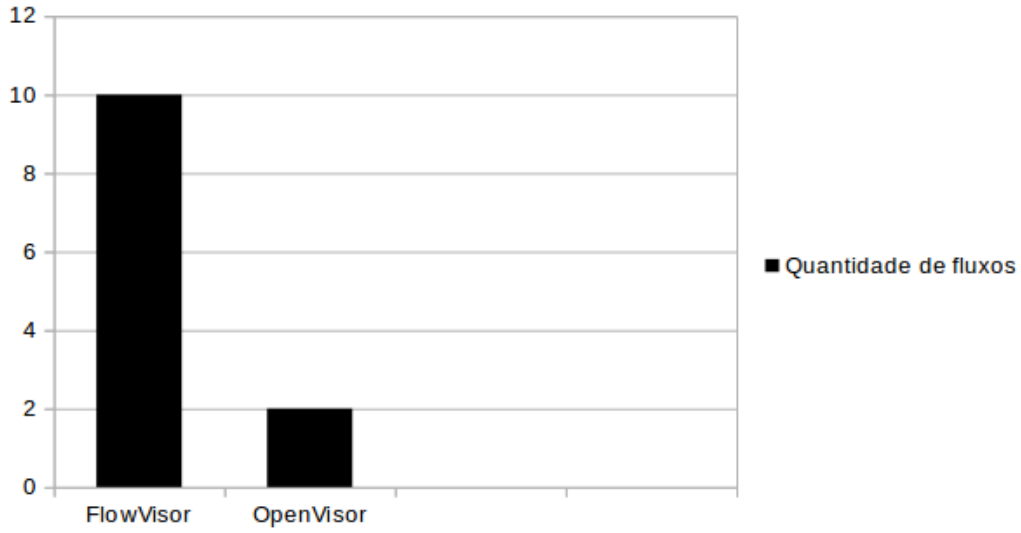
Nestes testes, definiu-se a mensuração de acordo com a quantidade de fluxos necessários para o usuário na execução de seus experimentos no *testbed*. Testou-se uma comunicação entre os *hosts* H1 e H3 sobre o contexto de experimentação apresentado na Figura 14. Comparou-se a quantia de fluxos sob diferentes testes, um através de uma rede SDN com suporte a virtualização da topologia, como é o caso do *OpenVisor* e outra sem esta característica, conforme é visto no funcionamento do *FlowVisor*.

Em um contexto de rede SDN, a quantidade de fluxos necessários está ligado diretamente ao tamanho da topologia do ambiente, mais especificamente na quantidade de saltos necessários entre o remetente e o destinatário. Seguindo essa lógica, através da equação (1) é possível mensurar esta quantia de fluxos para possibilitar uma comunicação entre dois dispositivos. Utiliza-se o ‘F’ para representar a quantidade de fluxos e o ‘N’ o número de saltos entre o remetente e o destinatário. Por fim multiplica-se por 2 representando os caminhos de ida e volta da comunicação.

$$F = 2*N \quad (1)$$

Para demonstrar este processo, utilizou-se a mesma topologia apresentada na Figura 45, sendo uma comunicação entre H1 e H3 passando pelos comutadores S1, S2, S5, S8 e S10, sobre a gerência do FV e do *OpenVisor*. No caso do FV, o usuário necessita criar 10 regras de fluxos para habilitar a comunicação entre os *hosts*, visto que N=5. Já no uso do *OpenVisor*, este número cai consideravelmente, pois é realizado a virtualização da topologia e o usuário passa a ter a perspectiva de apenas um *switch*. Portanto, o valor de N é 1 e o número de fluxos será 2, conforme mostra a Figura 52.

Figura 52 – Comparação de fluxos necessários entre o FV e o OpenVisor.



Fonte: Próprio autor.

Através deste experimento, foi possível identificar que o FV requer 5x mais regras de fluxos se comparado com o *OpenVisor*. Além disso, diferente do FV em que o número de fluxos depende do tamanho da rede e do número de saltos entre os *hosts* de origem e destino, no caso do *OpenVisor* a quantidade de fluxos será sempre fixa, independentemente do tamanho da rede, em virtude do uso de topologias virtuais e arbitrárias, ou seja, o *OpenVisor* sempre transmitirá ao usuário a perspectiva de uma rede com um único *switch* e os *hosts* conectados a ele, consequentemente para comunicação entre dois hosts será necessário 2 regras de fluxos.

Conclui-se que o *OpenVisor* necessita de um número consideravelmente menor de fluxos em relação do FV, resultando em um menor esforço na perspectiva dos usuários em programar e configurar seus experimentos. Conclui-se que o *OpenVisor* dominui a complexidade de operação em comparação com o FV.

8.3.2 Desempenho

No quesito desempenho, analisou-se o *framework OpenVisor* e comparou-se com o *OpenVirteX* e o *FlowVisor* a fim de identificar diferenças consideráveis resultantes da

implementação realizada. Foram realizados testes com base nas seguintes métricas: vazão, latência sobre o primeiro pacote e o tempo médio de uma comunicação fim a fim. Todos testes foram efetuados sobre a ferramenta *Mininet* sobre o contexto de experimentação da Figura 38, utilizando os *hosts* H1 e H3. Para todas as métricas analisadas, os valores apresentados foram obtidos a partir da execução de 20 testes com a ferramenta *Iperf*, conforme recomendação da RFC 2544, desprezando o melhor e o pior valor e calculado a média aritmética dos restantes.

1. Vazão: Buscou-se avaliar se a vazão seria prejudicada com a integração proposta. Para tanto, foi utilizada a ferramenta *Iperf* para gerar tráfego entre os nós H1 e H3. O Quadro 25 apresenta o resultado das medições realizadas.

Quadro 25- Comparativo de teste de vazão

Hypervisor	**Protocolo**	**Tamanho da transferência (média)**	**Vazão (média)**
OpenVisor	TCP	1,01 GBytes	866 Mbps
	UDP	0,98 GBytes	845 Mbps
OpenVirteX	TCP	0,99 GBytes	857 Mbps
	UDP	0,98 GBytes	849 Mbps
FlowVisor	TCP	1,02 GBytes	877 Mbps
	UDP	0,99 GBytes	854 Mbps

Os resultados identificaram uma variação muito alta nos valores encontrados em cada um das 20 amostras realizadas em cada teste. Portanto, conclui-se que os valores de vazão encontrados no *OpenVisor* em relação ao *OpenVirteX* e o *FlowVisor* não detectaram diferenças significativas em virtude da variação dos resultados. Conclui-se que neste experimento, o resultado não demonstrou impacto relevante sobre a vazão da comunicação do *OpenVisor* em comparação com as demais ferramentas, visto a variação de resultados encontrado em cada teste.

2. Tempo médio de processamento do primeiro pacote: Este teste objetiva avaliar o tempo médio de resposta do primeiro pacote de uma comunicação acontecendo sobre o framework

OpenVisor em comparação com o OVX e o FV. Visto que tanto o FV quanto o OVX operam nativamente de forma reativa, ou seja, qualquer fluxo novo que chegue no switch é encaminhado ao controlador seu primeiro pacote para que seja definido uma ação para aquele fluxo. Desta forma, o primeiro pacote sempre possui um tempo de reposta consideravelmente maior dos próximos, chegando ao switch é encaminhado para o hypervisor em busca de instruções de operação e depois encaminhado ao destino. Os pacotes subsequentes do mesmo são enviados diretamente ao destino, sem o intermédio do hypervisor.

Portanto, este teste buscou avaliar o impacto no tempo de processamento do primeiro pacote em comparação com o OVX e o FV. Para este experimento realizou-se uma comunicação *ping* entre H1 e H3. O resultado das medições é encontrado no Quadro 26.

Quadro 26- Tempo de rede do primeiro pacote

Hypervisor	**Tempo de resposta (médio)**
FlowVisor	34,6 ms
OpenVirteX	31,2 ms
OpenVisor	38,4 ms

Os dados obtidos nestes testes demonstram que não foi percebida uma diferença significativa em relação ao tempo de resposta dos primeiros pacotes. Vale lembrar que o resultado obtido refere-se apenas à troca do primeiro pacote entre os *hosts*. Nas demais trocas, não houve diferença de desempenho entre as ferramentas, visto que as tabelas de fluxo já estavam preenchidas.

3. Tempo médio de resposta dos demais pacotes: Este teste tem por objetivo avaliar o tempo de transmissão dos demais pacotes entre os nós, utilizando requisições ICMP. O teste foi executado de forma semelhante ao teste do tempo médio de resposta do primeiro pacote, porém considerou-se apenas os pacotes seguintes ao primeiro. A tabela a seguir apresenta os resultados encontrados.

Quadro 27- Comparação do tempo de resposta a partir do segundo pacote

Hypervisor	**Tempo de resposta (médio)**
FlowVisor	0,096 ms
OpenVirteX	0,092 ms
OpenVisor	0,097 ms

As medidas do tempo médio de resposta a partir do segundo pacote, mostraram uma variação ainda menor do que no teste anterior. A média do tempo de resposta da comunicação do *OpenVisor* foi maior que das demais, porém percebeu-se que em ambas as ferramentas houve uma variação grande ao longo das 20 amostragens, tanto para mais quanto para menos, corroborando com os demais resultados de desempenho dos testes anteriores, que não houve impacto significativo do *framework* desenvolvido com as ferramentas citadas.

De forma geral, os testes de desempenho realizados não permitiram identificar diferenças reais entre as ferramentas, pelo menos no ambiente simulado, em razão das variações encontradas em cada teste realizado. Acredita-se que em um ambiente real os resultados podem divergir permitindo averiguar diferenças.

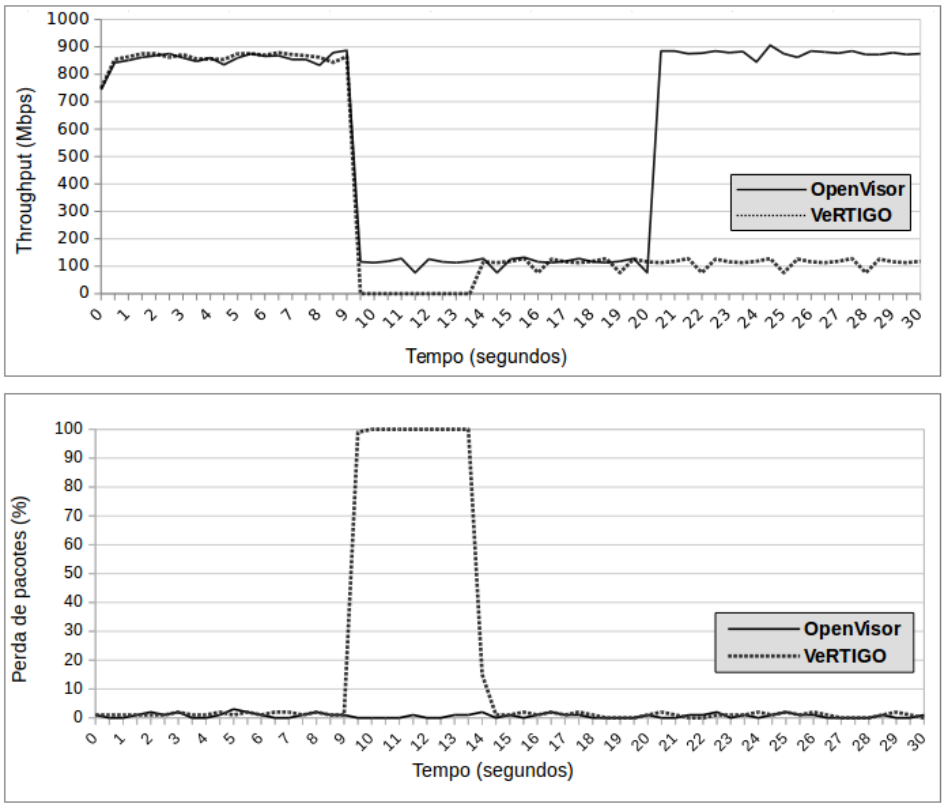
8.3.3 Recuperação de falhas

Neste experimento buscou-se comparar o tempo necessário para o restabelecimento de uma comunicação após a ocorrência de uma falha física na infraestrutura. Comparou-se os resultados do *OpenVisor* com a ferramenta *VeRTIGO*, apresentada na seção 2.3.1.2 e única dentre as demais a apresentar a funcionalidade de recuperação de falhas, além do OVX.

Os resultados demonstraram o que já era esperado, que o *Openvisor* detem vantagem neste quesito em virtude do método como realiza a convergência após a ocorrência de falhas. Logo quando o *OpenVisor* define a comunicação entre os *hosts*, é armazenado internamente além da rota principal, caminhos alternativos para que havendo uma falha no caminho principal seja alterado para uma rota alternativa sem a necessidade de um recálculo. Já o *VeRTIGO* opera de forma diferente e a cada falha na rede física é necessário realizar um novo cálculo de rotas para restabelecer a comunicação, gerando um tempo de convergência maior em comparação com o *OpenVisor*.

A figura 53 demonstra os testes realizados, aplicados identicamente no OpenVisor e no VeRTIGO. Utilizou-se a ferramenta *Iperf* para gerar tráfego UDP entre os *hosts* H1 e H3 por 30 segundos, analisando o *throughput* e a taxa de perda de pacotes. Em ambos os testes a comunicação iniciou utilizando a rota principal, que neste caso era o caminho identificado pela maior taxa de transmissão, 1000Mbits/s. Após 9 segundos, simulou-se uma falha no caminho principal, onde a comunicação estava ocorrendo, obrigando as ferramentas a realizar a mudança de rota por um caminho alternativo.

Figura 53 – Comparação do tempo de convergência entre o FV e o *OpenVisor*.



Fonte: Próprio autor.

Como resultado, verificou-se que o *VeRTIGO* levou aproximadamente 5 segundos para calcular uma nova rota e restabelecer a comunicação, cuja taxa de perda de pacotes foi de 0 para 100% no momento da falha até a retomada da comunicação. Já o *OpenVisor* em nenhum instante perdeu a conectividade, ficando com a perda de pacotes estável todo o tempo em aproximadamente 0%.

No instante 20 foi restabelecido o *link* principal anteriormente desativado para o teste, objetivando fazer as ferramentas alterar novamente os fluxos para a rota principal. Identificou-se que o *VeRTIGO* continuou utilizando a rota secundária, conforme é possível ver pelo *throughput* de aproximadamente 100Mbits/s, mesmo após a rota principal voltar a operar. O *OpenVisor*, verificando as duas rotas em funcionamento, converge para a melhor rota e retoma o caminho com a maior taxa de transmissão. Após a falha, a taxa de perda de pacotes fica estável para ambas ferramentas em 0%.

Desta forma, os testes realizados permitem indicar que o *OpenVisor* consegue restabelecer a comunicação dos fluxos com um tempo consideravelmente menor que o *VeRTIGO*. Ainda, através da simulação da infraestrutura de rede não foi detectado perda de conectividade na convergência de rotas do *OpenVisor*, diferente do *VeRTIGO* que levou aproximadamente 5 segundos para restabelecer a comunicação.

8.3.4 Síntese das comparações

Os testes de comparação desenvolvidos para avaliar o desempenho do *framework OpenVisor***,** com relação as ferramentas citadas neste trabalho, permitem explicitar o seguinte quadro comparativo:

Quadro 28- Comparação 1 das funcionalidades dos *hypervisors* de rede

Hypervisors	Topologia virtual e arbitrária	Flexibilidade	Resiliência a falhas	Complexidade	Controle absoluto	Tempo de convergência
VeRTIGO	X	X	X	BAIXA	-	Alto
FlowVisor	-	X	-	ALTA	-	-
OpenVirteX	X	-	X	BAIXA	X	Baixo
OpenVisor	**X**	**X**	**X**	**BAIXA**	**-**	**Baixo**

Como evidencia-se no quadro 28, o FV possui a flexibilidade como sua principal característica e é considerado com alta complexidade de operação, na perspectiva dos usuários. O *VeRTIGO*, possui o intuito de incrementar as funcionalidades do *ADVisor* e

adicionar a característica de recuperação de falhas. Logo, permite criar topologias virtuais e arbitrárias e sem impactar na flexbilidade das redes. Ainda, possui a característica de recuperação de falhas, porém em comparação com o OVX e consequentemente com o *OpenVisor*, seu tempo de convergência para falhas é consideravelmente maior.

Já o f*ramework OpenVisor* possui as funcionalidades de topologias virtuais e arbitrárias, recuperação de falhas e dispõe do controle absoluto das redes virtuais aos usuários. Porém, sua principal fragilidade é a falta de flexibilidade na definição das redes virtuais.

Considerando a literatura consultada, ainda, pode-se incluir nesta comparação as ferramentas *Double-FlowVisors* e *ADVisor,* o que é explicitado no quadro 29 que segue:

Quadro 29- Comparação 2 das funcionalidades dos *hypervisors* de rede

Hypervisors	Topologia virtual e arbitrária	Flexibilidade	Resiliência a falhas	Complexidade	Controle absoluto	Tempo de convergência
Double-FlowVisors	-	X	-	ALTA	-	-
ADVisor	X*	X*	-	BAIXA	-	-
OpenVisor	**X**	**X**	**X**	**BAIXA**	**-**	**Baixo**

O *Double-FlowVisors* possui as mesmas funcionalidades do FV, visto que sua única diferença é na utilização de duas instâncias do FV utilizadas recursivamente no mesmo cenário de rede.

Já o *ADVisor* surge com a característica de virtualização de topologia e arbitrariedade. Porém, a forma como implementa isso impacta na flexibilidade, pois ele utiliza métricas de VLAN para o mapeamento das informações físicas para virtuais. Ainda, sua complexidade é considerada baixa, pois permite criar topologias virtuais e arbitrárias. Porém, o *ADVisor* não traz nenhuma solução para recuperação de falhas.

Neste contexto, é possível indicar que o framework proposto é promissor para ambiente de experimentos tendo em vista seu funcionamento bem como as características oferecidas aos usuários.

9 CONCLUSÃO

O estudo realizado desenvolveu um *framework* para redes de experimentação *OpenFlow*, objetivando proporcionar aos usuários redes virtuais flexíveis, de baixa complexidade de operacionalização, dispondo de controle absoluto e resiliente a falhas. A hipótese que direcionou a investigação foi que através da integração das ferramentas *OpenVirteX* e *FlowVisor e* consequentemente de suas funcionalidades, o *framework* resultante possibilitaria atingir tal intuito.

A integração foi implementada e avaliada a partir de experimentos que contemplaram os quesitos de: funcionamento, funcionalidades, bem como comparativos com ferramentas já existentes.

No quesito funcionamento, os testes realizados detectaram que o *OpenVisor* apresenta comportamento adequado com a integração de ambas as ferramentas, sem identificação de falhas nos experimentos desenvolvidos, quais foram: a comunicação entre um simples servidor Web e um cliente, exemplificando uma comunicação normal e outra com um tráfego sensível através de uma transmissão de vídeo pela rede.

No quesito funcionalidade os testes permitiram evidenciar que a integração das ferramentas *OpenVirteX* e *FlowVisor* permite não somente unir as características de cada um dos *hypervisors*, como também solucionar algumas de suas fragilidades. No que se refere a união das características dos *hypervisors*, a integração resultou em um *framework* capaz de disponibilizar aos usuários de ambientes de experimentação *OpenFlow* redes com alta flexibilidade de segmentação – funcionalidade do FV; simples operacionalização através do uso de topologias virtuais e arbitrárias; resiliente a falhas através da recuperação automáticas a rupturas de conectividade entre *enlaces* – ambas características fornecidas pelo OVX.

Como fragilidades solucionadas, evidenciou-se que a integração permitiu superar as fragilidades do FV, no que se refere: a complexidade de operação nas redes virtuais; e a recuperação de conectividade na ocorrência de falhas físicas. Já com relação ao OVX, a fragilidade sanada foi a flexibilidade, resultando em uma ferramenta com alta flexibilização.

No quesito comparação com as demais ferramentas, o *OpenVisor* foi avaliado comparativamente a partir dos critérios de: quantidade de fluxos, desempenho, tempo de convergência. No que se refere ao número de fluxos, comparou-se exclusivamente com o FV visto sua ausência da característica de virtualização de topologia. Constatou-se que para o cenário de experimentação utilizado, o FV exige um número consideravelmente maior de

fluxos do usuário para permitir seus testes. Verificou-se nos testes que o FV exigiu quatro vezes mais fluxos que o mesmo teste realizado com o *OpenVisor*. Desta forma, concluiu-se que o FV possui uma complexidade de operação mais elevada que o *OpenVisor.*

Já o quesito desempenho, comparou-se tanto com o OVX quanto o FV. Realizou-se diferentes tipos de testes, a fim de encontrar diferenças significativas através do cenário de experimentação utilizado, tais como os seguintes: taxa de vazão, tempo médio de resposta do primeiro pacote e tempo médio de resposta dos demais pacotes. Os resultados encontrados não foram suficientes para avaliar diferenças significativas de desempenho entre as ferramentas analisadas. Os testes desenvolvidos servem para indicar a necessidade de novos estudos que contemplem testes de desempenho, em uma infraestrutura de rede *OpenFlow* real.

Por fim, verificou-se o tempo de convergência do *OpenVisor* comparado ao *VeRTIGO*, visto que foi a única ferramenta com a funcionalidade de recuperação de falhas, além do OVX. Evidenciou-se que o *VeRTIGO* leva um tempo consideravelmente maior para retornar a conectividade na ocorrência de falhas se comparado com o *OpenVisor*. Além disso, identificou-se que para o cenário de testes considerado, não foi notado perda de conectividade nos testes com o *OpenVisor*.

Nesta perspectiva, o estudo desenvolvido permite confirmar parte da hipótese norteadora da proposta uma vez que o *framework OpenVisor* se mostrou:

- **Flexível,** pois permite utilizar quaisquer métricas do cabeçalho *OpenFlow* para a segmentação das redes virtuais. Esta afirmação está ancorada no teste realizado com a definição de duas redes virtuais, sendo uma delas com controle sobre os fluxos com as métricas de protocolo TCP e porta 9999, enquanto a outra gerenciando quaisquer fluxos, desde que diferentes da anterior.
- **Baixa complexidade na operacionalização**, tendo em vista a utilização de uma topologia virtual e arbitrária composta por um único *switch* virtual referenciando-se a totalidade dos componentes da infraestrutura de rede. Os testes de usabilidade evidenciaram a facilidade de operação do *OpenVisor* quando comparado ao FV, pela quantidade consideravelmente menor de regras de fluxos necessárias para estabelecer as comunicações.
- **Resiliente a falhas de conectividade** tendo em vista que na testagem realizada de recuperação de falhas, a ferramenta se mostrou capaz de redefinir

a comunicação por uma rota alternativa, impedindo a falha de conectividade. Outro teste desenvolvido, que referenda esta funcionalidade foi a simulação de falhas de conexão no momento da transmissão de vídeo pela rede, o que não representou impacto significativo na comunicação.

No que se refere ao **controle absoluto** os resultados não referendaram a funcionalidade deste quesito, permitindo inferir que a flexibilidade e controle absoluto são características dissonantes no *framework* proposto, ou seja, o incremento da flexibilidade é incompatível com o controle absoluto. Observou-se que disponibilizar o controle total da rede para o usuário tem o impacto de fragilizar a flexibilidade do ambiente de experimentação.

Portanto, este trabalho teve como principal contribuição apresentar um *framework* para ser utilizado em redes de experimentação *OpenFlow* provendo as características de flexibilidade na criação de redes virtuais, baixa complexidade de operacionalização através do uso de topologias virtuais e arbitrárias e, ainda, com suporte total a recuperação de falhas físicas de conectividade.

Acredita-se que novos estudos poderão enriquecer e consolidar os resultados encontrados. Para tanto, o protótipo desenvolvido terá seu código disponibilizado como contribuição para futuros trabalhos. Desta forma, o *framework OpenVisor* poderá ser testado com maior variedade de cenários para ser aprimorado.

Para dar continuidade a esta pesquisa, torna-se interessante explorar alguns pontos discutidos, como por exemplo, realizar novos testes de desempenho, *overhead* e tempo de convergência do *OpenVisor* em uma infraestrutura SDN real e comparar com as demais ferramentas citadas neste trabalho. Outra proposta de trabalho futuro seria modificar as estratégias de engenharia de tráfego utilizadas pelo OVX. Atualmente o OVX considera apenas o estado do *link* para a definição do melhor caminho, desprezando a utilização. Logo, mesmo com uma rota principal saturada, o OVX não tem capacidade de modificar os fluxos para uma rota secundária, apenas após a falha de conectividade de algum *enlace*.

Acredita-se que os caminhos indicados são promissores de pesquisas e estudos capazes de fomentar a construção de ferramentas promotoras de espaços de experimentação que permitam aos pesquisadores e usuários, ambientes e infraestrutura reais, para o desenvolvimento de testes e avaliações cada vez mais aprimorados e adequados às necessidades emergentes.

REFERÊNCIAS BIBLIOGRÁFICAS

ADRICHEM, N. L. M. van. et al. **OpenNetMon: Network monitoring in openflow software-defined networks**, in Proc. IEEE Netw. Oper.Manage. Symp., 2014, DOI: 10.1109/NOMS.2014. 6838228.

AL-SHABIBI, A. et al. **OpenVirteX: a network hypervisor**. Open Networking Summit – ONS, Santa Clara, CA, 2014a.

ANDERSON, T. et al. **Overcoming the internet impasse through virtualization. Computer**, IEEE Computer Society Press, Los Alamitos, CA, USA, v. 38, n. 4, p. 34–41, abr. 2005. ISSN 0018-9162. Disponível em: <http://dx.doi.org/10.1109/MC.2005.136>.

BUENO, I. et al. **An OpenNaaS based SDN framework for dynamic QoS control**. Workshop on Software Defined Networks for Future Networks and Services, Trento, Italia. 2013. ISSN 978-1-4799-2781-4.

CONSORTIUM, O. **OpenFlow Tutorial**. 2012. Disponível em: http://tinyurl.com/7uu93yk. Acesso em: 15/07/2015.

BASTIN, N. et al. "**The InstaGENI initiative: An architecture for distributed systems and advanced programmable networks**. Special issue on Future Internet Testbeds", Computer Networks, Special issue on Future Internet Testbeds - Part I, vol. 61, pp. 24–38, Jan. 3, 2014, issn: 1389-1286.

BERMAN, M. et al. **Geni: A federated testbed for innovative network experiments**, Comput. Netw., vol. 61, pp. 5–23, Mar. 2014, issn: 1389-1286.

BLENK, A. et al. **Survey on Network Virtualization Hypervisors for Software Defined Networking**. IEEE Communications Surveys & Tutorials, p. 1–31, 2015.

BRITO, 2013. **Blog LabCisco**. Disponível em http://labcisco.blogspot.com.br/2013/07/paradigma-sdn-de-redes-programaveis.html. Acesso em: 15/04/2016.

BOZAKOV, Z. e PAPADIMITRIOU, P. **Towards a scalable software-defined network virtualization platform**, in 2014 IEEE Network Operations and Management Symposium (NOMS), May 2014, pp. 1–8.

CIUFFO, L. et al. **Testbed FIBRE: Passado, Presente e Perspectivas**. VII WORKSHOP DE PESQUISA EXPERIMENTAL DA INTERNET DO FUTURO (WPEIF), XXXIV Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos (SBRC), Salvador, Bahia, 2016.

CHEN, J-L. et al. **EnterpriseVisor: A Software-Defined Enterprise Network Resource Management Engine**. Proceedings of the 2014 IEEE/SICE International Symposium on System Integration, Chuo University, Tokyo, Japan, December 13-15, 2014

CHOWDHURY, N.; BOUTABA, R. **Network virtualization: state of the art and research challenges**. IEEE Commun. Mag, p. 20-26, 2009.

CHOI, T. et al. **SUMA: Software-defined Unified Monitoring Agent for SDN**. IEEE/IFIP NOMS – IEEE/IFIP Network Operations and Management Symposium: Management in a Software Defined World, 2014.

CORIN, R. D. et al. **Vertigo: Network virtualization and beyond**. European Workshop on Software Defined Networking (EWSDN), Oct. 2012, pp. 24–29

DUARTE, N. G. **OpenVirteX no ambiente FIBRE**. Disponível em: http://fibre.org.br/wp-content/uploads/2015/10/RF-OpenVirteX_CT-FIBRE.pdf. Acesso em 24/01/2016.

ELLIOT, D. **Exploring the Challenges and Opportunities of Implementing Software-Defined Networking in a Research Testbed**. 2015. Faculty of North Carolina State University.

FARIAS, F. et al. **Pesquisa Experimental para a Internet do Futuro: Uma Proposta Utilizando Virtualização e o Framework Openflow**. XXIX Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos – SBRC, p. 1-62, Campo Grande, MS, 2011.

FIBRE. **Future Internet Brazilian Environment for experimentation**. Disponível em: https://fibre.org.br/. Acesso em: 16/03/2016.

FLOODLIGHT, Project. **OpenFlow controller**. Disponível em: http://www.projectfloodl ight.org/floodlight/. Acesso em: 18/12/2015.

FlowSpace firewall, GlobalNOC - **FSFW: FlowSpace Firewall**, Disponível em: http://globalnoc. iu.edu/sdn/fsfw.html. Acesso em: 15/02/2016.

GENI. **Global environment for network innovations**. Disponível em http://www.geni.net. Acesso em: 16/03/2016.

GIATSIOS, D. et al. **Integrating FlowVisor access control in a publicly available OpenFlow testbed with slicing support**. TridentCom. Thessaloniki, Grecia. Junho, 2012.

GUEDES, D. et al. **Redes Definidas por Software : uma abordagem sistêmica para o desenvolvimento das pesquisas em redes de computadores**. XXX Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos - SBRC, [S.l.], p.160–210, 2012.

IPERF (2014) "**A TCP, UDP, and SCTP network bandwidth measurement tool**". Disponível em: https://github.com/esnet/iperf/. Acesso em: Janeiro, 2016.

KOBAYASHI, M. et al. **Maturing of OpenFlow and software-defined networking through deployments. Computer Networks**, Special issue on Future Internet Testbeds - Part I, vol. 61, pp. 151–175, Mar. 14, 2014, issn: 1389-1286.

KREUTZ, D. et al. **Software-Defined Networking: a comprehensive survey**. Proceedings of the IEEE, vol. 103, no. 1, pp. 14–76, Jan. 2015, issn: 0018-9219.

MATTOS, D. M. F. **Uma Arquitetura de Virtualização de Redes Orientada à Migração com Qualidade de Serviço**. Dissertação de Mestrado (Universidade Federal do Rio de Janeiro). 2012. Disponível em: http://www.gta.ufrj.br/ftp/gta/TechReports/diogo12.pdf. Acesso em: 02/05/2016

MCKEOWN, N. et al. **OpenFlow: enabling innovation in campus networks**. ACM SIGCOMM Computer Communication, p.1–6, 2008.

MININET (2014) "**Overview**". Disponível em: http://Mininet.org/overview. Acesso em: Janeiro, 2016.

MUSSMAN, H. **GENI: An Introduction**. 2012. Disponível em http://groups.geni.net/geni/raw-attachment/wiki/GeniSysOvrvw/Geni-Anintroduction-28Feb2012.pdf. Acesso em: 04/01/2016.

MYERS, G. **The Art of Software Testing**, Second edition. 2004.

NEVADO, P. P et al. **Popper e a investigação: a metodologia hipotética – dedutiva**, 2008. Universidade Técnica de Lisboa. Instituto Superior de Economia e Gestão.

NITOS testbed. Disponível em: http://nitlab.inf.uth.gr/NITlab/nitos/openflow. Acesso em 12/04/2016.

OFELIA. **Openflow in europe - linking infrastructure and applications**. Disponível em: http://www.fp7-ofelia.eu. Acesso em: 16/03/2016.

ONF - Open Networking Fundation. **Software-Defined Networking: The New Norm for Networks**. ONF White Paper. Palo Alto, US: Open Networking Foundation, 2012.

PECHLIVANIDOU, K. et al. **NITOS Testbed: A Cloud based Wireless Experimentation Facility,** FIDC, 26th International Teletraffic Congress (ITC 26). Karlskrona, Sweden, 2014

PRESSMAN, R. **Engenharia de Software - Uma abordagem profissional**. 7 Edição ed. 2011.

PUPATWIBUL, P. New Information Model that Allows Logical Distribution of the Control Plane for Software-Defined Networking. 2016. Tese de doutorado - University of Technology Sydney.

ROTHENBERG, C. E, et al. **OpenFlow e redes definidas por software: um novo paradigma de controle e inovação em redes de pacotes**, Cad. CPqD Tecnologia, Campinas, v.7, p.65-76, 2011.

SALMITO, T. et al. **FIBRE – an international testbed for future internet experimentation**, in Simpsio Brasileiro de Redes de Computadores e Sistemas Distribudos-SBRC 2014, 2014, p–969.

SALVADORI, E. et al. **Generalizing virtual network topologies in OpenFlow-based networks**. IEEE Global Telecommunications Conference (GLOBECOM 2011), Dec. 2011, pp. 1–6.

SHERWOOD, R. et al. **FlowVisor A Network Virtualization Layer**. Technical Report Openflow, Stanford University, 2009.

SHERWOOD, R. et al. **Can the production network be the testbed? Operating Systems Design and Implementation**, ser. OSDI'10, Berkeley, CA, USA: USENIX Association, 2010, pp. 1–6.

SU, M. et al. **Design and implementation of the OFELIA FP7 facility: The european OpenFlow testbed**, Computer Networks, Special issue on Future Internet Testbeds Part I, vol. 61, pp. 132–150, Mar. 14, 2014, issn: 1389-1286.

SUÑÉ, M. et al. **Design and implementation of the OFELIA FP7 facility: The European OpenFlow testbed**. Computer Networks, 2013.

Stanford University (2016). Disponível em: https://openflow.stanford.edu/display/SDEP/home. Acesso em: Feveiro, 2016.

YAP, K. et al. **The stanford OpenRoads deployment**, in Proceedings of the 4th ACM International Workshop on Experimental Evaluation and Characterization, ser. WINTECH '09, New York, NY, USA: ACM, 2009, pp. 59–66, isbn: 978-1-60558-740-0.

YIN, X. et al. **Software defined virtualization platform based on double-FlowVisors in multiple domain networks**. Proc. CHINACOM, no. 1, pp. 776–780, 2013.