OVX^{R} - *OpenVirteX* Recursivo Integrado ao *FlowVisor* para Utilização em Ambientes de Experimentação *OpenFlow*

Lucas Powaczuk[1], Eduardo R. Cargnelutti[1], Gabriel Marchesan[1], Ricardo B. Gomes[1], Roseclea D. Medina[1]

[1]GRECA – Grupo de Redes e Computação Aplicada
PPGI – Programa de Pós-Graduação em Informática – Universidade Federal de Santa Maria (UFSM) – Santa Maria, RS - Brasil

{lucaspwk, ecargnelutti, gmarchesan, ricardo, rose}@inf.ufsm.br

Abstract. *This paper proposes a framework incorporated into OpenFlow experimentation networks. This framework is the integration of hypervisors OpenVirteX (OVX) and FlowVisor (FV), providing to users flexible virtual networks with arbitrary topologies, as well as resilience to connectivity failures. For this purpose, it was necessary to add a feature of recursion in OVX, what was called as OVX^{R} (OVX Recursive). As proof of concept, the initial experiment showed that through the OVX^{R}'s integration with FV it is possible to provide a resilient environment along with complete abstraction of infrastructure and flexible slicing policies for flows segmentation.*

Resumo. *Este trabalho propõe um framework para ser incorporado em redes de experimentação OpenFlow. Trata-se da integração dos virtualizadores de rede OpenVirteX (OVX) e FlowVisor (FV), disponibilizando aos usuários redes virtuais flexíveis, com topologias arbitrárias e resilientes a falhas de conectividade. Para isso, fez-se necessário adicionar ao OVX o suporte a recursividade, denominando-o como OVX^{R} (OpenVirteX Recursivo). Como prova de conceito, o experimento inicial mostrou que, através da integração do OVX^{R} ao FV, é possível oferecer um ambiente resiliente com total abstração da infraestrutura utilizando políticas de fatiamento flexíveis para a segmentação de fluxos.*

1. Introdução

A Internet desde sua criação passou por inúmeras mudanças para viabilizar sua ampliação. Porém apesar dos diversos ajustes, ainda se utiliza a mesma arquitetura do projeto inicial, resultando no surgimento de inúmeras limitações no modelo TCP/IP. Por conta disso, Farias *et al*. (2011) acreditam que uma das soluções efetivas para os problemas existentes seja um redesenho da atual arquitetura da Internet. No entanto, o desenvolvimento de uma nova arquitetura para a Internet é um trabalho complexo e delicado. Um dos principais obstáculos refere-se à validação das propostas, em virtude da inviabilidade de testes em infraestruturas reais em produção.

Nesta direção, surgiram as redes de experimentação (*testbeds*), com o intuito de disponibilizar ambientes para a realização de experimentos para que pesquisadores possam testar e desenvolver novas tecnologias em infraestruturas de redes reais obtendo resultados mais significativos. A premissa é que estes ambientes suportem múltiplas redes lógicas simultaneamente sobre a mesma infraestrutura física e possibilite a programabilidade por parte dos usuários. Uma das soluções que vem se destacando e

atendendo estas características é o *framework OpenFlow* [McKeown *et al*., 2008], pois possibilita aos experimentadores utilizar a rede de produção para testar seus protocolos e arquiteturas experimentais, de forma isolada do tráfego de produção. Outra característica importante é a possibilidade através de um controlador especializado de rede (*hypervisor*), de prover a virtualização e abstração da infraestrutura.

Atualmente existem vários *testbeds OpenFlow* em funcionamento, cujo os mais conhecidos são: o GENI [Geni, 2011], FIBRE [Fibre, 2011] e OFELIA [Ofelia, 2011]. A maioria dos *testbeds OpenFlow*, incluindo os citados anteriormente, utilizam o *FlowVisor* (FV) [Sherwood *et al*., 2009] como *hypervisor* responsável pela virtualização da rede, permitindo que diversas redes virtuais, denominadas *slices*, operem simultaneamente e independentemente sobre a mesma infraestrutura física. Sua principal característica é a flexibilidade na criação de redes virtuais, podendo criá-las por diversas métricas inclusas nos cabeçalhos de fluxos *OpenFlow*.

No entanto, o FV possui algumas fragilidades que ainda são pouco exploradas, das quais incluem-se: a) a impossibilidade de operar com topologias virtuais e arbitrárias, ou seja, desassociadas dos componentes da rede física, dificultando a operacionalização e aumentando a complexidade dos testes por parte dos operadores; b) ausência do controle absoluto pelos experimentadores, visto que a definição dos *slices* se dá através de métricas, consequentemente um único operador não pode controlar a totalidade de métricas; e finalmente, c) não é tolerante a rupturas de *enlaces*, portanto na ocorrência de falhas, o FV não consegue restabelecer a conexão às redes virtuais. A partir das limitações do FV, *testbeds* como o GENI, OFELIA, entre outros, estão em busca de ferramentas alternativas ao FV [Elliott, 2015].

O FV, sendo o *hypervisor OpenFlow* pioneiro, serviu de base para a criação de outros, como é visto nos trabalhos de [Salvadori *et al*., 2011], [Corin*et al*., 2012] e [Yin *et al*., 2013]. No entanto, nenhum destes solucionou por completo as fragilidades do FV. Por sua vez, surge o *OpenVirteX* (OVX) [Al-Shabibi *et al*., 2014] com o propósito de superar de fato estas limitações, proporcionando aos operadores o controle absoluto das redes; a utilização de topologias virtuais e arbitrárias; e redes resilientes a falhas físicas. Através destas características, é possível criar redes com topologias mais simples através da abstração dos nós e *links* físicos, com total controle ao usuário na operacionalização de seus testes sobre sua rede virtual e ainda dispor de um ambiente com *failover* nativo, ou seja, com recuperação automática de falhas de conectividade. Entretanto, apesar de suprir as fragilidades do FV, o OVX também possui suas limitações. A principal delas é a incapacidade de criar redes virtuais flexíveis da mesma forma que o FV, por métricas do cabeçalho *OpenFlow*.

Considerando este contexto, este trabalho tem por objetivo propor um *framework* para ser incorporado em redes de experimentação *OpenFlow*. Trata-se da integração do virtualizador de rede OVX ao FV, disponibilizando aos usuários redes virtuais flexíveis, com uma topologia virtualizada e arbitrária, desacoplada do substrato físico, e ainda resiliente a falhas de conectividade entre enlaces. No entanto, testes iniciais elucidaram que o OVX, diferentemente do FV, não suporta o funcionamento recursivo, com mais de um *hypervisor* sendo executados simultaneamente sobre a infraestrutura subjacente. Portanto, como tarefa inicial da integração, foi necessário adicionar a funcionalidade de recursividade ao OVX, denominando-o como OVX^R (*OpenVirteX* Recursivo).

2. Virtualização de Redes *OpenFlow*

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 [Farias *et al*., 2011]. Análoga a virtualização tradicional, a virtualização em redes *OpenFlow* também possui uma camada de abstração de *hardware*. Em 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.

2.1 *FlowVisor*

O *FlowVisor* (FV) foi o primeiro *hypervisor* criado para virtualização de Redes Definidas por *Software* (SDN) baseado no protocolo *OpenFlow* [Sherwood *et al*., 2009]. A principal contribuição do FV foi possibilitar que redes de produção e experimentais possam coexistir simultaneamente sobre a mesma infraestrutura física, isolando o tráfego da rede experimental do tráfego de produção.

Este *hypervisor* segmenta a rede em fatias (*slices*), operando como um controlador de rede responsável apenas pela divisão do espaço de endereçamento disponível na rede *OpenFlow* [Guedes *et al*., 2012]. Cada *slice* é composto por um conjunto de fluxos com lógicas de encaminhamento distintas, também chamado de *flowspace* [Sherwood *et al*., 2009]. O FV aloca para cada *slice* a sua própria fatia do *flowspace*, ou seja, o seu espaço de endereçamento de campos do cabeçalho *OpenFlow*, garantindo desta maneira que as redes virtuais não se sobreponham [Blenk *et al*., 2015]. Logo, os *slices* podem ser definidos pelos seguintes campos: portas dos *switches* (*Layer* 1); *Mac-Address* de origem e destino (*Layer* 2); endereços IP de origem e destino (*Layer* 3) e portas TCP/UDP de origem e destino (*Layer* 4).

Ainda que o FV seja capaz de suportar diversos controladores, conferindo uma fatia a cada um deles, o mesmo possui algumas limitações, como a de não suportar a utilização de topologias virtuais e arbitrárias, ou seja, topologias virtuais desassociadas dos componentes da rede física. Logo, entende-se que não é possível criar topologias com abstrações da rede física, como exemplo, dois ou mais comutadores físicos não podem ser abstraídos por apenas um *switch* virtual, dificultando a operacionalização e aumentando a complexidade por parte dos operadores [Blenk *et al*., 2015], exigindo que conheçam detalhadamente os componentes da rede física (comutadores, portas e *links*) para operar seus testes em seus *slices*.

Outra limitação do FV é referente ao compartilhamento do *flowspace*. Cada *slice* recebe uma fatia do *flowspace* da rede, fazendo-se necessário o compartilhamento dos campos do cabeçalho *OpenFlow* para distinguir os *slices*, não possibilitando nenhuma rede virtual controlar e operar sobre a totalidade do *flowspace*. Com isso Al-Shabibi *et al.* (2014) afirmam que o FV não suporta uma virtualização completa para as redes virtuais.

Ainda, o FV apresenta uma limitação de recuperação de falhas físicas, ou seja, não possui um sistema dinâmico para re-rotear fluxos em resposta a falhas de conectividade e rupturas de enlaces na rede física. Logo, é necessário que o experimentador programe manualmente caminhos redundantes para o seu *slice*, especificando as rotas a serem tomadas caso uma falha ocorra. Ademais, com base nas limitações apresentadas, surgiram diversas iniciativas de implementações sobre o FV a

fim de solucioná-las. Dentre elas, pode-se citar *ADVisor*, *VeRTIGO* e *Double-FlowVisors*, abordados na seção de trabalhos relacionados.

2.2 *OpenVirteX*

O *OpenVirteX* (OVX) foi criado com objetivo de simplificar e adicionar flexibilidade no processo de provisionamento de redes [Al-Shabibi *et al*., 2014]. No processo de customização da topologia, é possível criar topologias de forma arbitrária, ou seja, desassociadas dos componentes da infraestrutura física. Logo, um *link* virtual pode ser mapeado para múltiplos *links* físicos e vice-versa. Também, um *switch* virtual pode ser referenciado por múltiplos *switches* físicos, sendo chamado de *big switch*.

Assim, uma rede física complexa pode ser abstraída para uma topologia virtual mais simples, com menos nós e ligações, facilitando a operacionalização por parte dos experimentadores. Al-Shabibi *et al*. (2014) corroboram esta ideia afirmando que o acoplamento de componentes físicos e virtuais na forma de mapeamento N:1, ou seja, um conjunto de componentes físicos mapeados para um virtual, apresenta um ganho significativo de versatilidade, possibilitando duas importantes funcionalidades para as Redes Definidas por *Softwares* Virtuais (vSDNs):

a) Topologia customizável: vSDNs podem criar topologias virtuais sem a necessidade de restringir sua topologia real. Um *link* virtual pode abranger vários saltos contínuos e *switches* virtuais podem abstrair partes ou a totalidade de uma rede.

b) Resiliência: Um *link* ou *switch* virtual pode ser mapeado em vários componentes físicos para fornecer redundância. Um *link* virtual resiliente é caracterizado por múltiplos caminhos físicos até o *host* final.

Portanto, através da virtualização da topologia física, o OVX cria redes virtuais resilientes, as quais reagem de forma transparente e automática a problemas na infraestrutura, como exemplo de rupturas de *link*, para assegurar o transporte contínuo nas vSDNs. Logo, essa característica possibilita aos usuários a total abstração da infraestrutura subjacente, sem a necessidade de se preocupar com falhas de conectividade, pois a própria infraestrutura é resiliente e já provê essa funcionalidade.

Além destas funcionalidades, Blenk *et al*. (2015) afirmam que o OVX soluciona o problema do *flowspace*, possibilitando para cada vSDN o uso do espaço completo dos campos do cabeçalho *OpenFlow*. A fim de conseguir isso, o OVX coloca *switches* de borda (*edge switchs*) nas fronteiras da rede SDN física. O *edge switch* tem a tarefa de atribuir endereços IP e MAC virtuais para cada vSDN, e posteriormente se desfazer deles para serem usados dentro da rede SDN física. Logo, é possível isolar as vSDNs sem a necessidade de fatiar o *flowspace*. Ainda, garante o correto mapeamento dos endereços virtuais e físicos. Com esta abordagem de mapeamento, todo o *flowspace* pode ser fornecido para cada vSDN.

No entanto, apesar do OVX possuir funcionalidades não encontradas no FV, o mesmo possui limitações que não estão presentes no FV. Por exemplo, visto que o OVX não permite fatiar o *flowspace*, não é possível criar vSDNs tão flexíveis como no FV, que realiza o fatiamento através de métricas de identificação de fluxos. Ademais, o OVX não permite associar um mesmo *host* a duas ou mais vSDNs, visto que cada rede possui o *flowspace* completo para utilização e o mesmo *host* não poderá responder para duas ou mais redes distintas, diferentemente do FV o qual permite que um *host* possa

pertencer a diversos *slices*. Portanto, o OVX não permite separar redes por tráfegos distintos, tornando-o assim menos flexível que o FV.

3. Trabalhos Relacionados

Pode-se encontrar alguns trabalhos desenvolvidos recentemente abordando os principais conceitos, características, aplicações e implementações utilizando diferentes virtualizadores de redes *OpenFlow*. O FV, por sua vez, é considerado o virtualizador de redes *OpenFlow* mais utilizado atualmente, destacando-se em pesquisas acadêmicas e utilizado em redes de produção e *testbeds*. Por este motivo, serviu de base para o desenvolvimento de novos *hypervisors*, dentre eles pode-se citar: *ADVisor* [Salvadori *et al*., 2011], *VeRTIGO* [Corin *et al*., 2012] e *Double-FlowVisors* [Yin *et al*., 2013].

Salvadori *et al*. (2011) apresentam o *ADVisor* (*ADvanced FlowVisor*), o qual fornece uma camada para a criação de topologias virtuais básicas, restritas aos componentes da topologia física. O *ADVisor* supera a incapacidade do FV em criar topologias virtuais, no entanto, limita-se ainda por não criar topologias arbitrárias desassociadas dos componentes da infraestrutura física, ou seja, com a abstração de nós e ligações. Ainda, limita-se ao uso de cabeçalhos de VLAN para o mapeamento das topologias virtuais.

Corin *et al*. (2012) apresentam o *hypervisor VeRTIGO* (*ViRtual TopologIes Generalization in OpenFlow networks*). Assim como o OVX, ele permite trabalhar com topologias arbitrárias possibilitando abstrair toda a infraestrutura física em um simples componente virtual e ainda com suporte a recuperação automática de falhas, re-roteando os fluxos das redes virtuais. No entanto, o *VeRTIGO* necessita de um alto tempo de convergência para o restabelecimento da conexão, pois necessita de novos cálculos de rota a cada falha de enlace, diferentemente do OVX, que logo no início quando define o melhor caminho para o determinado fluxo, armazena também rotas secundárias, não exigindo re-cálculo na ocorrência de falhas.

O trabalho de Yin *et al*. (2013) apresenta uma plataforma de virtualização baseada em dois FVs sobre a mesma infraestrutura, denominada *Double-FlowVisors*, introduzindo o conceito de recursividade na utilização de *hypervisors*. Basicamente o primeiro FV opera entre os *switches OpenFlow* e descreve a topologia física para as camadas acima. O segundo FV é responsável por calcular, manter topologias virtuais específicas para as aplicações da rede e gerar a topologia de rede requerida por elas. Entretanto, diferentemente do OVX, esta abordagem possui a limitação de não abstrair totalmente a topologia física, não virtualizando *switches, links* e portas, restringindo-se apenas a delegação de fatias virtuais da rede física para cada controlador.

A proposta deste trabalho difere-se dos trabalhos citados por apresentar uma arquitetura para ambientes de experimentação baseada na integração entre os *hypervisors* OVX e FV, contemplando as características e funcionalidades de ambos. Supera o *ADVisor* pois permite criar redes virtuais arbitrárias e com suporte de *failover*. Em relação ao *VeRTIGO*, se sobrepõe pois além de permitir o controle total do *flowspace* para cada vSDN, possui um tempo de convergência possivelmente menor, visto que não necessita re-calcular rotas na ocorrência de falhas. Por fim, se difere do *Double-FlowVisors* por apresentar o OVX operando de forma integrada ao FV e não duas instâncias do FV recursivamente no mesmo cenário.

4. Procedimentos de Desenvolvimento e Arquitetura Proposta

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 por sua vez, intermedia a comunicação com os controladores *OpenFlow*. A seguir são apresentados os procedimentos aplicados durante o desenvolvimento da proposta e uma visão geral da solução.

No funcionamento padrão do FV, é enviada uma mensagem de "*flow_delete*" do tipo "*OFPT_FEATURES_REQUEST*" para os dispositivos na infraestrutura, com o objetivo de deletar quaisquer regras armazenadas nos *switches* subjacentes. Já o OVX quando se conecta aos controladores, inicialmente envia uma mensagem de "*send_hello*" do tipo "*OFPT_HELLO*", e aguarda outra mensagem de retorno "*recv_hello*", para finalizar o processo de estabelecimento de conexão, chamado de *handshake OpenFlow*. No caso do OVX não receber a mensagem de retorno correta, é fechado o canal de comunicação com o controlador, neste caso, com o FV.

Portanto, identificou-se que a falha de conectividade entre o OVX e o FV ocorria no estabelecimento da conexão, visto que o FV enviava uma mensagem da qual o OVX não estava esperando, antes mesmo de terminar o processo de *handshake*. Logo, foi necessário modificar a classe responsável pelo processo de estabelecimento de conexão do OVX com os controladores, neste caso, com o FV, de modo a aceitar as mensagens vindas do FV, mesmo que diferentes da esperada (*recv_hello*).

Após a modificação, o OVX passou a aceitar mensagens de "*flow_delete*", vindas do FV no momento de estabelecimento da conexão, conseguindo finalizar o *handshake* e estabelecendo conexão. Adicionada a funcionalidade de recursividade ao OVX, denominou-se a solução como *OpenVirteX* Recursivo (OVX^R). Após o funcionamento da integração, foram realizados testes com o uso do OVX^R junto ao FV, que demonstraram a prova de conceito referente ao funcionamento da integração.

5. Testes de Validação

Como prova de conceito, os primeiros experimentos foram realizados em um cenário *OpenFlow* simulado por meio do *Mininet* [Mininet, 2014]. Para este cenário, definiu-se uma rede idêntica a apresentada na Figura 1**(a)**, onde a topologia física é composta por *links* de 1000 Mbps ligados aos *hosts* e entre os comutadores A-C-D. Já a ligação entre os comutadores A-B-D foi definida em 100 Mbps. Utilizou-se o OVX^R para a definição de uma rede virtual resiliente com topologia virtual arbitrária. A rede criada na Figura 1 **(b)** é composta por apenas um *switch* virtual denominado "SV", que abstrai os quatro *switches OpenFlow* da rede física (A, B, C e D), bem como seus *enlaces*.

Já o FV que recebe as informações do OVX^R, possui a perspectiva da topologia criada pelo OVX^R, conforme mostrado na Figura 1**(b)**, e a função de segmentar a rede virtual criada em *slices*, adicionando a flexibilidade no fatiamento e atribuindo a cada *slice* o seu espaço de endereçamento de *flowspace*. Por fim, os controladores são conectados ao FV, que controla as comunicações entre os *slices* e as destina ao OVX^R. Cada controlador apresentado na Figura 1**(c)** é destinado a um usuário, e sua utilização ocorre de acordo com seus interesses.

Para o experimento, utilizou-se a ferramenta *Iperf* [Iperf, 2014], inclusa no *Mininet*, gerando tráfego entre os *hosts* H1 e H3, sob gerência do controlador A, em intervalos de 0.5 segundo, tempo mínimo suportado pela ferramenta. O controlador, por

ter a visão somente da topologia virtual, desconhece o caminho real que o tráfego percorre. Logo, o OVX^R define a rota que o fluxo irá percorrer na rede física, por meio do algoritmo SPF (*Shortest Path First*).

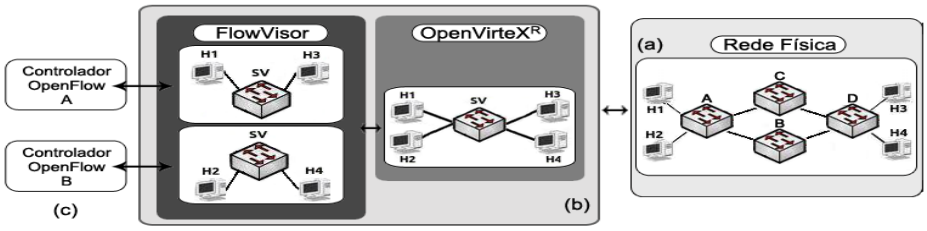

Figura 1. OVX^R - Arquitetura da solução proposta

Inicialmente, o OVX^R estabeleceu a comunicação entre H1 e H3 pelo caminho A-C-D (*links* de maior largura de banda). Depois de 10 segundos, foi simulada uma falha no *enlace* A-C. O OVX^R encarregou-se de manter a conectividade, alterando automaticamente a rota para A-B-D. No instante 20, restabeleceu-se a comunicação A-C, permitindo novamente que o OVX^R retomasse a rota original para A-C-D, demonstrando a resiliência da rede através do *failover* (Figura 2).

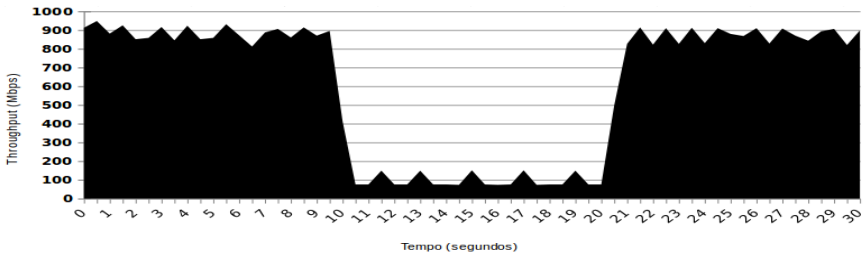

Figura 2. Alternância de rotas através do *failover* do OVX^R

6. Considerações Finais

No âmbito de redes de experimentação *OpenFlow*, o FV destaca-se como a principal ferramenta para habilitar a virtualização de rede. No entanto ainda apresenta algumas limitações, como a falta de suporte para topologias virtuais arbitrárias; o compartilhamento de *flowspace;* e a ausência de recuperação automática para falhas de conectividade, fazendo inclusive alguns *testbeds* considerar novas ferramentas para a substituição do FV. Por sua vez, as funcionalidades inexistentes no FV podem ser encontradas no OVX, visto que ele proporciona redes virtuais com controle do *flowspace* completo, topologias arbitrárias e resilientes a falhas físicas. No entanto, o OVX não possui a mesma flexibilidade que o FV na virtualização da rede por não segmentar a rede de acordo com as métricas do cabeçalho dos fluxos *OpenFlow*.

Em suma, as duas ferramentas se complementam, pois através desta integração pode-se contar com diferentes políticas de segmentação de tráfego bem como a virtualização da topologia subjacente da rede e a capacidade de resiliência (*failover*). Portanto, este trabalho apresentou uma solução para fornecer aos operadores de *testbeds OpenFlow* as funcionalidades existentes em ambas as ferramentas, por meio da adição da característica de recursividade na arquitetura do OVX. Reconhece-se a necessidade de novos testes de modo a validar as características e limitações desta integração a fim

de obter resultados mais precisos. Assim, em trabalhos futuros pretende-se realizar novos testes focando na eficácia da solução criada, de modo a medir e comparar com as demais soluções existentes, verificando quesitos como desempenho, *overhead* e tempo de convergência. Ademais, é importante ressaltar que em um ambiente real, os resultados podem ser diferentes, tendo em vista que os apresentados até então se deram em um ambiente simulado e controlado.

Referências

Al-Shabibi, A. *et al*. (2014) "OpenVirteX: A Network Hypervisor", emOpen Networking Summit - ONS, Santa Clara, CA.

Blenk, A. *et al*. (2015) "Survey on Network Virtualization Hypervisors for Software Defined Networking", emIEEE Communications Surveys & Tutorials, p. 1-31.

Corin, R. D. *et al*. (2012) "Vertigo: Network Virtualization and Beyond", emEuropean Workshop on Software Defined Networking - EWSDN, p. 24-29.

Elliott, S. D. (2015)"Exploring the Challenges and Opportunities of Implementing Software DefinedNetworking in a Research Testbed", Faculty of North Carolina State University.

Farias, F. *et al*. (2011) "Pesquisa Experimental para a Internet do Futuro: Uma Proposta Utilizando Virtualização e o Framework OpenFlow", em XXIX Simpósio Brasileiro deRedes de Computadores e Sistemas Distribuídos-SBRC, p.1-62.

Fibre (2011) "Future Internet Brazilian Environment for Experimentation".Disponível em: https://fibre.org.br. Acesso em: Maio, 2016.

Geni (2011) "Global Environment for Network Innovations".Disponível em: http://www.geni.net. Acesso em: Maio, 2016.

Guedes, D. *et al*. (2012) "Redes Definidas por Software: Uma Abordagem Sistêmica para o Desenvolvimento das Pesquisas em Redes de Computadores", em XXX Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos - SBRC, p. 160-210.

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

McKeown, N. *et al*. (2008) "Enabling Innovation in Campus Networks", emACM SIGCOMM Computer Communication, p.1-6.

Mininet (2014) "Overview". Disponível em: http://Mininet.org/overview. Acessoem: Maio, 2016.

Ofelia (2011) "OpenFlow in Europe - Linking Infrastructure and Applications".Disponível em: http://www.fp7-ofelia.eu. Acesso em: Maio, 2016.

Salvadori, E. *et al*. (2011) "Generalizing Virtual Network Topologies OpenFlow-based Networks", em IEEE Global Telecommunications Conference, p.1-6.

Sherwood, R. *et al*. (2009) "FlowVisorA Network Virtualization Layer",emTechnical Report Openflow, Stanford University.

Yin, X. *et al.* (2013) "Software Defined Virtualization Platform Based on Double-FlowVisors in Multiple Domain Networks", emCHINACOM, no. 1, p. 776-780.