Alexis Seiki Saito

Implementação de Suporte à Alta Disponibilidade em ambientes NFV usando OSM

Sorocaba, SP

23 de Maio de 2018

Alexis Seiki Saito

Implementação de Suporte à Alta Disponibilidade em ambientes NFV usando OSM

Dissertação de mestrado apresentada ao Programa de Pós-Graduação em Ciência da Computação (PPGCC-So) da Universidade Federal de São Carlos como parte dos requisitos exigidos para a obtenção do título de Mestre em Ciência da Computação. Linha de pesquisa: Engenharia de Software e Sistemas de Computação.

Universidade Federal de São Carlos – UFSCar

Centro de Ciências em Gestão e Tecnologia – CCGT

Programa de Pós-Graduação em Ciência da Computação – PPGCC-So

Orientador: Prof. Dr. Fábio Luciano Verdi

Sorocaba, SP

23 de Maio de 2018

Saito, Alexis Seiki
Implementação de Suporte à Alta Disponibilidade em ambientes NFV usando OSM/ Alexis Seiki Saito. – 2018.
79 f. : 30 cm.

Dissertação (Mestrado) – Universidade Federal de São Carlos – UFSCar
Centro de Ciências em Gestão e Tecnologia – CCGT
Programa de Pós-Graduação em Ciência da Computação – PPGCC-So.
Orientador: Prof. Dr. Fábio Luciano Verdi
Banca examinadora: Prof. Dr. Fábio Luciano Verdi, Prof. Dr. Edmundo Roberto Mauro Madeira, Profª. Drª. Yeda R. Venturini
Bibliografia

1. Alta Disponibilidade. 2. Virtualização de Função de Rede. 3. Gerenciamento e Orquestração de Rede. I. Prof. Dr. Fábio Luciano Verdi. II. Universidade Federal de São Carlos. III. Implementação de Suporte à Alta Disponibilidade em ambientes NFV usando OSM

UNIVERSIDADE FEDERAL DE SÃO CARLOS

Centro de Ciências em Gestão e Tecnologia
Programa de Pós-Graduação em Ciência da Computação

Folha de Aprovação

Assinaturas dos membros da comissão examinadora que avaliou e aprovou a Defesa de Dissertação de Mestrado do candidato Alexis Seiki Saito, realizada em 23/05/2018:

Prof. Dr. Fabio Luciano Verdi
UFSCar

Prof. Dr. Edmundo Roberto Mauro Madeira
UNICAMP

Profa. Dra. Yeda Regina Venturini
UFSCar

Dedico essa dissertação primeiramente à minha mãe Cristina Tisuko Maeda Saito que dedicou sua vida a mim e a meus irmãos. Dedicatória especial a Paula Regiane Guarnieri que compartilha comigo esse caminho que é viver. Finalmente, dedico também a todos que me ajudaram (e não foram poucos) a construir a minha história de vida até aqui, em especial ao meu tio João Casagrande (in memorian).

Agradecimentos

Agradeço,

a minha noiva Paula que, com amor e carinho, deu seu apoio e dedicação incondicionais, sempre acreditando em meu potencial e estando presente nos momentos difíceis.

a minha mãe Cristina por me apoiar e compreender a minha ausência, além de revisar o texto.

ao meu gerente Rogério de Sousa Aguiar da Ericsson por incentivar e compartilhar da visão de futuro do desenvolvimento tecnológico.

ao meu colega André Luiz Beltrami Rocha que ajudou bastante na manutenção do laboratório e outros suportes.

Finalmente, agradeço ao meu orientador Prof. Dr. Fábio Luciano Verdi pela dedicação, ensinamentos, orientação e paciência nessa longa jornada que foi o mestrado.

"Aquilo que não nos mata, nos torna mais fortes."
(Twilight of the Idols, Friedrich Wilhelm Nietzsche)

Resumo

A alta disponibilidade é o aspecto mais importante quando discutimos a resiliência de sistemas de telecomunicações no contexto da adoção do paradigma de computação em nuvem, um grande desafio para as operadoras de telecomunicações. Iniciativas de implementações de plataformas NFV como o OpenMANO, fortemente baseado na padronização NFV da ETSI, servem de base para experimentações como as propostas neste projeto de mestrado em que ilustramos nossa contribuição para a implementação do gerenciamento da alta disponibilidade na plataforma Open Source MANO (OSM). A implementação de alta disponibilidade apresentada neste trabalho é aderente à padronização ETSI e contribui para um dos mais importantes frameworks NFV que até então não apresentava o suporte a este item. Neste projeto, apresentamos uma experimentação sobre a plataforma proposta pelo OSM que envolve todo o ambiente com sua complexidade tecnológica. Em termos concretos, nossa contribuição se insere em dois aspectos principais: uma proposta de modelagem desta informação baseada na padronização ETSI sobre alta disponibilidade e sua implementação desta no framework OSM. Tal implementação consiste na detecção de falha pelo orquestrador implicando em uma reconfiguração de rede como mecanismo de suporte à redundância em um cenário ativo-ativo. O experimento foi testado e avaliado em um ambiente real minimalista a fim de comprovar o funcionamento da implementação realizada para gerenciar a informação de alta disponibilidade e reorquestrar fluxos de dados entre VNFs redundantes.

Palavras-chaves: Alta disponibilidade. Virtualização de Função de Rede. Gerenciamento e Orquestração de Rede.

Abstract

High Availability is the most important aspect when discussing resiliency over telecommunications systems in the context of the adoption of the cloud computing paradigm, a big challenge for telecommunications service providers. Initiatives of NFV platform implementations like OpenMANO (strongly based on NFV standardization from ETSI) serve as base to run experiments like the one proposed in this master project, in which we describe our contribution to an implementation of management for high availability on top of platform proposed by Open Source MANO (OSM). The high availability implementation presented in this project is adherent to ETSI standardization and contributes to one of the most important NFV frameworks that up to now has not presented support to such feature. In this project, we present an experimentation over OSM implementation which involves the whole environment with its technological complexity. In concrete terms, our contribution addresses two main aspects: a proposal for information modeling based on ETSI standardization on high availability and its implementation in the OSM framework. Such implementation consists in the detection of failure by the orchestrator implying in a network reconfiguration as a mechanism to support redundancy in an active-active scenario. The experiment was tested and evaluated in a real minimalist environment to demonstrate the implementation performed to manage high availability information and re-orchestrate data flows between redundant VNFs.

Key-words: High Availability. Network Function Virtualization. Management And Network Orchestation.

Lista de ilustrações

Lista de tabelas

Lista de abreviaturas e siglas

API	*Appplication Protocol Interface*
AWS	*Amazon Web Services*
BSS	*Business Support Systems*
CAPEX	Capital Expenses
DPI	*Deep Package Inspection*
EMS	*Element Management System*
ENUM	*Enummerated*
ETSI	*European Telecommunications Standards Institute*
EPA	*Enhanced Platform Awareness*
GS	*Group Specification*
ICT	*Information and communications technology*
IEEE	Instituto de Engenheiros Eletricistas e Eletrônicos
IP	*Internet Protocol*
ISG	*Industry Specification Group*
KVM	*Kernel-based Virtual Machine*
MANO	*Management ANd Orchestration*
MVNO	*Mobile Virtual Network Operator*
NAT	*Network Address Translator*
NetFPGA	*Net Field Programmable Gate Array*
NFV	*Network Function Virtualization*
NFVI	*NFV Infrastructure*
NSD	*Network Service Descriptor*
RTP	*Real-time Transport Protocol*

SDN	*Software-Defined Networks*
SLA	*Service Level Agreement*
TCP	*Transmission Control Protocol*
VDU	*Virtualisation Deployment Unit*
VIM	*Virtualised Infrastructure Manager*
VLC	*VideoLAN Client/Server*
VLD	*Virtual Link Descriptor*
VM	*Virtual Machine*
VNF	*Virtualised Network Function*
VNFC	*Virtualised Network Function Component*
VNFCI	*VNFC Instance*
VNFD	*Virtualised Network Function Descriptor*
VNFI	*VNF Instance*
VNFM	*VNF Manager*
ONOS	*Open Source Network Operating System*
OPEX	*Operational Expenses*
OPNFV	*Open Platform for NFV*
OSM	*Open Source MANO*
OSS	*Operations Support Systems*
OTT	*Over The Top*
OVN	*Open Virtual Network*
OVS	*Open Virtual Switch*
QoE	*Quality of Experience*
REST	*Representational State Transfer*
SBRC	Simpósio Brasiliero de Redes de Computadores
TI	Tecnologia da Informação

UDP	*User Datagram Protocol*
Vl-Ha	*Virtualisation Layer - Hardware Resources (Reference Point)*
Vn-Nf	*VNF - NFV Infrastructure (Reference Point)*
Or-Vnfm	*NFV Orchestrator - VNF Manager (Reference Point)*
Vi-Vnfm	*Virtualised Infrastructure Manager - VNF Manager (Reference Point)*
Or-Vi	*NFV Orchestrator - Virtualised Infrastructure Manager (Reference Point)*
Nf-Vi	*NFVI - Virtualised Infrastructure Manager (Reference Point)*
Os-Ma	*OSS/BSS - NFV Management and Orchestration (Reference Point)*
Ve-Vnfm	*VNF/EM - VNF Manager (Reference Point)*

Sumário

1 Introdução

Nos últimos anos, temos observado a rápida introdução de tecnologias para a implementação do modelo de processamento em nuvem com o desenvolvimento de plataformas bem definidas de *software* para o mercado de Tecnologia da Informação (TI). O objetivo da computação em nuvem é oferecer uma plataforma em que serviços são disponibilizados de maneira facilmente acessível, com garantias de disponibilidade e escalabilidade a custos operacionais e de investimentos reduzidos. Ao mesmo tempo, há a convergência de TI e telecomunicações, em que as operadoras tradicionais de telecomunicações têm procurado ganhos de eficiência operacional e inovação em serviços, tirando proveito do conceito de computação em nuvem.

Um bom exemplo de esforço das operadoras em adoção de novos modelos de negócios para se manterem competitivas tentando ampliar o portfólio de serviços é o oferecimento do modelo MVNO (*Mobile Virtual Network Operator*), ou seja, hospedagem de parte ou em totalidade de uma operadora virtual. Como descrito por Yrjo e Rushil (2011), o modelo MVNO tem o maior potencial de se beneficiar do paradigma de computação em nuvem por suas características de escalabilidade e isolamento de serviços.

Sabe-se também que os serviços de telecomunicações tradicionais impõem requisitos muito mais elevados de qualidade e confiabilidade quando comparados aos serviços de TI. Neste contexto, um dos maiores desafios encontrados atualmente pelas operadoras ao migrarem para o ambiente virtualizado está em como manter seus níveis de serviço e confiabilidade compatíveis com os atuais, garantidos por uma infraestrutura há muito tempo estabelecida, comprovadamente robusta e confiável. Nos sistemas de telecomunicações atuais, os níveis de qualidade e confiabilidade de serviços são garantidos por implementações em *hardware* e *software* que compreendem uma arquitetura especialmente desenhada para atender esses altos requisitos. Esses requisitos são categorizados como *Carrier Grade* (grau de operadora), ou seja, sistemas extremamente confiáveis, testados e provados em suas funções. Como exemplo, podemos mencionar famílias de plataformas proprietárias dos vários fornecedores de equipamentos. Tais plataformas apresentam arquitetura com várias placas de processamento e redundância múltipla de rede, especialização de *hardware* e *software* para a finalidade específica de servir como base para aplicações de telecomunicações que se tornam altamente dependentes de suas definições de implementação.

Esse tipo de plataforma tem um alto custo de projeto, produção, implantação, operação e manutenção, exigindo mão de obra especializada e *software* especificamente construído para ela, aumentando assim os custos de investimento (CAPEX - *CAPital EXpenses*) e operações (OPEX – *Operational EXpenses*).

Podemos dizer que o paradoxo observado no passado quando da introdução dos serviços de dados (anos 1990) com relação aos serviços tradicionais de telecomunicações (por exemplo, voz e mensagens curtas) quanto a qualidade e confiabilidade, encontra analogia com a mudança atual de paradigma com a introdução da computação em nuvem para serviços de telecomunicações. Há, portanto, um amplo espaço para estudo desses aspectos de qualidade e confiabilidade e da forma como eles podem fomentar a inovação com novas propostas de arquitetura e tecnologias para o alcance dos mesmos níveis atualmente observados. Especificamente, a Alta Disponibilidade é um aspecto-chave quando analisamos qualidade e confiabilidade de serviços. Disponibilidade é entendida como "a razão do tempo total em que uma unidade funcional é capaz de ser utilizada durante um dado intervalo" (GLOSSARY, 2007). Alta disponibilidade pode ser definida quando essa razão atinge valores muito próximos a 100%. O valor mais aceito de maneira formal pela indústria de telecomunicações é 99,999%, conhecido também como os cinco noves da alta disponibilidade, o que equivale a 5 minutos de indisponibilidade por ano (QIU et al., 2011). Juntamente com desempenho, a disponibilidade é o aspecto que mais separa o avanço da computação em nuvem da aplicabilidade real nas operadoras de telecomunicações devido à distância existente entre as arquiteturas especializadas atualmente implementadas e as genéricas propostas pela computação em nuvem e virtualização.

Para o entendimento de como a alta disponibilidade impacta a implementação de sistemas de telecomunicações em nuvem, é importante estudar os conceitos básicos de alta disponibilidade como descrito em Qiu et al. (2011) e os desafios de confiabilidade em modelos de computação em nuvem (TRIVEDE et al., 2006), onde as potenciais causas de falha são identificadas: falha de *hardware*, falha de *software* e falha de operação. Um framework de virtualização de funções de rede de telecomunicações que se propõe a superar os desafios de confiabilidade de serviços deve ser capaz de prover mecanismos de recuperação para essas causas de falha. Para que isso ocorra, a aplicação necessita implementar mecanismos de replicação e a infraestrutura de computação em nuvem precisa prover mecanismos de gerenciamento das funções de rede e reorganização da orquestração de serviço (inclusive com o uso de *Software Defined Networking*) para suportar a replicação, benefício inerente à computação em nuvem.

Neste contexto, um aspecto especificamente importante para a alta disponibilidade é como a replicação da aplicação deve se comportar e quais características devem ser levadas em consideração quando uma determinada unidade de informação é tratada pela função de rede virtualizada. Mecanismos de replicação de uma determinada função virtualizada de rede devem minimizar o tempo de recuperação da informação em processamento e resgatar o estado de uma determinada execução.

Sendo assim, em termos gerais, o suporte a alta disponibilidade em sistemas executados em nuvem dependem da implementação em três níveis: suporte oferecido pela

infraestrutura de virtualização, suporte oferecido pela aplicação e gerenciamento SDN da infraestrutura de rede. O gerenciamento de rede SDN inserido no contexto da aplicação MANO (Gerenciamento e Orquestração - *Management ANd Orchestration*) aparece como um desafio à adoção do paradigma NFV como descrito por Mijumbi et al. (2016). O aspecto mais relevante para este estudo está na existência de uma dependência entre o suporte da plataforma de virtualização e o gerenciamento de rede SDN, pois o controle de execução das funções virtualizadas replicadas e a conectividade de rede para controle e oferecimento de serviço precisam ser orquestrados.

Reconhecendo a importância que a padronização tem na indústria de telecomunicações, definimos uma proposta de implementação que consiste principalmente no respeito aos princípios estabelecidos pela ETSI nos padrões definidos pelo ISG NFV (ETSI ISG NFV, 2012). Portanto, o objetivo deste projeto é demonstrar como uma implementação de gerenciamento da alta disponibilidade em um dos frameworks NFV é aderente à padronização ETSI e pode contribuir para a evolução das plataformas de NFV. Para tanto, é bastante apropriado que busquemos aplicar tal implementação sobre uma plataforma que siga a padronização, o que nos levou a utilizar a plataforma de código aberto OSM (Open Source MANO, 2016). Analisamos a arquitetura e modelagem de informação do OSM (OpenMANO e OpenVIM) para propor uma modelagem de informação de suporte à alta disponibilidade que comportasse um mecanismo de redundância que possibilitasse a reconfiguração de rede de acesso ao serviço VNF.

Portanto, nosso trabalho tem como contribuição primária o estudo e uma proposta de implementação de alta disponibilidade para serviços NFV segundo a especificação ETSI, contribuindo para a verificação dessa especificação em uma implementação prática.

Uma segunda contribuição importante é através da aplicação do modelo e implementação em código aberto na plataforma OSM, adicionando função a plataforma e contribuindo com a comunidade de desenvolvimento.

Além de buscarmos contribuir com o projeto OSM, temos a oportunidade de pesquisar como o paradigma de computação em nuvem pode ser impactado por implementações que visam cobrir requisitos que residiam em outro contexto, exercitando assim o conceito de *cloud computing* através do estudo e experimentação sobre a implementação proposta pelo OSM que envolve todo o ambiente com seus *compute nodes*, *OpenFlow switches* e *softwares* de virtualização e controle. Com a implementação do algoritmo para suporte à alta disponibilidade com foco na reconfiguração da rede envolvendo o relacionamento NFV-SDN temos uma terceira contribuição direta deste projeto.

Como já comentado, é importante ressaltar que os aspectos de alta disponibilidade dependem de mecanismos presentes na infraestrutura virtualizada e na aplicação. No estudo apresentado aqui, analisaremos somente a dimensão da infraestrutura respeitando os papéis dos componentes orquestração e gerenciamento de infraestrutura descritos na

documentação da padronização no que tange ao gerenciamento da informação de alta disponibilidade, sua implicação no gerenciamento da conectividade (redirecionamento de fluxos em uma rede SDN), e na sua implementação sobre o OSM. Os impactos impostos à aplicação não serão considerados neste trabalho e serão desenvolvidos em trabalhos futuros.

Este documento está organizado da seguinte forma: a Seção 2 apresenta os trabalhos relacionados; a Seção 3 descreve os conceitos fundamentais sobre NFV, principais aspectos da padronização ETSI relevantes para a alta disponibilidade e uma descrição da plataforma OSM; a Seção 4 foca os pontos da implementação sobre o código aberto OSM relevantes a este trabalho e apresenta a proposta de modificações e inserções feitas na plataforma; a Seção 5 descreve a avaliação dos resultados do experimento. E, finalmente, a Seção 6 conclui a dissertação.

2 Trabalhos relacionados

Algumas iniciativas para mapear casos específicos de redundância e disponibilidade de sistemas NFV podem ser encontradas no trabalho de Taleb, Ksentini e Sericola (2016), no qual um framework é proposto para garantir alta disponibilidade em sistemas 5G virtualizados. Tal proposta estabelece uma interação em que os componentes virtualizados do sistema 5G sejam redundantes e mecanismos de replicação com marcação de pacotes garantam que requisições do serviço não sejam perdidas ou erroneamente tratadas. Entretanto, essa proposta não leva em consideração a padronização NFV ISG definida pelo ETSI apresentada na Figura 2, na qual nos baseamos para propor uma solução padronizada.

Os aspectos de *Carrier Grade* são mapeados de forma a dirigir a indústria no caminho do desenvolvimento de tecnologias para atender aos requisitos de qualidade e confiabilidade comparáveis aos atualmente implementados nos sistemas dedicados das operadoras. Com foco no objeto deste trabalho, existe um projeto específico no OPNFV (*Open Platform for NFV*) (OPNFV, 2016b) para estudar e propor soluções para cenários de alta disponibilidade. A Open Platform for NFV (OPNFV) é um projeto de código aberto que visa facilitar o desenvolvimento e a evolução de componentes NFV em vários ecossistemas de código aberto. Através da integração, implantação e teste em nível de sistema, o OPNFV cria uma plataforma NFV de referência para acelerar a transformação de redes corporativas e de provedores de serviços. A plataforma de referência é composta por um conjunto de artefatos de *software* e uma metodologia. Ao integrar componentes de vários projetos, a comunidade é capaz de conduzir o desenvolvimento de *use cases* baseados em casos em várias soluções para garantir a adequação da plataforma aos casos de uso de NFV. O OPNFV trabalha com outras comunidades de código aberto para trazer contribuições e aprendizados de seu trabalho diretamente para essas comunidades como OpenDaylight, ONOS, OpenContrail, OVN, OpenStack, Kubernetes, Ceph, KVM, Open vSwitch e Linux. Portanto, o projeto foca sua contribuição construindo referências para *NFV Infrastructure* (NFVI) e *Virtualized Infrastructure Management* (VIM). A proposta do projeto concentra-se nos requisitos de alta disponibilidade da plataforma OPNFV sobre os cenários NFV de recuperação de falhas. Esses requisitos são baseados em casos reais mapeados de uma lista de requisitos elaborados pela China Mobile (OPNFV, 2016a). Podemos considerar que os estudos desenvolvidos nesse projeto nos dão subsídios para o entendimento dos casos de uso relevantes a serem considerados quando analisamos o relacionamento entre os diferentes níveis de implementação da alta disponibilidade. O que diferencia a nossa proposta do OPNFV é nossa preocupação em estudar os impactos referentes à camada MANO da arquitetura ETSI de referência.

Monteleone e Paglierani (2013) também descrevem as possibilidades de complemen-

tação entre os conceitos NFV e SDN em uma experimentação realizada na operadora Italtel em que *Session Border Controllers* (SBCs) virtualizados interagem com roteadores de *software* para controlar redes SDN para otimização e aplicação de políticas de roteamento. O trabalho citado demonstra um caso em que o gerenciamento de rede SDN é influenciado pelo controle de sessão de borda, diferentemente de nossa proposta de gerenciamento da alta disponibilidade que não está associada à natureza do VNF em si.

Em Soenen et al. (2017) são apresentados dois modelos de replicação e mecanismos possíveis de serem utilizados por um MANO para implementação de alta disponibilidade e recuperação de falhas com uso de VNFs baseadas em microserviços com alguma experimentação prática. Apesar de o trabalho usar como referência a padronização ETSI, os autores não determinaram a aderência aos padrões e tampouco definiram o modelo de informação por não utilizarem uma plataforma já disponível no mercado. No trabalho aqui proposto, apresentamos um modelo de informação no âmbito da implementação OpenMANO que mapeie os requisitos de padronização e operacionais do OpenMANO.

Um aspecto abordado neste trabalho é o relacionado à implementação de abstração de serviço pela plataforma OpenMANO. É similar ao descrito por Makaya et al. (2015) onde os autores propõem um *framework* baseado em políticas em que existe um nível de abstração para os serviços modelado em um esquema similar ao encontrado no NSD e VNFD do OpenMANO. Apesar de o trabalho relacionar a alta disponibilidade como requisito para a implementação de um sistema MANO típico, os autores não detalharam como esse requisito é atendido pelo modelo e plataforma, não especificando quais funções de gerenciamento de VMs do OpenStack poderiam ser utilizadas para uma implementação de replicação e *failover*. Em nosso trabalho, propomos um modelo regido pela especificação (padronização) mapeado sobre o modelo de informação VNFD e implementado no modelo de dados (tabela em banco de dados) do OpenMANO.

Após as pesquisas por plataformas para implementação da experimentação de alta disponibilidade em NFV, onde optamos pelo OSM, uma nova iniciativa de desenvolvimento de plataforma foi lançada em 2016, o OpenBaton. Segundo seus idealizadores "o objetivo principal é o desenvolvimento de uma estrutura extensível e personalizável capaz de orquestrar serviços de rede em infraestruturas heterogêneas NFV"(OPENBATON, 2016). Sendo assim, a intenção não é implementar o componente VIM da arquitetura de referência. Assim como o OpenMANO, o OpenBaton tem a proposta de mapear a padronização ETSI de maneira fiel, sendo, portanto, uma boa candidata como plataforma para implementação do nosso projeto. Assim como o OSM, a plataforma não apresenta suporte à alta disponibilidade. A diferença para o OSM reside na não implementação do VIM, utilizando assim o OpenStack como gerenciador de infraestrutura. Para a completude de nossa implementação é conveniente que tenhamos a correspondente implementação no VIM. Sendo o OpenVIM parte do projeto OSM, seu código aberto está disponível

para propormos as alterações necessárias para a implementação do gerenciamento da alta disponibilidade.

Existem ainda trabalhos relacionados à utlização do paradigma SDN para prover a flexibilidade necessária para que o gerenciamento do suporte à alta disponibilidade ocorra. Um desse trabalhos é o apresentado por Kim, Koo e Paik (2015), no qual os autores apresentam um estudo das métricas propostas pelos organismos de padronização e propõem métricas para avaliação de confiabilidade de de sistemas NFV e SDN do ponto de vista das operadoras. Contemplamos em nosso trabalho a avaliação da performance de chaveamento de fluxos para de um acesso de serviço incorreto (com falha) para um acesso ao serviço redundante. Enquanto nosso trabalho foca no aspecto do gerenciamento SDN no caso de *failover*, o trabalho de Kim, Koo e Paik (2015) abrange um amplo espectro de métricas além do que analisa a padronização IETF (diferentemente da ETSI, no nosso caso).

Rossem et al. (2015) apresenta um trabalho que aborda a elasticidade em ambientes em nuvem, focando no impacto que o gerenciamento da elasticidade tem na orquestração de rede SDN. A elasticidade pode ser outra forma de manutenção da disponibilidade. O artigo propõe uma solução no contexto do SAF (*Service Availability Forum*) porém o artigo não trabalha em caso de uso buscando a alta disponibilidade, focando em solução de clusterização. De maneira similar, temos a nosso trabalho a análise do impacto da replicação para alcançar a alta disponibilidade considerando a rede SDN. Porém, nosso trabalho busca levar em consideração o que a aplicação necessita de suporte em termos de replicação.

Tabela 1 – Resumo comparativo entre os trabalhos relacionados e este trabalho.

Característica	**Trabalho Relacionado**	**Este trabalho**
SDN+NFV	Rossem et al. (2015): diponibilidade usando SAF (*Service Availability Forum*) e elasticidade.	Alta disponibilidade e ETSI NFV.
SDN+NFV	Kim, Koo e Paik (2015): avaliação de confiabilidade de de sistemas NFV e SDN.	Avaliação de caso de implementação de Alta disponibilidade.
Experimentação de Alta Disponibilidade em NFV	Taleb, Ksentini e Sericola (2016): aplicação de alta disponibilidade em aplicações redundantes 5G não aderente a ETSI NFV.	Aderente a padronização ETSI NFV.
Plataforma MANO	(OPNFV, 2016b): define casos de uso no âmbito do VIM e aderente ao ETSI NFV.	Caso de uso MANO e VIM.
SDN+NFV	Monteleone e Paglierani (2013): influência da aplicação SBC na organização da rede SDN.	Influência da alta disponibilidade na rede SDN.
Plataforma MANO	Soenen et al. (2017): modelos de replicação em NFV e microserviços.	Modelo de replicação com modelo de informação.
Modelo de informação MANO	Makaya et al. (2015): modelo baseado em políticas que definem alta disponibilidade não aderente a ETSI NFV.	Modelo aderente ao TESI NFV sem políticas.
Plataforma *open source* MANO	(OPENBATON, 2016): plataforma MANO sem camada VIM.	Utiliza plataforma OpenMANO e OpenVIM.

3 Conceitos Fundamentais

Nesta seção, listaremos os conceitos fundamentais utilizados na pesquisa e elaboração da proposta de solução de gerenciamento da alta disponibilidade.

3.1 Alta Disponibilidade

Segundo Avizienis et al. (2004), disponibilidade significa prontidão para o serviço correto. O serviço é correto quando entrega a implementação da função do sistema. Uma falha de serviço, muitas vezes abreviada como simplesmente falha, é um evento que ocorre quando a entrega do serviço se desvia do serviço correto.

A definição de disponibilidade é derivada do conceito de confiabilidade, que é a capacidade de prestar serviço que possa ser justificadamente confiável. Esta definição salienta a necessidade de justificação da confiança. Uma definição alternativa, que fornece o critério para decidir se o serviço é confiável, é a capacidade do sistema de evitar falhas que são mais frequentes e mais graves do que é aceitável.

Já a Alta Disponibilidade é um conceito de mercado que coloca requisitos mensuráveis ao conceito de disponibilidade. Conhecido como os cinco noves da alta disponibilidade, o que equivale a 5 minutos de indisponibilidade por ano (QIU et al., 2011). Em termos formais, significa que a disponibilidade de 99,999% por um determinado período deve ser observada em sistemas considerados de alta disponibilidade.

Mesmo que um sistema seja projetado com dispositivos de *hardware* e *software* altamente confiáveis, ainda não se torna um sistema altamente disponível se tiver muitos pontos únicos de falha, o que requer intervenção humana para diagnóstico e reparo, que usualmente necessita algumas horas de manutenção, impossibilitando assim, que o requisito de alta disponibilidade seja atingido.

Portanto, é necessário que um mecanismo de redundância seja implementado. Redundância significa que uma aplicação extra está operando e esperando para assumir em caso de falha da aplicação que está em funcionamento. Existem vários modelos de redundância, e.g. 2N, N+1, etc., porém, para o trabalho aqui proposto, o aspecto importante a ser analisado é a capacidade que a aplicação extra (replicada) tem de assumir rapidamente as operações de uma aplicação que está em funcionamento (ativo). Desse ponto de vista, temos as configurações ativo-ativo e ativo-passivo.

As configurações de redundância ativo-ativo são configurações nas quais todas as entidades estão executando ativamente as funções do aplicativo e, portanto, são consideradas ativas em relação ao processamento do aplicativo. Já as configurações de redundância

ativo-passivo são aquelas nas quais as entidades ativas estão executando o aplicativo e fornecendo serviço específico e esses elementos ativos são protegidos por uma ou mais entidades em espera, que não estão fornecendo serviço ativamente.

Mecanismos de replicação de uma determinada aplicação devem minimizar o tempo de recuperação da informação em processamento e resgatar o estado de uma determinada execução de aplicativo. Conceitualmente, podemos definir dois tipos de aplicações do ponto de vista de replicação: aplicações dependentes de estado (*statefull*) e não dependentes (*stateless*). Aplicações *statefull* dependerão de mecanismos sofisticados de sincronismo de informação em tempo real para manter os mesmos estados de execução em instâncias replicadas de aplicações. Enquanto isso, aplicações *stateless* não necessitam de sincronismo por terem natureza unívoca em suas funções (cada operação é única e não depende de estado ou sequência de execução. Para o nosso trabalho, essa característica será relevada uma vez que a proposta não inclui o desenvolvimento de replicação da aplicação, focando a orquestração da replicação no nível da VNFC.

3.2 Network Function Virtualization

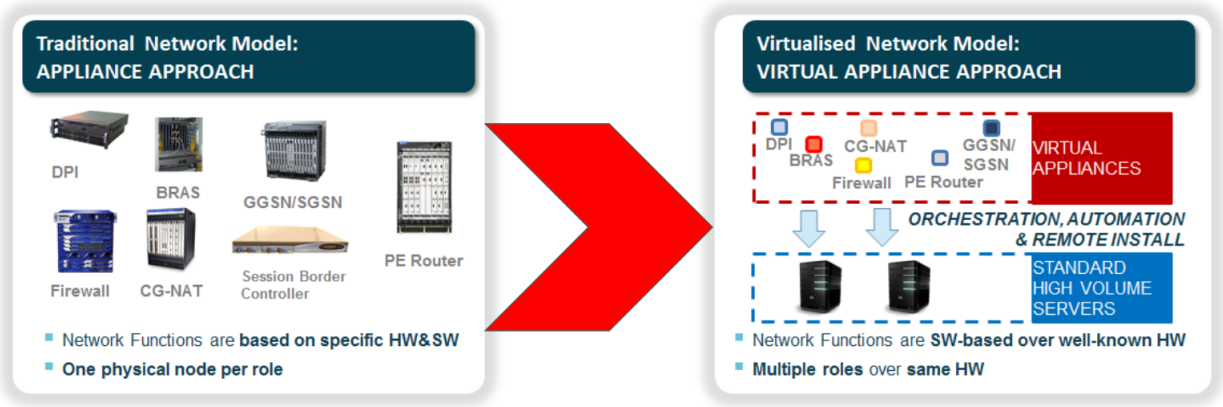
Como mencionado na Seção 1, nos sistemas de telecomunicações atuais, os níveis de qualidade e confiabilidade de serviços são garantidos por implementações em *hardware* e *software* que compreendem uma arquitetura especialmente desenhada para atender esses altos requisitos.

Na representação da Figura 1, observamos que a indústria de telecomunicações precisa promover a mudança de paradigma de sistemas dedicados (lado esquerdo, e.g. DPI, SBC, etc) para sistemas virtualizados (lado direito), de forma a tirar proveito dos ganhos econômicos promovidos pelo paradigma de computação em nuvem. Esse é portanto, o motivador para adoção do conceito NFV.

NFV (*Network Function Virtualization*) é o conceito criado pela indústria de telecomunicações e ICT para tentar mapear os problemas com a grande variedade de sistemas de propósito único em *hardware* e *software* presentes atualmente nas operadoras e que convergirão para o modelo de computação em nuvem (ETSI, 2012a). Tal migração ou criação de serviços deve ser feita em uma plataforma genérica seguindo padronizações que possibilitem a integração de aplicações e rede, além do gerenciamento e encadeamento de serviço (orquestração e encadeamento de serviço).

O NFV representa para as operadoras de telecomunicações a maneira de continuarem relevantes no mercado, promovendo redução de custos de investimento e operação, enfrentando a concorrência das OTT (*Over The Tops*) que oferecem serviços de Internet sobre as redes dessas operadoras tomando assim parte da sua receita. NFV também representa maior capacidade de inovação e agilidade na introdução de novos serviços se

Figura 1 – Mudança de paradigma para adoção de funções virtualizadas de rede.



Fonte: Lopez (2014).

valendo de características como código aberto, escalabilidade e migração de serviços.

NFV define como VNF (*Virtual Network Function*) a completa função de rede virtualizada executada em um conjunto de máquinas virtuais, incluindo o aspecto da alta disponibilidade inerente à aplicação.

Outros dois conceitos básicos que faremos referência neste trabalho é o de MANO (*Management ANd Orchestration*) e VIM (*Virtualized Infrastructure Manager*). O MANO compreende o *framework* de arquitetura especificado pela ETSI ISG NFV que define o gerenciamento e orquestração dos serviços NFV. Ele é dividido em três componentes:

- *NFV Orchestrator* (NFVO): responsável pela orquestração de recursos do NFVI em vários VIMs e o gerenciamento do ciclo de vida dos serviços orquestrados;
- *VNF Manager* (VNFM): responsável pelo gerenciamento do ciclo de vida de instâncias das VNFs;
- *Virtualized Infrastructure Manager* (VIM): responsável pelo controle e gerenciamento dos recursos do NFVI (computação, armazenamento e rede), geralmente dentro do domínio de infraestrutura de uma operadora.

Deste ponto para frente no texto, relacionaremos o MANO ao orquestrador (NFVO) e ao gerente de VNF (VNFM) de forma única e ao VIM de maneira separada para que possamos relacionar ao OpenMANO e ao OpenVIM, respectivamente.

Uma função de rede que pode ser virtualizada e, portanto, se aproveitar dos ganhos que o conceito de NFV pode trazer é o *Evolved Packet Core* (EPC).

EPC é uma arquitetura totalmente baseada em IP que (GONZALEZ et al., 2015):

> Fornece voz e dados convergentes em redes 4G e é responsável por lidar com os fluxos de tráfego de milhões de assinantes fornecendo garantias de alta disponibilidade. O EPC consiste em um conjunto de funções que geralmente são executadas em caixas de *hardware* proprietárias e especializadas. Quando o tráfego ou a sinalização exceder certo limite, uma nova caixa de *hardware* especializada deve ser instalada.

Um vEPC (*virtual* EPC) foi avaliado por Gonzalez et al. (2015) do ponto de vista de disponibilidade de serviços.

3.3 Software-Defined Network

O desenvolvimento de SDN (*Software Defined Networking*) é em grande parte devido ao fato de que podemos tornar as redes mais flexíveis, encurtando a provisão de rede, melhorando a qualidade do serviço, reduzindo os custos operacionais e aprimorando a segurança das redes. O plano de controle é separado do plano de dados, permitindo uma inteligência centralizada (lógica) em uma plataforma baseada em *software*, que mantém uma visão global da rede e possibilita a programação direta dos controladores de roteamento. O SDN separa o sistema que toma decisões sobre onde o tráfego é enviado (ou seja, o plano de controle) dos sistemas subjacentes que encaminham o tráfego para o destino selecionado (ou seja, o plano de encaminhamento de pacotes). Devido a esse desacoplamento, um administrador de rede não precisa esperar que todas as unidades ao longo do caminho estejam configuradas de maneira ideal. Essa característica é o que torna essa tecnologia essencial para a implementação viável de controle de roteamento de dados em ambiente de computação em nuvem. Não haveria possibilidade de uma reconfiguração de rede dinâmica em tempo aceitável em casos de migração de serviços ou escalabilidade sem que houvesse possibilidade operacional de alterar o roteamento de pacotes selecionados em uma rede de acesso ao serviço de maneira igualmente dinâmica.

Como exemplo de implementações de alguns serviços NFV que tipicamente reconfiguram as características de rede, temos os *Session Border Controllers* (SBCs).

Segundo a RFC 5853 (HAUTAKORPI, 2010):

> Os SBCs geralmente ficam entre duas redes de provedores de serviços (em interconexão de provedores), ou entre uma rede de acesso e uma rede *backbone* para prestar serviços a clientes residenciais e/ou empresariais. Eles fornecem uma variedade de funções para habilitar ou aprimorar serviços multimídia (por exemplo, voz sobre IP).

Portanto, o SBC pode alterar características da rede. Como exemplo prático de uma avaliação de um SBC virtualizado vemos a proposta por Monteleone e Paglierani (2013) no experimento de virtualização onde alguns cenários de alta disponibilidade foram

testados. Neste trabalho podemos observar que a implementação do SBC garante o controle de tráfego e o isolamento de rede para as VNFs.

Tomamos, portanto, essa característica essencial como um ponto crítico para o nosso trabalho: a dependência que mecanismos de reconfiguração dinâmica de rede para que seja viável o gerenciamento de serviços NFV pela arquitetura padrão de referência ETSI NFV de forma a cumprirmos com os requisitos funcionais de alta disponibilidade.

3.4 Padronização ETSI NFV

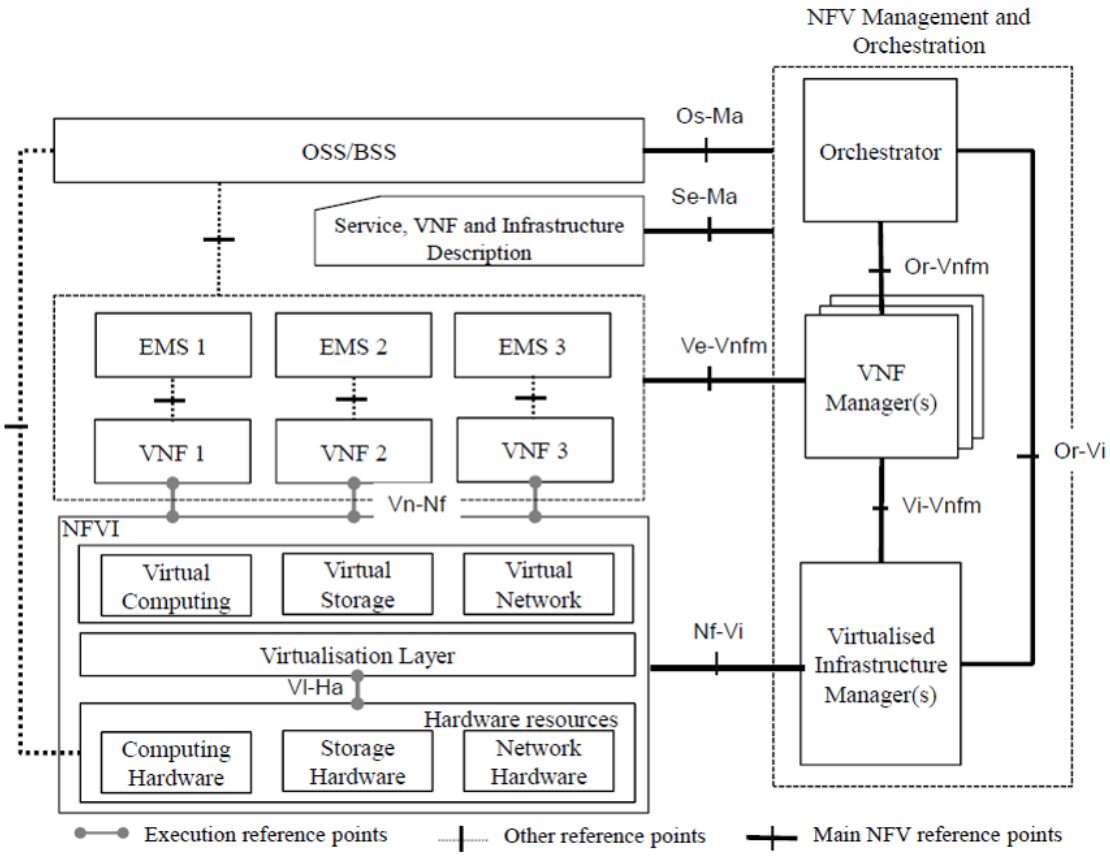
Nesta seção descreveremos brevemente a padronização ETSI NFV.

Os padrões estabelecidos pelo NFV ISG do ETSI (ETSI, 2012b) descrevem vários aspectos relevantes à visão da indústria sobre como uma implementação NFV deveria ser arquitetada e como deveria se comportar. Um aspecto importante, em termos de arquitetura, estabelecida pelo ETSI, é o *Framework* de Arquitetura de Referência que objetiva ser o resultado da transformação promovida pela virtualização da rede da operadora. Nesse *framework* estão definidos os blocos funcionais que compõem a arquitetura e os pontos de referência que definem as interfaces de relacionamento entre os blocos. Este *framework* descreve o modelo funcional e não propõe nenhuma implementação em específico (ETSI, 2018).

A documentação principal da padronização que utilizamos como referência para este trabalho é a descrição do *framework* de gerenciamento e orquestração das funções de rede (VNFs) (ETSI, 2014b). Neste documento, estão especificados os papéis de todos os componentes da arquitetura, como definidos na Figura 2. Nesse *framework*, estão definidos os blocos funcionais que compõem a arquitetura e os pontos de referência que definem as interfaces de relacionamento entre os blocos. Tal *framework* descreve o modelo funcional e não propõe nenhuma implementação em específico.

Como ilustrado na Figura 2, a arquitetura de referência define, no lado esquerdo inferior, a infraestrutura de execução NFVI (*Network Function Virtualization Infrastructure*) onde estão os componentes físicos de *hardware* e *software* básicos para suporte (armazenamento, rede e processamento) à virtualização dos serviços de rede. Sobre essa infraestrutura, as VNFs são executadas, assim como os gerenciadores de elementos EMS (Element Management System) responsáveis pelo gerenciamento de falha, configuração, contabilização, desempenho e segurança (FCAPS - *Fault, Configuration, Accounting, Performance, Security*). Ainda no lado esquerdo, temos ilustrados os componentes OSS (*Operations Support System*), que representam a camada de operação e manutenção lógica dos sistemas, e BSS (*Business Support System*) que controlam os parâmetros lógicos de negócio dos sistemas.

Figura 2 – Arquitetura de referência NFV ETSI.



Fonte: ETSI (2018).

No lado direito, está representado o gerenciamento e orquestração dos serviços virtualizados de rede (*NFV Management and Orchestration*). Esses componentes são responsáveis por organizar os serviços de rede logicamente na infraestrutura de execução, promovendo o encadeamento de serviços dependentes, gerenciando a conectividade para acesso aos serviços e garantindo que as VNFs e NFVI estejam executando corretamente. O VIM (*Virtualised Infrastructure Manager*) é responsável por alocar VMs (*Virtual Machines*) de VNFs nos nós de execução bem como gerenciar a conectividade interna e a oferta do serviço externamente. O VNF Manager controla as VNFs verificando sua execução pelas VMs e garantindo que o serviço esteja disponível coordenando com o VIM o mapeamento VNFVM. Por fim, o orquestrador (*Orchestrator*) é responsável pelo gerenciamento geral dos serviços, encadeando-os logicamente a fim de entregar uma visão fim-a-fim.

Na Figura 2, também observamos que a padronização procura especificar como os componentes trocam informações entre eles, definindo os pontos de referência (reference points) onde são descritos os modelos de informações trocados entre os componentes.

A especificação NFV ISG ETSI também define requisitos de resiliência (ETSI,

2015) com referência à alta disponibilidade.

Portanto, utilizando estas definições, buscamos contribuir para a plataforma OSM de código aberto com propostas para:

a) A modelagem de informação de alta disponibilidade de acordo com o modelo de informação das VNFs e NSD (*Network Service Description*);

b) A definição de fluxo de informação entre o VIM e o MANO para notificação de mudança de estado de uma VM pertencente a uma VNF;

c) A implementação lógica de identificação de conectividade de rede de tráfego de serviço e o redirecionamento do fluxo no switch OpenFlow via controlador.

3.5 Open Source MANO (OSM)

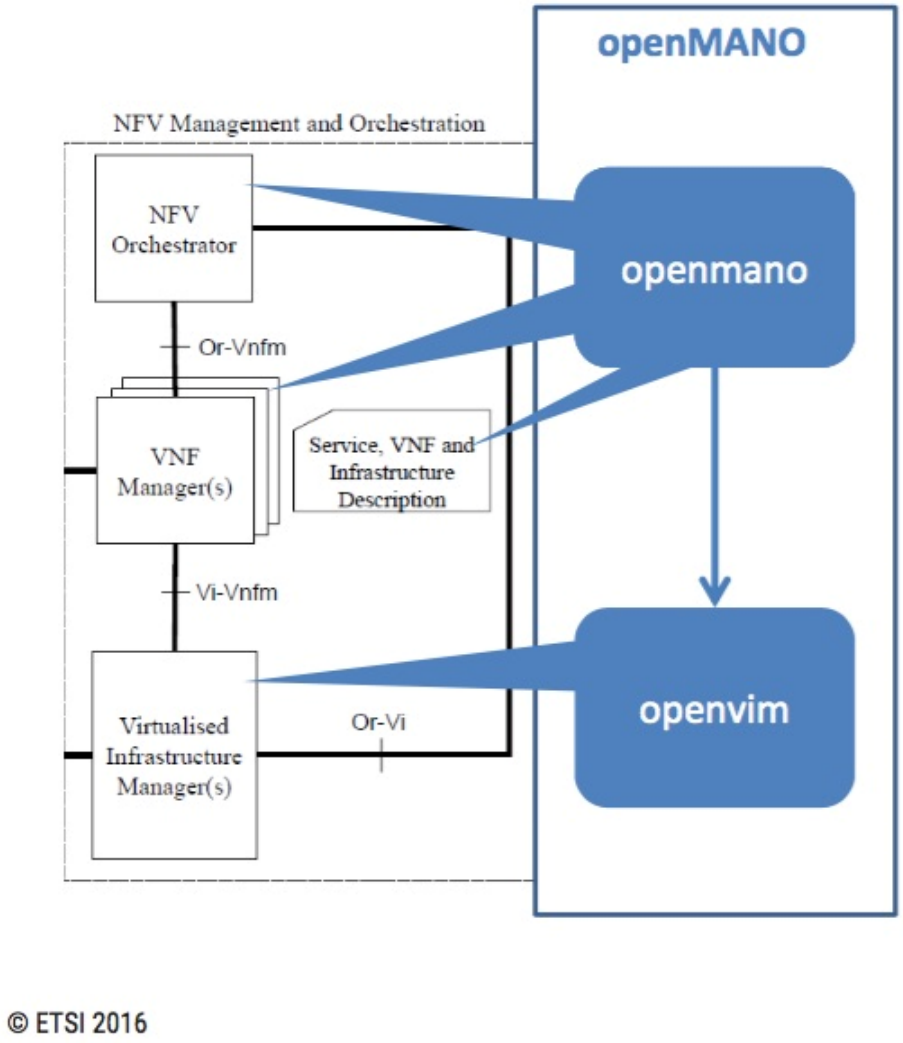
Para que possamos avaliar a aplicação dos padrões, além do mecanismo de replicação da aplicação, necessitamos utilizar uma plataforma de infraestrutura de virtualização (*Virtual Infrastructure Manager*) e um sistema de gerenciamento e orquestração, uma vez que esses dois componentes influenciam diretamente no processo de replicação e recuperação em caso de falhas.

A Telefônica é pioneira no estudo e adoção de NFV e compartilha essa visão geral com a comunidade (LOPEZ, 2014). Recentemente, a empresa lançou uma iniciativa de inovação em que o objetivo principal é implementar e testar a arquitetura de referência segundo a padronização NFV ISG do ETSI, chamada de Telefónica NFV Reference Lab ("TELEFóNICA..., 1988). A iniciativa inclui o estabelecimento de uma comunidade em que são discutidos os aspectos práticos da implementação NFV, bem como a colaboração no desenvolvimento de *software* aberto para a experimentação e evolução de uma plataforma, denominada OSM (*Open Source MANO*).

A proposta do OSM é mapear a arquitetura NFV ISG ETSI em uma implementação de plataforma de código aberto que está disponível em repositório gitweb no site do projeto OSM (OSM..., 2015). O pacote de *software* consiste em um *stack* com dois componentes básicos: o openmano e o openvim. Como podemos ver na Figura 3, a arquitetura é composta pelo gerenciamento e orquestração (OpenMANO) que compreende as funções *NFV Orchestrator* e *VNF Managers* descritas na Arquitetura de Referência (Figura 2). As informações de configuração de serviços, VNFs e infraestrutura estão armazenadas nesse componente. O OpenVIM é o gerenciador de infraestrutura virtualizada (VIM) responsável por instanciar fisicamente as VMs em servidores de execução. O *software* do OpenMANO é majoritariamente escrito em Python e roda sobre sistemas operacionais Linux tendo sido verificado sobre Ubuntu 16.04 Server. O OSM está submetido à licença Apache 2.0. Além de fornecer os componentes do gerenciamento e orquestração NFV, o OSM pretende

melhorar a experiência da administração e desenvolvimento em termos de usabilidade e instalação, assim como o modelamento de VNFs e serviços de rede. O projeto OSM também tem o objetivo de contribuir com a evolução da Arquitetura de Referência proposta pelo padrão NFV ETSI, fomentando o seu desenvolvimento e a experimentação da plataforma.

Figura 3 – Mapeamento OpenMANO/OpenVIM sobre a arquitetura de referência NFV ETSI.



Fonte: https://osm.etsi.org/ .

O orquestrador OpenMANO implementa o modelo de informação segundo a especificação ETSI NFV. A especificação define modelos que são representados como descritores, o que na prática são arquivos JSON com a descrição do conteúdo necessário para o OpenMANO ordenar a criação de uma instância de VNF para o OpenVIM. Existem duas definições relevantes para esse estudo: NSD (*Network Service Descriptor*) e VNFD (*Virtual Network Function Descriptor*). O NSD é o modelo de implementação de alto nível para um Serviço de Rede (*Network Service*). Um NSD contém os atributos para um grupo

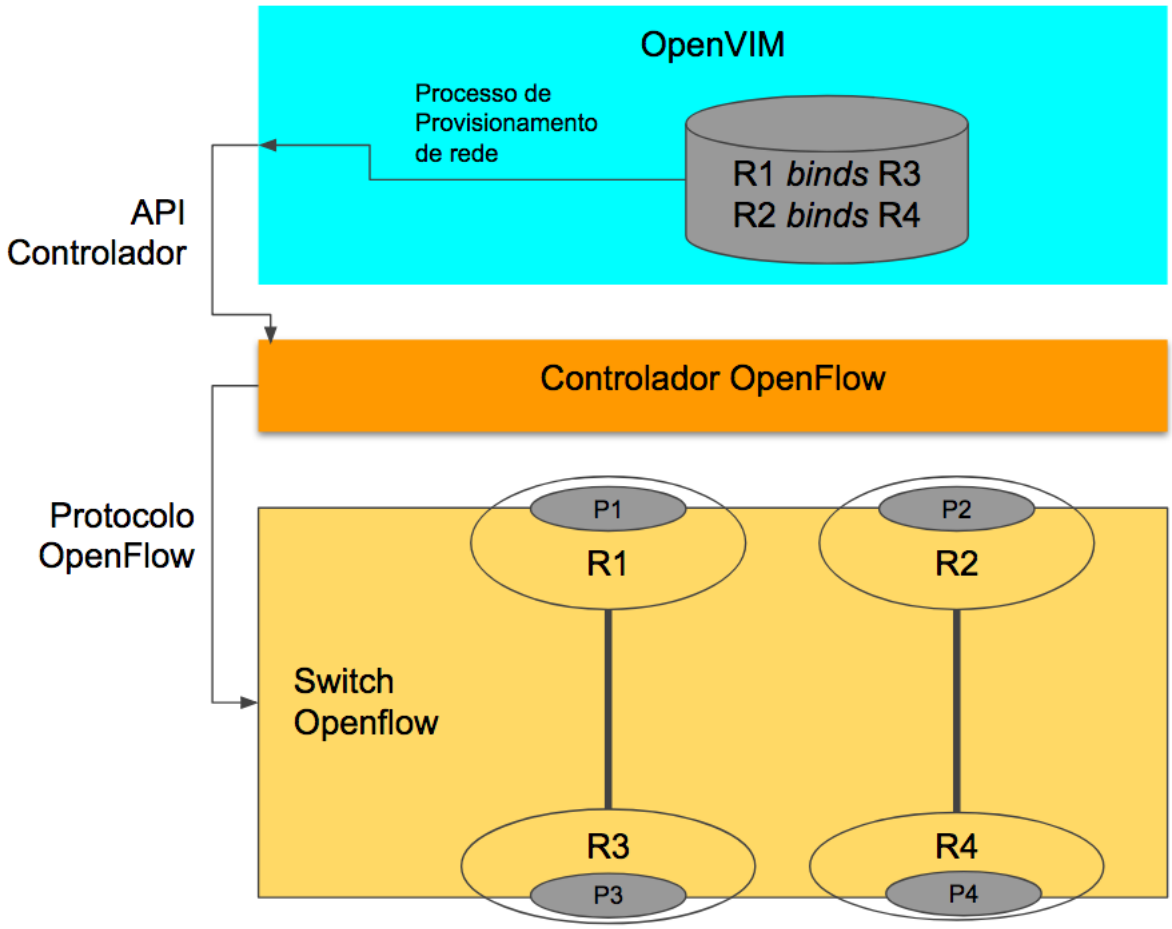
de funções de rede, que constituem uma definição de serviço. Esses atributos contêm os requisitos de relacionamento das VNFs encadeados como um serviço. A NSD faz referência a uma ou mais VNFDs, bem como a outros descritores que são usados para projetar as cadeias de serviço. Se considerarmos um serviço como um conjunto de VNFs e essas têm a mesma função, podemos propor que a redundância seja implementada como um serviço específico. Já a VNFD é um modelo que contém os atributos de uma VNF que define os requisitos de recursos da plataforma, como CPU, memória, interfaces e rede. O VNFM usa um VNFD durante o processo de instanciação e durante o gerenciamento do ciclo de vida de uma instância da VNF. As informações fornecidas no VNFD também são usadas pelo OpenMANO para gerenciar e orquestrar serviços de rede e recursos virtualizados no NFVI (NFV *Infrastructure*). O VNFD também contém requisitos de conectividade, interface de rede que podem ser usadas para estabelecer links virtuais apropriados dentro do NFVI entre suas instâncias do *Virtual Network Function Component* (VNFC) ou entre uma instância da VNF e a interface do nó para as outras funções de rede. Essas VNFs são conectados por links virtuais (VLDs). Além disso, o NSD expõe um conjunto de pontos de conexão para permitir a conectividade a outros serviços de rede ou ao mundo externo. Na implementação OpenMANO, o VNFD contém a definição de VDU (*Virtual Deployment Unit*). Uma VDU é um elemento que pode ser usado como um modelo de informação. Ele descreve a implementação e o comportamento operacional de um subconjunto de uma VNF, ou a VNF inteira, como a especificação da máquina virtual, os recursos de computação, requisitos de armazenamento, scripts de inicialização e finalização, modelo de redundância de alta disponibilidade e escalabilidade. Uma VDU é implementada como VM no VNF. Em RIFT.io (2017), o modelo de informação da VDU atualmente implementado no OpenMANO é especificado. Este modelo é utilizado para definir quais os dados da VM serão necessários para a ativação do serviço NFV. Como podemos observar, nesse modelo não temos a definição de informação sobre a alta disponibilidade como seria previsto na especificação ETSI ETSI (2014c).

Outro aspecto importante no gerenciamento das VNFs é a troca de informações sobre o estado das VMs do VIM para o MANO (ponto de referência *Vi-Vnfm*). Deve ser possível ao VIM notificar ao MANO mudanças de estado das VMs que compõem uma dada função VNF. A padronização NFV ETSI não especifica os detalhes do conteúdo desse ponto de referência, mas define que deve haver troca de informações para as quais o VNFM se inscreveu (por exemplo, eventos, resultados de medição e registros de uso de recursos NFVI usados por um VNF). Assumimos aqui que uma mudança de estado de uma VM referente a uma VNF pode ser um evento notificado do VIM (OpenVIM) para o MANO (OpenMANO).

Dessa forma, verificamos que a implementação do OpenVIM e OpenMANO não contempla tal evento de notificação, o que será objeto de nossa proposta.

Um terceiro aspecto necessário para que a alta disponibilidade seja possível é a possibilidade de a plataforma realizar o redirecionamento de fluxos de serviços de uma VM para outra no contexto da VNF. A arquitetura NFV concebe que esta função deva ser implementada utilizando o paradigma SDN com controladores de rede. No caso do OpenMANO, o OpenVIM implementa funções de controle de rede tendo integração com um controlador SDN (OpenFlow). O OpenVIM implementa o controle sobre as redes virtuais e externas através de abstração desse controle e ordenando ao controlador SDN (OpenFlow) que provisione fluxos no switch OpenFlow para refletir esta abstração.

Figura 4 – Exemplo de mapeamento de rede física em abstração feita pelo OpenVIM.



Na Figura 4, observamos um exemplo de como o OpenVIM mapeia as redes de forma a gerenciá-las de maneira abstrata. Em um *switch* OpenFlow em que temos quatro portas (P1, ..., P4), cada porta significa acesso a uma rede gerenciada pelo VIM via controlador SDN (OpenFlow). Considerando, por exemplo, que as portas P1 e P2 são conectadas fisicamente a interfaces de rede de um ou mais servidores de um *data center*, e que as portas P3 e P4 estejam conectadas a uma rede de serviço da operadora, o OpenVIM mapeia as redes em entidades internas de gerenciamento (R1, ..., R4) referenciando as

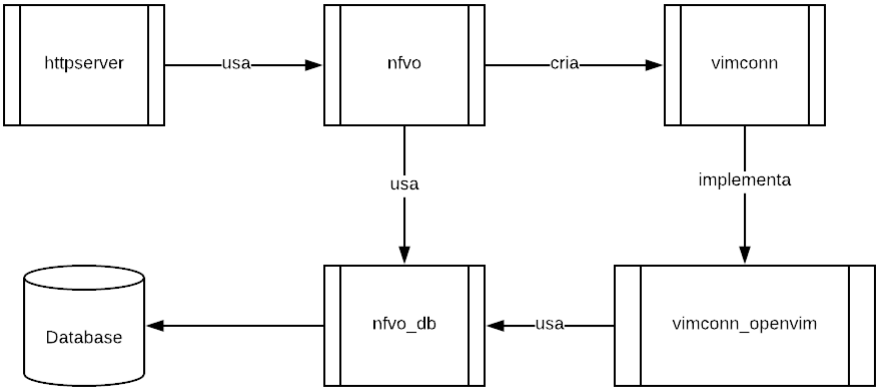
respectivas portas. Portanto, o OpenVIM cria um mecanismo de gerenciamento chamado de *bind* que relaciona as redes criando conexões lógicas na base de informação do OpenVIM e as quais implicam em uma implementação de fluxos no *switch* OpenFlow através do gerenciamento do controlador SDN. As informações de rede abstratas estão armazenadas em tabela de banco de dados no OpenVIM e um campo específico nessa tabela define o *bind* para outra rede abstrata. Quando ocorre uma mudança desse campo *bind*, o OpenVIM inicia um processo de reprovisionamento de configuração de rede que envia comandos para o controlador OpenFlow via API de modo que a rede reflita o que foi especificado na tabela. O controlador OpenFlow envia comandos de alterações de tabelas no *switch* OpenFlow (protocolo OpenFlow) para que os fluxos correspondam aos encaminhamentos definidos na rede abstrata ($R1 \leftrightarrow R3, R2 \leftrightarrow R4$).

4 Implementação do Suporte à Alta Disponibilidade sobre o OSM

Nesse capítulo, apresentamos as alterações no código aberto OSM do OpenMANO e OpenVIM para suportar alta disponibilidade no que diz respeito ao redirecionamento de fluxos relacionados ao serviço no *switch* OpenFlow. Para fins de explicação e com o propósito de sermos didáticos, adotamos um cenário em que uma VNF é composta por duas VNFCs (o que fisicamente implica em duas VMs). Antes, descreveremos brevemente na próxima seção a arquitetura do OpenMANO e OpenVIM para entendimento dos componentes afetados.

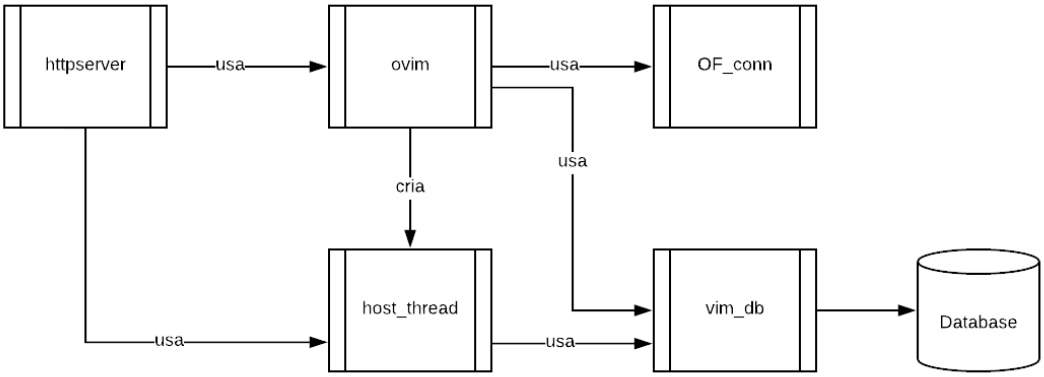
4.1 Descrição dos componentes OpenMANO e OpenVIM

Figura 5 – Diagrama de componentes do OpenMANO.



A Figura 5 apresenta os componentes do OpenMANO e o relacionamento funcional entre eles. O componente *httpserver* é o responsável por receber os comandos REST de sistemas de gerenciamento superiores e fazer chamadas de funções no *nfvo* que implementa o orquestrador em si; *httpserver* e *nfvo* são dois processos independentes iniciados pelo programa principal do OpenMANO. Para cada VIM provisionado no OpenMANO, o *nfvo* cria uma instância de *vimconn* de acordo com o tipo, no nosso caso, *vimconn_openvim* que implementa os comandos de interface REST compatíveis com o servidor REST do OpenVIM. Os dados provisionados e resultantes de execução da orquestração são armazenados em um banco de dados relacional MySQL e a abstração de acesso a esse banco é implementada por *nfvo_db*. *nfvo* e *vimconn_openvim* fazem uso de funções para armazenamento e consulta ao banco de dados.

Figura 6 – Diagrama de componentes do OpenVIM.



A Figura 6 apresenta os componentes do OpenVIM e o relacionamento entre eles. O componente *httpserver* é o responsável por receber os comandos REST de sistemas MANO e fazer chamadas de funções do componente *ovim* que implementa a lógica central do VIM. *httpserver* e *ovim* são dois processos independentes iniciados pelo programa principal do OpenVIM. Para cada *compute node* provisionado no OpenVIM, este cria um processo do componente *host_thread* que implementa o gerenciamento das máquinas virtuais no hipervisor KVM no *compute node* através da biblioteca libvirt. Os dados provisionados e resultantes de execução do gerenciamento da infraestrutura são armazenados em um banco de dados relacional MySQL e a abstração de acesso a esse banco é implementada por *vim_db*. *ovim* e *host_thread* fazem uso dessas funções para armazenamento e consulta ao banco de dados.

4.2 Fluxo básico de reconhecimento de falha e *failover*

O fluxo básico do processo que identifica a falha em uma VM e as consequentes ações tomadas pelo VIM (OpenVIM) e MANO (OpenMANO) é demonstrado na Figura 7:

1. O OpenMANO busca informações de VNFCs (VMs) através de *pooling* de intervalos de 5 minutos. Na nossa experimentação, reduzimos esse tempo para 1 segundo para que as atualizações simulassem um mecanismo de tempo real.

2. O servidor (através do *hipervisor* KVM) em que uma VM está sendo executada notifica o OpenVIM (biblioteca libvirt) de que o estado de uma VM foi alterado.

3. Numa próxima iteração do *pooling* o estado da VNFC é reportado para o OpenMANO.

4. Ao receber a informação de atualização de estado da VNFC, o OpenMANO atualiza seu banco de dados com o novo estado da VM.

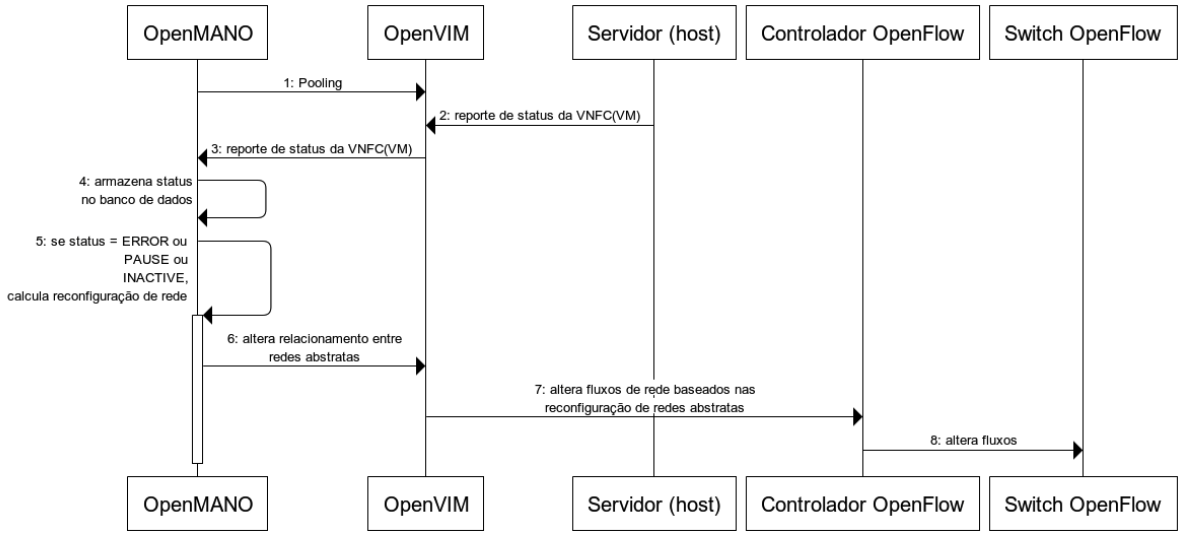
5. Caso o novo estado da VM corresponda a um estado considerado fora de operação (*ERROR, PAUSED ou INACTIVE*), o OpenMANO iniciará o processo de determinação da reconfiguração de rede (abstração do OpenVIM). Existem mais interações entre o OpenMANO e OpenVIM para obtenção de informação de organização da rede abstrata que estão omitidas nesse momento e discutidas mais à frente.

6. Após definir qual a nova configuração de rede, o OpenMANO ordena ao OpenVIM que uma nova configuração de rede abstrata seja provisionada na infraestrutura.

7. O OpenVIM identifica a nova configuração de rede no nível abstrato e envia ao controlador OpenFlow (via API) comandos para alteração dos fluxos.

8. O controlador OpenFlow envia comandos de alteração de tabelas de fluxos via protocolo OpenFlow para o *switch* OpenFlow.

Figura 7 – Fluxo básico de detecção de falha e alteração de fluxo de rede.



O OpenMANO é o responsável indiretamente pelo controle da redundância no nível de rede, tomando a decisão sobre a organização de rede abstraída pelo OpenVIM e, por consequência, o redirecionamento de fluxos no *switch* OpenFlow. O OpenMANO, ao ser notificado pelo VIM que uma VM pertencente a uma VNF falhou, requisita a reorganização dos fluxos para que uma VM redundante receba os pacotes que seriam destinados à VM que falhou.

Percebemos que na implementação atual do OpenMANO, o orquestrador realiza *pooling* de 5 minutos para monitorar o estado das VNFCs (VMs) no OpenVIM. Essa implementação compromete o desempenho final da solução e deve ser melhorada para que um evento de notificação transite do OpenVIM para o OpenMANO reportando as mudanças de estado das VNFCs. Segundo a padronização (ETSI, 2014c, seção 5.7.7), o

ponto de referência *Vi-Vnfm* deve trocar informações entre o VNFM (no caso OpenMANO, como visto na Figura 3) e o VIM (OpenVIM), que são as informações que o VNFM se registrou para receber do VIM. Apesar de esse mecanismo influenciar no tempo de recuperação da implementação proposta nesse projeto, sua relevância funcional para o que tentamos conceituar não é alta e sua implementação depende de discussões de arquitetura que não caberiam ao projeto. Esse estudo e proposta ficarão como trabalho futuro.

Outro aspecto que não está contemplado neste projeto é a implementação do provisionamento das informações de alta disponibilidade no OpenMANO. Tal implementação consiste em alterações em interfaces gráficas e processos de instanciação de entidades dentro do OpenMANO. Como focamos o modelo de informação necessária e nas consequências da falha de um serviço, nos preocupamos em entender que informações eram relevantes e como seriam utilizadas. As formas que essa informação é provisionada tem menor relevância para a prova de conceito proposta.

Nas próximas subseções, iremos descrever como:

1. O OpenMANO foi modificado para suportar o provisionamento da alta disponibilidade e como ele identifica um evento de mudança de estado de VM a partir do OpenVIM,

2. O OpenMANO altera a organização da rede no OpenVIM que por sua vez modifica os fluxos provisionados no switch Openflow direcionando-os para uma instância redundante de uma VNF.

4.3 Implementando o controle da Alta Disponibilidade no OpenMANO

Como descrito em (RIFT.IO, 2017), o modelo de informação da VDU não inclui informações de alta disponibilidade. O OpenMANO ainda não contempla funções de suporte à alta disponibilidade até o release *TWO*. Portanto, com o objetivo de implementar o redirecionamento de fluxos entre instâncias de VNFs redundantes, precisamos definir os dados no modelo de informação para que logicamente possamos identificar o caso em que a alta disponibilidade está solicitada e deve ser provisionada.

Segundo a padronização ETSI, o modelo de informação deve contemplar a informação de alta disponibilidade no contexto da VDU que é a entidade que descreve as características de todas as instâncias de VMs que compõem uma VNF. A definição para a alta disponibilidade apresenta um parâmetro no modelo de informação como descrito na Tabela 2.

Tabela 2 – Modelo de informação sobre alta disponibilidade na VDU recomendado pela ETSI.

Identification	**Type**	**Cardinality**	**Descritpion**
high_availability	Leaf	0...1	Defines redundancy model to ensure high availability examples include: • ActiveActive: Implies that two instances of the same VDU will co-exist with continuous data synchronization. • ActivePassive: Implies that two instances of the same VDU will co-exists without any data synchronization.

Fonte: (ETSI, 2014c)

A informação de alta disponibilidade pode existir ou não, tendo cardinalidade 0 ou 1, respectivamente. O modelo ETSI define duas formas de redundância: ativo-ativo e ativo-passivo. Neste trabalho, focaremos o modelo ativo-ativo pelo qual duas instâncias ativas de uma função estão presentes e tratando diferentes fluxos e, em caso de falha de uma delas, a outra pode assumir todos os fluxos de dados. No modelo ativo-passivo, uma das instâncias replicadas não processa nenhum fluxo e fica exclusivamente aguardando a falha da instância principal.

Essa informação deve ser provisionada para cada instância de VM gerenciada pelo OpenMANO. Quando o OpenMANO requisita a criação de uma VNF para o OpenVIM, as instâncias de VM representando essa VNF são refletidas na tabela *instance_vms* (Tabela 3) do banco de dados MySQL do OpenMANO. A modificação proposta nesse caso é a criação de um campo *high_availability* do tipo ENUM com valores *0='ACTIVE_ACTIVE' e 1='ACTIVE_PASSIVE'*. O campo tem valor padrão NULL, o que significa que não há redundância.

Tabela 3 – Tabela *instance_vms* MySQL no OpenMANO.

Field	**Type**	**Null**	**Key**	**Default**
uuid	varchar(36)	NO	PRI	NULL
instance_vnf_id	varchar(36)	NO	MUL	NULL
vm_id	varchar(36)	NO	MUL	NULL
vim_vm_id	varchar(36)	NO	UNI	NULL
status	enum('ACTIVE:NoMgmtIP','ACTIVE','INACTIVE', 'BUILD','ERROR','VIM_ERROR','PAUSED', 'SUSPENDED','DELETED')	NO		BUILD
error_msg	varchar(1024)	YES		NULL
vim_info	text	YES		NULL
created_at	double	NO		NULL
modified_at	double	YES		NULL
*high_availability	enum('ACTIVE_ACTIVE','ACTIVE_PASSIVE')	YES		NULL
*replicated	varchar(36)	YES		NULL
*service_interface	varchar(10)	YES		NULL

Notas: * campos criados por este projeto

Outras duas informações são necessárias para que o orquestrador possa tomar a decisão em fazer o gerenciamento da disponibilidade. A primeira é a identificação da instância replicada da VM ao qual a VM com falha está associada. Criamos, portanto, o campo *replicated* que tem a identificação única *uuid*[1] desta mesma tabela para a VM replicada que deve assumir o serviço deixado pela VM com falha. A segunda informação é a interface (*service_interface*) das VMs utilizadas para disponibilizar o serviço. Como comentado anteriormente, não nos preocuparemos aqui como estas informações foram provisionadas, assumimos que serão providas pelo agente que configura o serviço e configura a imagem da VM a ser carregada pelo VIM. Mais à frente na descrição da implementação explicaremos a utilidade dessas informações.

Abaixo, apresentamos um resumo com a breve descrição dos componentes relevantes que serão mencionados mais à frente neste projeto e que foram alterados no OpenMANO para suportar a alta disponibilidade:

a) *httpserver*: componente instanciado em forma de *thread* que interpreta os comandos da API *northbound* e importa o componente *nfvo* para tratamento das requisições de orquestração de mais alto nível;

b) *nfvo*: componente com a lógica principal da orquestração, responsável por iniciar *threads* relacionadas a diferentes tipos de VIMs. Neste projeto, o VIM adotado é o OpenVIM;

c) *nfvo_db*: componente que implementa funções de interação com o banco de dados;

d) *vimconn_openvim*: componente que implementa funções específicas de interação com OpenVIM. É derivado da classe genérica *vimconn*.

A partir da informação do modelo de redundância (*high_availability*), o orquestrador pode monitorar o VIM por eventos que ocorram com a VM. Atualmente, o OpenMANO realiza *pooling* (como pode ser visto na Figura 7) para implementar a atualização sobre as VMs monitoradas por um VIM orquestrado. Essa implementação consistiu na alteração da classe *vim_thread* para que esta pudesse chamar uma função do nfvo que tratasse o evento de atualização de estado de VMs.

Ao ser notificado pelo VIM sobre uma mudança de estado de alguma VM, o OpenMANO deve calcular, em caso de indisponibilidade da VM, como fazer a reorganização das redes abstratas para que seja efetuado o redirecionamento dos pacotes destinados à VM que entrou em estado considerado não-funcional para a VM redundante. Utilizamos o mecanismo de *callback* onde passamos a referência para a função no construtor da classe *vim_thread*. Criado o mecanismo, implementamos a função *_callback_vm_event(vim_vm_id)* no componente *nfvo* que recebe a identificação da VM que teve o estado alterado,

[1] *uuids* são gerados internamente pelo OpenMANO e OpenVIM

Figura 8 – Modelo de execução e comunicação OpenMANO e OpenVIM para o tratamento de evento de falha em VNF (parte 1).

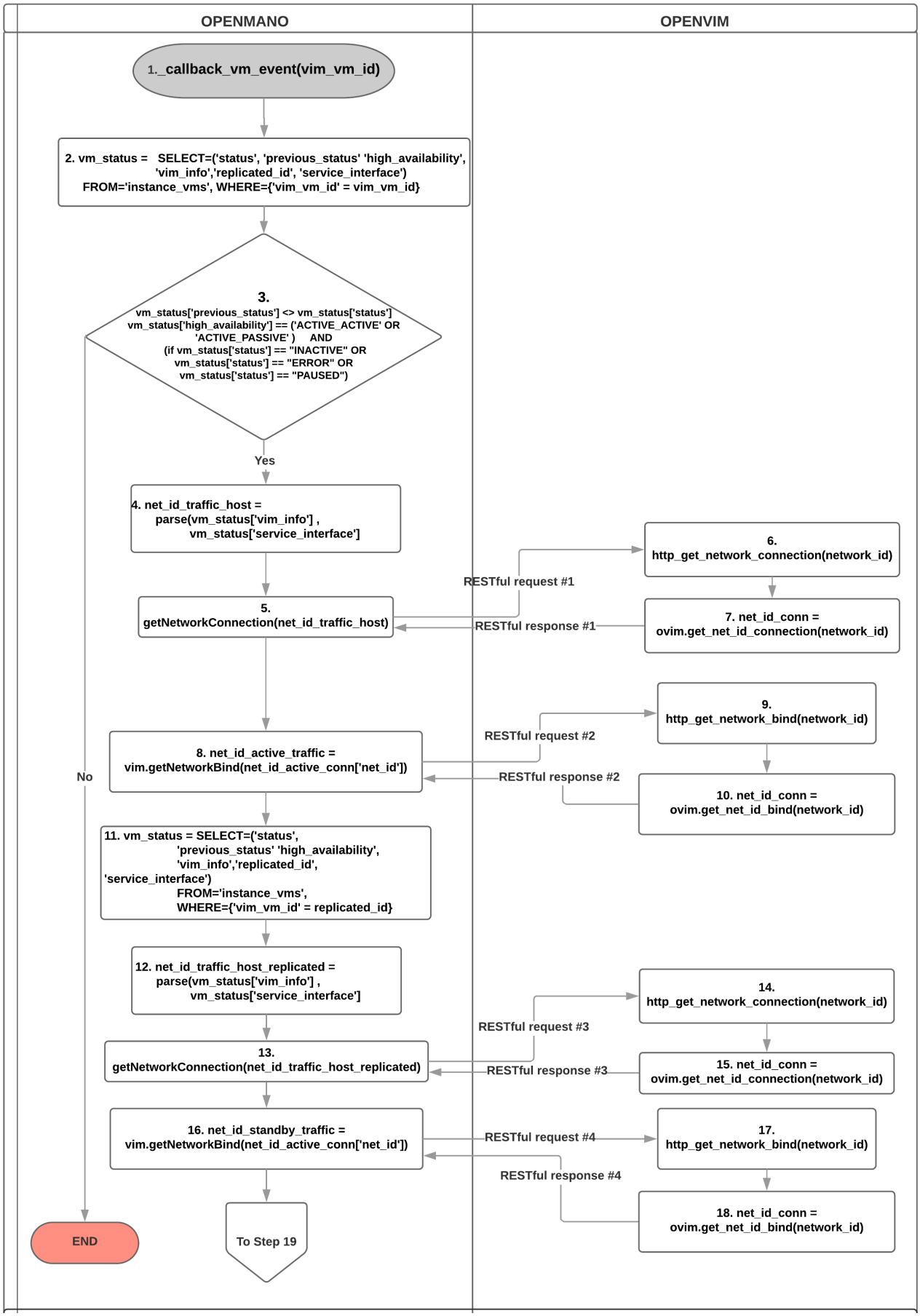
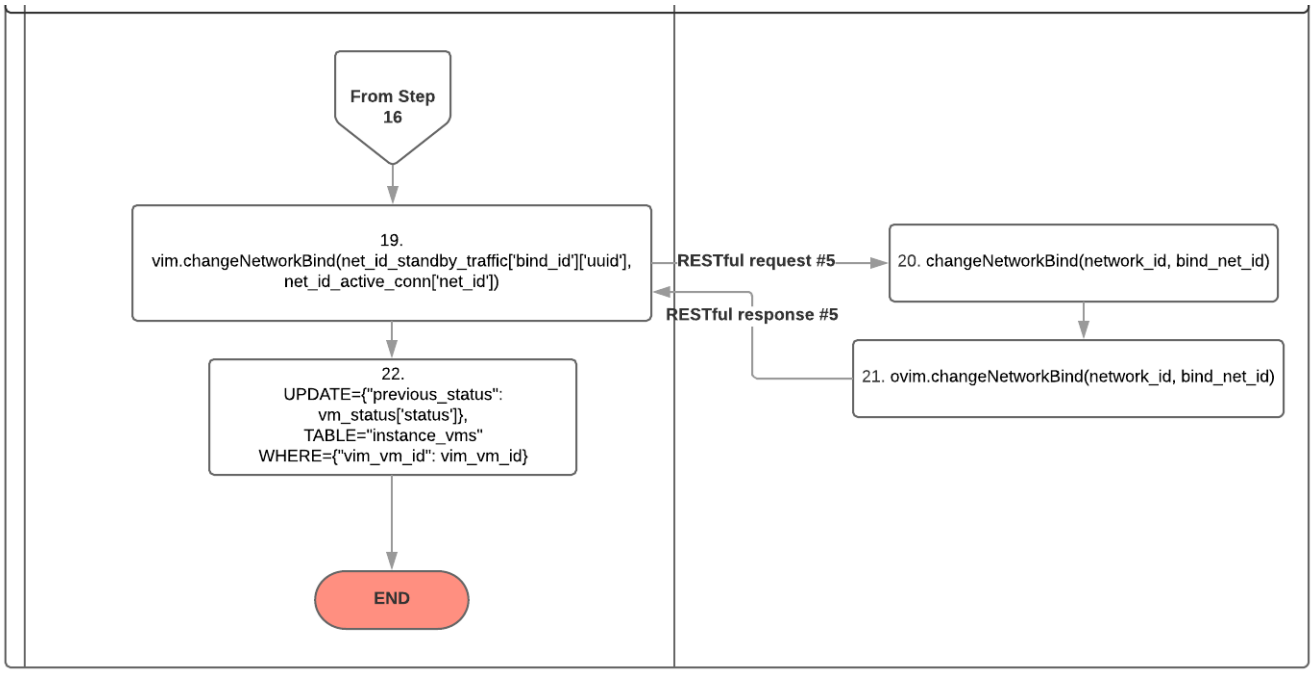


Figura 9 – Modelo de execução e comunicação OpenMANO e OpenVIM para o tratamento de evento de falha em VNF (parte 2).



onde *vim_vm_id* é a identificação *uuid* da VM em questão na tabela *instance_vms* (*tabMANODB*). Esta função identifica mudanças no valor *status* da VM. Caso o valor deste campo seja *'INACTIVE', 'ERROR' ou 'PAUSED'*, o OpenMANO considera que a instância relacionada ao VM *vim_vm_id* da VNF falhou. Os diagramas nas Figura 8 e Figura 9 detalham o fluxo de execução e interação entre o OpenMANO e OpenVIM. Baseado nesses diagramas, entendemos como a implementação no OpenMANO realiza as operações para reorganizar a conectividade de rede de tráfego de serviço desta VM (a que falhou) para a VM replicada com informações obtidas do OpenVIM:

1. *_callback_vm_event(vim_vm_id)* é chamada pelo mecanismo de *pooling*;

2. Consulta à base de dados para obter a informação de interface de serviço da VM (*service_interface*), as informações gerais da VM (*vim_info*) e a instância replicada (*replicated_id*);

3. Caso o último estado verificado para a VM seja diferente do estado reportado pelo OpenVIM (*'previous_status' <> 'status'*[2]), *'high_availability'* seja *'ACTIVE_-ACTIVE'* ou *'ACTIVE_PASSIVE'* e o campo *status* seja *'INACTIVE', 'ERROR' ou 'PAUSED'* seguirá para o passo 4. Caso contrário termina a função;

[2] A implementação presente no OpenMANO atualiza o campo *status* da tabela *instance_vms* antes de chamar a função *_callback_vm_event(vim_vm_id)*

4. A partir da identificação da interface de serviço (*service_interface*) é possível extrair pela busca no campo *vim_info* a identificação *uuid* da rede ao qual a interface está conectada;

5. Com essa identificação é possivel enviar um pedido para o OpenVIM retornar a qual rede abstrata de tráfego de serviço essa interface está conectada:

   "http://<ip-openvim>/openvim/networks/<uuid_rede_abstrata>/connection".

   Esse comando corresponde ao *RESTful request* #1 e a parte referente ao OpenVIM foi implementado nesse projeto sendo descrito em seção 4.4.

7. Baseado no *uuid* da rede de tráfego abstrata, o OpenMANO requisita a informação de qual rede abstrata de tráfego de serviço a requisições de cliente são atendidas:

   "http://<ip-openvim>/openvim/networks/<uuid_rede_abstrata>/bind".

   Esse comando corresponde ao *RESTful request* #2 e a parte referente ao OpenVIM foi implementado nesse projeto sendo descrito em seção 4.4;

11. Os passos 2, 4, 5 e 8 são repetidos (passos 11, 12, 13 e 16, respectivamente) para a obtenção das informações da rede abstrata de tráfego referente à instância replicada da VNFC ao qual está conectada;

19. Com as informações de conectividade carregadas, o OpenMANO envia um comando associando a rede de tráfego que era associada a VM que falhou a rede da VM redundante através do comando RESTful:

    "http://<ip-openvim>/openvim/network/<uuid-rede-a-se-associar>/<uuid-rede-associada>"

    Esse comando corresponde ao *RESTful request* #5 e a parte referente ao OpenVIM foi implementado nesse projeto sendo descrito em seção 4.4;

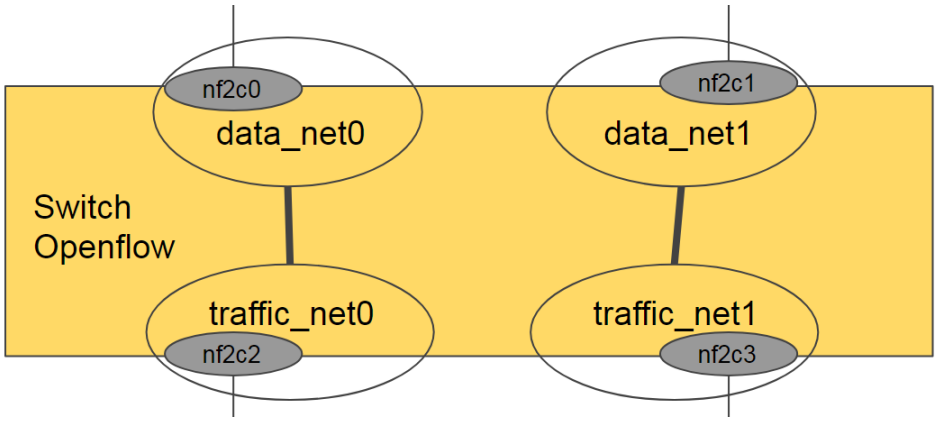
22. Ao final, o campo *'previous_status'* da tabela *instance_vms* deve ser atualizado com o *'status'* atual.

4.4 Implementando a orquestração de rede do OpenMANO sobre o OpenVIM

Para que o OpenMANO consiga orquestrar a conectividade de rede gerenciada pelo OpenVIM, é necessário que as informações sobre a organização da rede sejam disponibilizadas pelo OpenVIM via interface REST. Assim, o OpenMANO pode determinar a conectividade estabelecida no switch OpenFlow presente na arquitetura de maneira abstrata.

A abstração de rede que o OpenVIM implementa representa como esse elemento de controle manipula os recursos de rede. Tais recursos compreendem principalmente as redes

Figura 10 – Modelo de conectividade para o experimento.



virtuais presentes nos *switches* virtuais (OVS), interfaces físicas e virtuais nos *compute nodes* e nos *switches* físicos OpenFlow. A Figura 10 apresenta o modelo de conectividade implementado no experimento que serve de exemplo explicativo e que é formado por duas VMs, uma primária e outra replicada. No caso da primária falhar, os fluxos de dados serão redirecionados para a instância replicada através de modificações no fluxo do switch OpenFlow.

No *switch* OpenFlow, configuramos as conexões físicas nf2cX com cabos de rede Ethernet levando conectividade até o *Compute Node* usando as portas p1pX e nfc2c0 e nf2c1. Para que o OpenVIM possa identificar as portas referentes a redes específicas, uma correlação é feita entre as portas do *switch* OpenFlow e elementos gerenciáveis (redes abstratas). Portanto, o OpenVIM cria redes nomeadas como data_netX. Similarmente, o mesmo é feito para as portas nfc2c2 e nf2c3, sendo relacionadas a redes abstratas traffic_netX. A implementação OpenVIM então associa fluxos OpenFlow de forma a conectar as redes e assim trata a conectividade na rede SDN de forma abstrata. A relação entre as redes é feita por associação entre as redes data_netX com as correspondentes traffic_netX. Essa associação é feita via banco de dados com o campo *bind_net* na tabela nets do OpenVIM. Dessa forma, no momento em que uma falha em VM é detectada, é possível alterar os fluxos de pacotes para a VM redundante. O algoritmo de determinação dos fluxos baseados no modelo de informação está implementado no OpenVIM. A parte implementada neste caso foi a administração da associação das redes que necessitava de comandos REST.

O projeto OSM prevê que possam ser usadas algumas implementações de controlador SDN: ONOS, OpenDayLight ou FloodLight. No caso desse projeto, usaremos o controlador FloodLight pois este está pré-integrado e verificado com o OpenVIM pelo projeto, tendo ampla documentação de instalação e operação.

Abaixo, listamos os componentes relevantes no OpenVIM para a implementação

do redirecionamento de fluxos de dados na rede:

a) httpserver: componente instanciado em forma de *thread* que interpreta os comandos da API northbound e importa o componente ovim para tratamento das requisições feitas pelo OpenMANO;

b) ovim: componente com a lógica principal do gerenciamento de infraestrutura, responsável por iniciar *threads* relacionadas a diferentes tipos de *Compute Nodes* e controladores OpenFlow;

c) vim_db: componente que implementa funções de interação com o banco de dados;

d) floodlight: componente que implementa funções específicas de interação com o controlador FloodLight. É derivado da classe genérica openflow_conn (atualmente é uma *thread*).

Quando o OpenMANO identifica a falha em uma VM, calcula o redirecionamento e envia o seguinte comando REST para o OpenVIM:

"http://<ip-openvim>/openvim/network/<uuid-rede-a-se-associar>/<uuid-rede-associada>"

Esse comando altera a base de dados do OpenVIM, roda o algoritmo de determinação de fluxos OpenFlow e envia as novas regras para o controlador FloodLight.

A implementação desse comando consistiu na inclusão do novo comando REST especificado acima. No componente httpserver incluímos uma nova função *Python* que é mapeada no endereço http do comando REST. Esta função que implementa o tratamento do comando REST recebe os dois parâmetros <uuid-rede-a-se-associar> e <uuid-rede-associada> e realiza as seguintes operações:

1. Altera no banco de dados a entrada identificada por <uuid-rede-a-se-associar> no campo bind_net com o valor de <uuid-rede-associada>;

2. Chama a função do ovim edit_openflow_rules que executa o processo de atualização das regras de fluxos OpenFlow no FloodLight baseado nas informações de rede do banco de dados OpenVIM.

O efeito observado é que os pacotes passam a ser enviados para a VM redundante através dos novos fluxos configurados. No nosso experimento, temos que os fluxos oriundos do gerador de tráfego pela interface enp1s0f0 entram pela porta nf2c2 que estão mapeados na rede abstrata traffic_net0.

Em caso de falha da VM VNF à esquerda na Figura 10, a implementação irá desviar os pacotes da rede traffic_net0 para a rede abstrata data_net1 o que fisicamente significa fluxos no switch OpenFlow da porta nf2c2 para nf2c1 e vice-versa como ilustrado na Figura 7.

5 Avaliação

Para avaliarmos a solução proposta, utilizamos um ambiente minimalista real de distribuição de mídia (vídeo) de um servidor de mídia (Servidor de Mídia *Broadcast* na Figura 12) que envia uma transmissão para os servidores hospedados nas VMs da VNFs (abaixo do servidor de mídia). Estes redistribuem esse conteúdo para conexões RTP (*Real-time Transport Protocol*) sobre UDP como camada de transporte. As VMs representando VNFCs reenviam pacotes em *broadcast* para as portas *ethernet* correspondentes funcionando como repetidores de serviço *broadcast*. O intuito de utilizar um serviço de distribuição de mídia é poder avaliar a continuidade do serviço em uma arquitetura que duas instâncias de VMs compartilhassem o mesmo conteúdo. Dessa forma, a retomada de uma VM replicada apresentaria um conteúdo próximo do qual a aplicação cliente que consome o vídeo assiste, portanto o tempo de *failover* (TTR - *Time To Repair*) determina a percepção de descontinuidade do vídeo. Do ponto de vista da verificação da solução, o *setup* de teste atende ao requisito de haver a possibilidade de provisionamento de duas VNFCs (VMs) e o monitoramento e controle a partir do OpenMANO desse serviço (via OpenVIM). Poderíamos prover dois VIMs diferentes para que o provisionamento via OpenMANO alocasse duas VMs em diferentes *compute nodes*, porém esse provisionamento e controle são agnósticos com relação à localização das VMs. Portanto, o modelo aplicado e descrito na Seção 4 não diferencia a localização das VMs, sendo essa função desempenhada pelo OpenMANO de maneira nativa.

Na Figura 11, ilustramos o *setup* construído em laboratório e composto dos seguintes recursos:

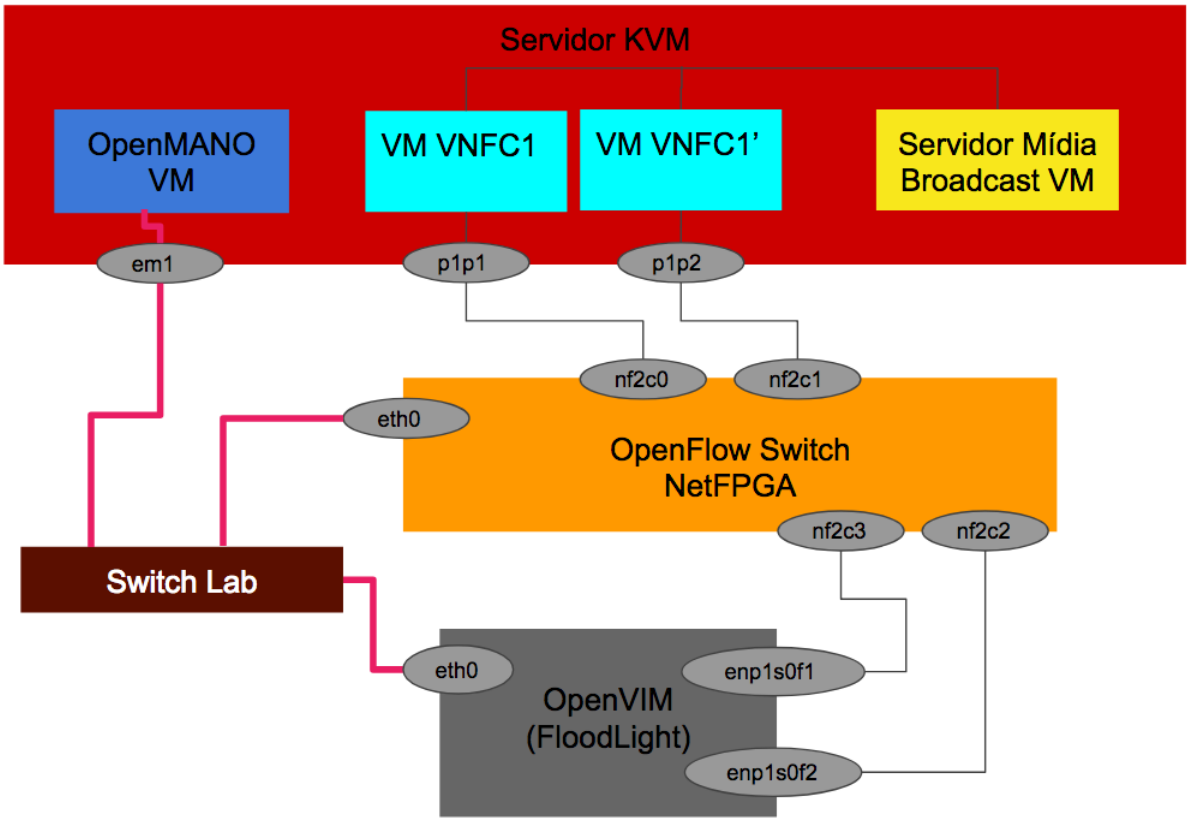
1. Servidor KVM: servidor físico Dell Inc. PowerEdge R420 Dell Inc. PowerEdge R420 com processador Intel(R) Xeon(R) CPU E5-2430 0 @ 2.20GHz, 2 CPUs, 20 *cores*, 40 *threads*, 50 GB de memória, 2 interfaces Broadcom Corporation NetXtreme II BCM57810 10 Gigabit Ethernet para rede de serviço e sistema operacional Ubuntu Server 16.04.

2. OpenVIM: máquina física HP com processador Intel i7 2600 da 2a geração com 3.4Ghz, 16 GB de RAM e 1 TB de disco rígido.

3. FloodLight: co-locado com o OpenVIM.

4. OpenMANO: VM instaciada no Servidor KVM com sistema operacional Ubuntu 14, 2 CPUS x86 e 16 GB de memória.

Figura 11 – *Setup* de experimentação do projeto e cenário de falha.



5. *Switch* OpenFlow: máquina física AMD Phenom(tm) 9850 Quad-Core, 38 GB de memória, 500 GB de disco, com placa NetFPGA 1G Xilinx Virtex-II Pro 50, 4 portas, com *firmware* OpenFlow-NetFPGA-1.0.0 da Standford University.

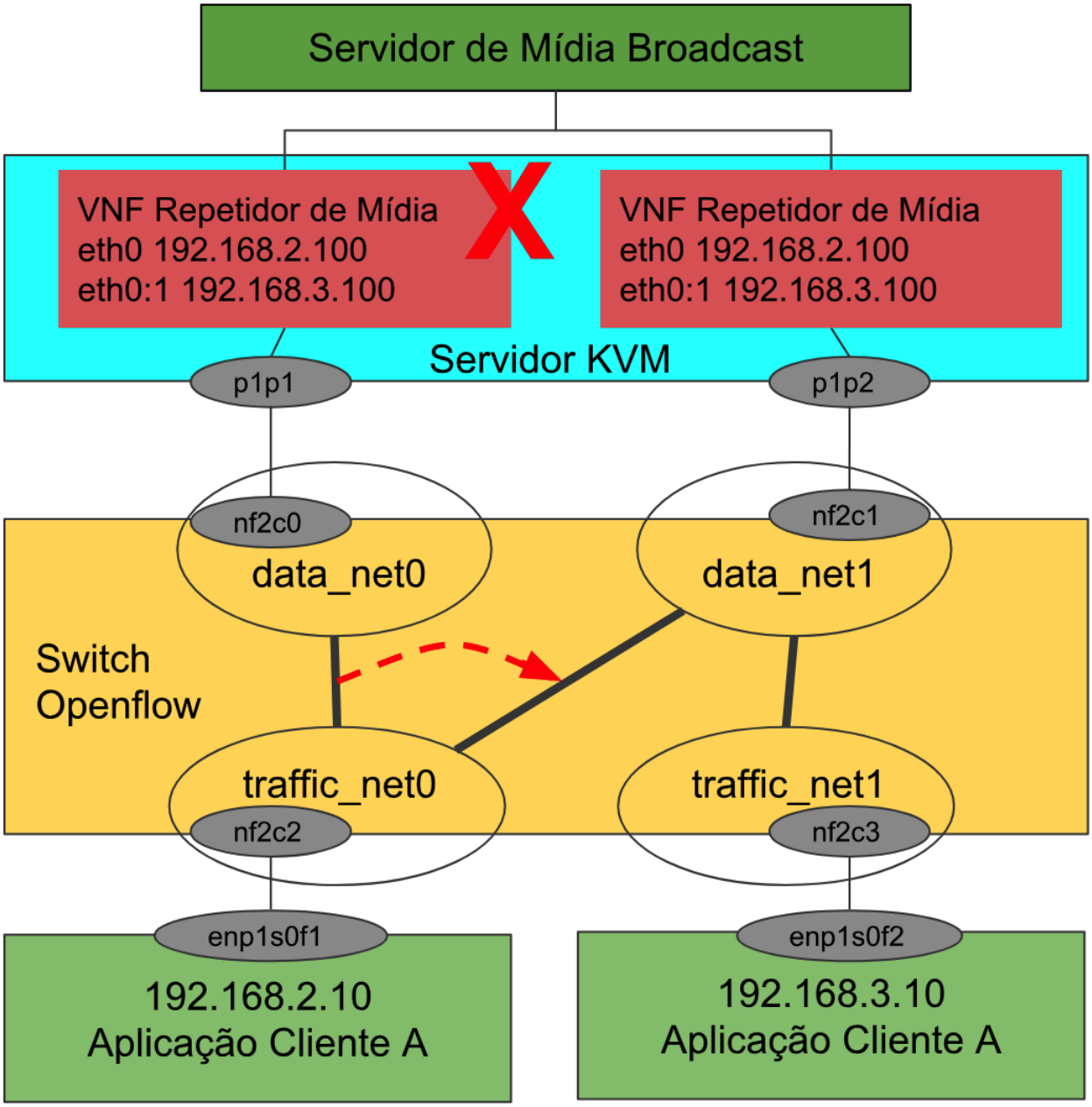
Como apresentado na Figura 12, os fluxos são redirecionados pela solução implementada quando é detectada que o estado de VM à esquerda não está mais operacional.

As aplicações clientes recebem os fluxos de vídeo dos respectivos servidores, mostrando o vídeo em uma aplicação cliente (VIDEOLAN, 2015). Inicialmente, temos pares servidor-cliente isolados pelos fluxos no *switch* OpenFlow.

A configuração de rede das VMs atribui endereços IPs idênticos (Figura 12) para ambas as VMs usando dois endereços IPs para a mesma interface *ethernet*. Um cliente que se conecta a um fluxo de servidor em um determinado domínio irá receber a mídia do servidor que atua na sua região ou domínio. Quando a VM de um servidor de serviço que atende a um determinado domínio falha, o redirecionamento de fluxos irá levar o tráfego ao servidor redundante que possui o mesmo endereço IP.

Nos testes executados, o redirecionamento de fluxos levou as aplicações clientes a atingirem o servidor ativo após simularmos a indisponibilidade de uma das VMs da

Figura 12 – Cenário de falha em que há o redirecionamento de fluxos.



VNF. Como demonstrado na Tabela 4, realizamos medições na cadeia de execução através dos *timestamps* nos arquivos de *tracing* de cada um dos elementos $OpenMANO \rightarrow OpenVIM \rightarrow FloodLight \rightarrow NetFPGA$. Os arquivos de *trace* não trazem precisão de milissegundos para os elementos da cadeia, exceto o FloodLight. O que podemos concluir desses dados é que o OpenMANO leva menos de 1 segundo para identificar uma mudança de estado de VNFC e executar o processo de redirecionamento na rede abstrata. O OpenVIM leva de 1 a um tempo menor que 2 segundos para processar a mudança na configuração de rede e enviar os pedidos de redirecionamento para o FloodLight. Esse, no entanto, leva em

média 682 milissegundos para processar os pedidos e encaminhar os comandos OpenFlow para o *switch* NetFPGA.

Tabela 4 – Medição em testes de simulação de indisponibilidade de VM.

Teste #	**OpenMANO início**	**OpenMANO fim**	**OpenVIM início**	**OpenVIM fim**	**FloodLight início**	**FloodLight fim**	**NetFPGA início**	**NetFPGA fim**	**TTR (s)**
1	20:31:02	20:31:02	20:31:02	20:31:03	20:31:02.658	20:31:03.451	20:31:02	20:31:03	1
2	20:43:11	23:43:12	20:43:12	20:43:12	20:43:12.347	20:43:13.084	20:43:12	20:43:13	2
3	20:51:11	20:51:11	20:51:11	20:51:12	20:51:11.779	20:51:12.374	20:51:12	20:51:12	1
4	21:08:51	21:08:52	21:08:52	21:08:52	21:08:52.479	21:08:53.105	21:08:52	21:08:53	2
5	21:19:08	21:19:08	21:19:08	21:19:08	21:19:09.471	21:19:10.052	21:19:09	21:19:10	2
6	21:25:07	21:25:07	21:25:07	21:25:08	21:25:08.192	21:25:08.574	21:25:08	21:25:08	1
7	21:29:36	21:29:36	21:29:36	21:29:37	21:29:36.772	21:29:37.348	21:29:37	21:29:37	1
8	21:34:49	21:34:49	21:34:49	21:34:50	21:34:49.835	21:34:50.663	21:34:50	21:34:50	1
9	21:39:23	21:39:23	21:39:23	21:39:24	21:39:23.257	21:39:24.184	21:39:23	21:39:24	1
10	21:56:36	21:56:36	21:56:36	21:56:37	21:56:36.445	21:56:37.224	21:56:36	21:56:37	1

Nota: TTR (*Time To Repair*), no caso do experimento, é o tempo que leva para que a cadeia de processamento e comunicação necessários para a determinação e efetivação da mudança de fluxos no *switch OpenFlow*.

Quando consideramos a qualidade de experiência (QoE) ao analisar a fluidez do vídeo transmitido, percebemos que a fluidez foi interrompida por um intervalo que varia de 2 a 5 segundos. Isso se deve à retomada do fluxo de vídeo que o aplicativo VLC realiza. O conteúdo do *buffer* acumulado pelo VLC não corresponde ao conteúdo transmitido pelo servidor replicado e, portanto, tem de reiniciar o vídeo do ponto de transmissão em que o servidor replicado está no momento da retomada.

Conseguimos observar que aspectos de gerenciamento e orquestração levaram a alterações de código e implementações que seguiram a padronização proposta pela ETSI NFV, com o cumprimento dos papéis definidos para o OpenMANO e OpenVIM. É clara a intenção de delegar ao OpenMANO a responsabilidade de tomada de decisão sobre as ações a serem executadas baseando-se nos dados de alta disponibilidade. Em nosso teste, utilizamos uma configuração simples que contemplava somente um VIM. No contexto de orquestração, VMs de uma mesma VNF podem estar distribuídas em diferentes VIMs.

6 Conclusão e Trabalhos Futuros

Nesta dissertação, apresentamos os detalhes de uma proposta de implementação para suportar alta disponibilidade em um framework NFV no que tange à organização de rede para possibilitar o acesso à instância redundante de um serviço VNF, descrevendo as implementações realizadas em um experimento utilizando a plataforma OSM. Podemos concluir que o projeto OSM tem conseguido mapear os requisitos e definições da padronização ETSI, tornando-se assim uma plataforma em que o framework permite a implementação de requisitos de alta disponibilidade da forma como definida no padrão. Nossa intenção é interagir com a comunidade que contribui para o projeto OSM a fim de disponibilizar nosso código para a plataforma de maneira aberta.

As Tabela 5 e Tabela 6 resumem as implementações feitas no código aberto do OSM e ilustra a diferença de papéis do OpenMANO e OpenVIM, respectivamente.

Tabela 5 – Resumo da implementação no OpenMANO.

Componentes do OpenMANO	**Implementação**
Banco de dados	Inserção dos novos campos *high_availability*, *replicated* e *service_interface*na tabela *instance_vms*
vim_thread	Implementação de registro de chamada de *callback* do *nfvo* quando um evento de mudança de estado de VM é identificado.
nfvo	Implementação da função que é chamada pelo *vim_thread* quando um evento de mudança de estado de VM é identificado. A função analisa se uma VM é parte de uma VNF em alta disponibilidade e calcula quais mudanças na organização de rede são necessárias.
vimconn	Classe base para as implementações específicas de classes que interfaceiam com VIMs definidos (OpenVIM, OpenStack ou AWS). Alteramos para incluir a assinatura de função de alteração de conectividade de rede.
vimconn_openvim	Implementação da função *changeNetworkBind* que é chamada para alterar o *bind* de duas redes existentes e assim redirecionar os fluxos.

Tabela 6 – Resumo da implementação no OpenVIM.

Componente do OpenVIM	**Implementação**
ovim	Desenvolvimento de função que implementa o comando de alteração de *bind* de rede. Altera o banco de dados no campo *bind_net* e chama *edit_openflow_rules* para que ocorra a redifinição dos fluxos e envio dos comandos OpenFlow do FloodLight para o *switch* físico.
httpserver	Implementação dos parâmetros de interface (json) e comandos REST do servidor HTTP do OpenVIM..

Os papéis do MANO e VIM na padronização podem ser observados como mapeados no OpenMANO e OpenVIM, respectivamente. A orquestração deve trazer a visão geral sobre os serviços nas quais a alta disponibilidade está inserida, no que diz respeito ao impacto

sobre o encadeamento dos serviços, ao passo que o gerenciador de infraestrutura deve oferecer as informações sobre a conectividade e situação dos serviços por este gerenciados por meio das interfaces definidas pelos *reference points* da padronização. Em serviços dependentes de contexto com conexões TCP orientadas a sessões de aplicação, a conexão tem que ser refeita caso o contexto não seja replicado. Como descrevemos no início desta dissertação, não estamos avaliando a alta disponibilidade do ponto de vista da aplicação, o que exigiria o estudo e a implementação de uma aplicação complexa com avaliação detalhada e comparativa.

Um grande desafio a ser estudado é o impacto de serviços complexos encadeados e sujeitos à alta disponibilidade. A distribuição desses serviços em vários domínios pode gerar uma grande complexidade em relação à implementação da alta disponibilidade. Como descrito por Ordonez-Lucena et al. (2017), dado o dinamismo e escalabilidade que o particionamento de serviços de redes apresenta, o gerenciamento e orquestração não são implementações simples.

Como trabalhos futuros temos os seguintes tópicos propostos:

1. Proposta de mecanismo de monitoramento de eventos do OpenVIM para o OpenMANO: como descrito na Seção 4, o OSM carece de uma solução que viabilize a notificação em tempo real de eventos (modelo *push*) de mudança de estado de VMs como previsto pela padronização ETSI (2014c, seção 5.7.7).

2. Experimentação com um serviço NFV: um serviço como o proposto neste trabalho (serviço de distribuição de mídia) é representativo pela avaliação da continuidade de serviço. Como enriquecimento da experimentação, podemos implementar e avaliar também aspectos funcionais e de desempenho de aplicação típica NFV como NAT ou DPI. Um aspecto importante é o suporte da aplicação à alta disponibilidade como proposto por Rajagopalan, Williams e Jamjoom (2013) (PICO). *PICO Replication* é um *framework* sistêmico que implementa replicação de pacotes por fluxos (*flow-level replication*). Esse mecanismo possibilita a verificação de falhas em altas frequências enquanto o processo principal continua tratando as informações. O *PICO Replication* pode ser utilizado no contexto da padronização especificada em ETSI GS NFV-REL 002 (ETSI, 2014a) em que são definidos métodos para alcançar a alta disponibilidade como o *Lightweight Rollback Recovery*.

3. Melhorias no provisionamento das informações de alta disponibilidade no OpenMANO: como descrito na Seção 4, uma solução completa deve incluir o provisionamento das informações de alta disponibilidade no OpenMANO, tal implementação consiste em alterações em interfaces gráficas e processos de instanciação de entidades dentro do OpenMANO.

4. Expansão do experimento para um contexto de mais amplo de encadeamento de serviço: aplicação do experimento para serviço encadeados e os efeitos na reorquestração dos serviços.

7 Apendice

7.1 Modificações e inserções de código no OpenMANO

Código inserido em **nfvo.py**:

```
def extract_netid_from_vim_info(vim_info, interface):
    # search for information on network connectivity from vm_info
    net_id_pos = string.find(vim_info, "net_id",string.find(vim_info, interface))
    net_id = vim_info[net_id_pos + 8:net_id_pos + 8 + 36]
    return net_id



def _callback_vm_event(vim_vm_id, datacenter_tenant_id):
    '''
    Callback function upon VM status change reported by VIM thread
    '''

    logger.debug("_callback_vm_event: NOTIFICATION ARRIVED vim_vm_id %s : datacenter_tenant_id %s", \
                                                                    vim_vm_id, datacenter_tenant_id)

    net_id = None
    bind_net_id = None

    # Initialize DB connection
    mydb = nfvo_db.nfvo_db();
    mydb.connect(global_config['db_host'], global_config['db_user'], \
                global_config['db_passwd'], global_config['db_name'])

    try:
        active_instance = mydb.get_rows(SELECT=('status','high_availability', 'previous_status', \
                                                'vim_info','replicated_id'), 'service_interface' \
                                                FROM='instance_vms' \
                                                WHERE={'vim_vm_id':vim_vm_id} )
        if len(active_instance) == 0:
            logger.warn("nfvo.get_hosts virtual machine at VIM '{}' not found".format(vim_vm_id))
        else:
            tenant_id = mydb.get_rows(SELECT=('nfvo_tenant_id',), \
                                                FROM='tenants_datacenters', \
                                                WHERE={'datacenter_tenant_id':datacenter_tenant_id} )
            logger.debug("_callback_vm_event: tenants = %s", tenant_id)
            for tenant in tenant_id:
                vim_id, vim = get_vim_thread(mydb, tenant['nfvo_tenant_id'])

            for vm_status in active_instance:
                logger.debug("_callback_vm_event: change in status for active_instance %s: " + \
                                "[status=%s] [previous_status=%s] [high_availability=%s]", \
                                vim_vm_id, vm_status['status'], vm_status['previous_status'], \
                                vm_status['high_availability'])
                if vm_status['high_availability'] == 'ACTIVE_ACTIVE' or \\
                    vm_status['high_availability'] == 'ACTIVE_PASSIVE':
                    logger.debug("_callback_vm_event: active_instance high_availability " + \
                                                                    "= ACTIVE_ACTIVE")
```

```
if vm_status['status'] == "INACTIVE" or \
      vm_status['status'] == "ERROR" or \
      vm_status['status'] == "PAUSED":
   logger.debug("_callback_vm_event: active_instance vm_status['status'] " + \
                                                "== %s", vm_status['status'])

   if vm_status['status'] != vm_status['previous_status']:
      # status have change, action must be taken
      logger.debug("_callback_vm_event: active_instance %s", \
                                                vm_status['vim_info'])

      # search for information on network connectivity for active vm
      net_id_service = extract_netid_from_vim_info(vm_status['vim_info'], \
                                      vm_status['service_interface'])
      logger.debug("_callback_vm_event: active_instance net_id_service %s", \
                                                   net_id_service)

      r, net_id_active_conn = vim.vim.getNetworkConnection(net_id_service)
      logger.debug("_callback_vm_event: active_instance net_id_active_conn %s", \
                                                net_id_active_conn['net_id'])

      r, net_id_active_service = \
              vim.vim.getNetworkBind(net_id_active_conn['net_id'])
      logger.debug("_callback_vm_event: active_instance " + \
                     "net_id_active_conn %s to net_id_active_service %s", \
                           net_id_active_conn['net_id'], net_id_active_service)
      #take network from standby instance
      standby_instance = mydb.get_rows(SELECT=('status','high_availability', \
                                                'previous_status','vim_info', \
                                                'replicated_id', 'service_interface') \
                                               FROM='instance_vms', \
                                          WHERE={'uuid':vm_status['replicated_id']} )
      if len(standby_instance) == 0:
         logger.warn("_callback_vm_event: '{}' not found in " + \
                     "instance_vms".format(vm_status['replicated_id']))
      else:
         for vm_status_standby in standby_instance:
             net_id_service = \
                      extract_netid_from_vim_info(vm_status['vim_info'], \
                                      vm_status['service_interface']
             logger.debug("_callback_vm_event: net_id_service_standby %s", \
                                                        net_id_service)

             r, net_id_standby_service = vim.vim.getNetworkConnection(net_id_service)
             logger.debug("_callback_vm_event: standby_instance " + \
                                      "net_id_standby_service %s", \
                                      net_id_standby_service['net_id'])

             r, net_id_standby_bind = \
                      vim.vim.getNetworkBind(net_id_standby_bind['net_id'])
             logger.debug("_callback_vm_event: standby_instance " + \
                      "net_id_standby_conn %s binded to net_id_standby_bind %s", \
                              net_id_standby_service['net_id'], net_id_standby_bind)
             vim.vim.changeNetworkBind(net_id_standby_traffic['bind_id']['uuid'], \
                                                net_id_active_conn['net_id'])
             logger.debug("_callback_vm_event: SWITCH PROCESS TERMINATED")
```

```
                    # previous status = status since there was a change
                    mydb.update_rows("instance_vms", UPDATE={"previous_status": vm_status['status']},
                                     WHERE={"vim_vm_id": vim_vm_id})

    except db_base_Exception as e:
            logger.warn("nfvo._callback_vm_event virtual machine VM {} error {}".format(vim_vm_id), \
                                                                                  str(e))

    return
```

Código inserido em **vimconn.py**, classe base para interfacear com VIM (classes derivadas como por exemplo vimconn_openvim).

```
    def changeNetworkBind(self, network_id, bind_net_id=None):
        """Change network's bind that influences packet routing in OpenFlow switch"""
        """Returns status code of the VIM response"""
        raise vimconnNotImplemented( "Should have implemented this" )

    def getNetworkBind(self, network_id):
        """Change network's bind that influences packet routing in OpenFlow switch"""
        """Returns status code of the VIM response"""
        raise vimconnNotImplemented( "Should have implemented this" )

    def getNetworkConnection(self, network_id):
        """Change network's bind that influences packet routing in OpenFlow switch"""
        """Returns status code of the VIM response"""
        raise vimconnNotImplemented( "Should have implemented this" )
```

Código inserido em **vimconn_openvim.py**, classe base para interfacear com OpenVIM (classe derivada de *vimconn_openvim*).

```
'''
Schema json formats for data exchange with OpenVIM
'''
bind_schema = {
    "title":"openvim bind_net response information schema",
    "$schema": "http://json-schema.org/draft-04/schema#",
    "type":"object",
    "properties":{
        "uuid":id_schema,
    },
    "required": ["uuid"]
}

get_bind_nets_response_schema = {
    "title":"openvim bind_net response information schema",
    "$schema": "http://json-schema.org/draft-04/schema#",
    "type":"object",
    "properties":{
        "bind_id":{
            "type":"array",
            "items": bind_schema,
        }
    },
    "required": ["bind_id"],
```

```
    "additionalProperties": False
}

net_id_schema = {
    "title":"openvim net response information schema",
    "$schema": "http://json-schema.org/draft-04/schema#",
    "type":"object",
    "properties":{
        "net_id":id_schema,
    },
    "required": ["net_id"]

}
get_network_connection_response_schema ={
    "title":"openvim net_connection response information schema",
    "$schema": "http://json-schema.org/draft-04/schema#",
    "type":"object",
    "properties":{
        "net_conn":{
            "type":"array",
            "items": net_id_schema,
        }
    },
    "required": ["net_conn"],
    "additionalProperties": False

}


'''
Methods to interface with OpenVIM to request data and change network bind
'''



    def changeNetworkBind(self, network_id, bind_net_id=None):
        '''Connects network_id to bind_net_id'''
        '''Returns status code of the VIM response'''

        self.logger.debug("VIMConnector: Binding networks network_id=%s to %s", network_id, bind_net_id)

        url= self.url
        try:
            if bind_net_id == None:
                vim_response = requests.delete(url +'/networks/'+network_id+'/delete', \
                                               headers = self.headers_req)
            else:
                vim_response = requests.put(url +'/networks/'+network_id+'/'+bind_net_id, \
                                                    headers = self.headers_req)

            ## refresh openflow rules
            vim_response = requests.delete(url +'/networks/clear/openflow', \
                                                    headers = self.headers_req)
            vim_response = requests.put(url +'/networks/all/openflow', \
                                                    headers = self.headers_req)

        except requests.exceptions.RequestException as e:
            self.logger.warn("bind_network Exception: %s", e.args);
            return -vimconn.HTTP_Not_Found, str(e.args[0])
        print vim_response
```

```
def getNetworkBind(self, network_id):
    '''Get bind_network_id to network_id'''
    '''Returns status code of the VIM response'''

    self.logger.debug("VIMConnector: Get Bind networks network_id=%s", network_id)

    url= self.url
    try:
        vim_response = requests.get(url +'/networks/'+network_id+'/bind', \
                                                    headers = self.headers_req)

    except requests.exceptions.RequestException as e:
        self.logger.warn("bind_network Exception: %s", e.args);
        return -vimconn.HTTP_Not_Found, str(e.args[0])

    res, bind_nets = self._format_in(vim_response, get_bind_nets_response_schema)

    if res==False:
        print "vimconnector.getNetworkBind error parsing GET BIND_NET vim response", \
                                                                        bind_nets
        return vimconn.HTTP_Internal_Server_Error, bind_nets
    return 200, bind_nets

def getNetworkConnection(self, network_id):
    '''Get network_id to bridge network to external to host network_id '''
    '''Returns status code of the VIM response'''

    self.logger.debug("VIMConnector: Get networks connection network_id=%s", network_id)

    url= self.url
    try:
        vim_response = requests.get(url +'/networks/'+network_id+'/connection', \
                                                    headers = self.headers_req)

    except requests.exceptions.RequestException as e:
        self.logger.warn("bind_network Exception: %s", e.args);
        return -vimconn.HTTP_Not_Found, str(e.args[0])
    res, net_id_connection = self._format_in(vim_response, \
                                    get_network_connection_response_schema)
    if res==False:
        print "vimconnector.getNetworkConnection error parsing"+ \
                    "GET NET CONNECTION vim_response", '\
                                                    net_id_connection
        return vimconn.HTTP_Internal_Server_Error, net_id_connection

    print "vimconnector.getNetworkConnection ", vim_response
    return 200, net_id_connection['net_conn']
```

7.2 Modificações e inserções de código no OpenVIM

Código inserido em **httpserver.py**, usada por uma *thread* para interfacear com o orquestrador.

```
'''
httpserver.py implements a set of functions to deal with HTTP REST commands
on OpenVIM side

'''

@bottle.route(url_base + '/networks/<network_id>/<bind_net_id>', method='PUT')
def changeNetworkBind(network_id=None, bind_net_id=None):
    """
    Change bind of a network_id to bind_net_id
    If bind_net_id=None then bindness is deleted
    """
    my = config_dic['http_threads'][threading.current_thread().name]
    my.logger.debug("changeNetworkBind: PROCESS TO REDIRECT FLOW INITIATED")
    try:
        my.ovim.changeNetworkBind(network_id, bind_net_id)
        my.ovim.delete_openflow_rules(None)
        my.ovim.edit_openflow_rules(None)

        #my.ovim.net_update_ofc_thread(network_id)
        #if bind_net_id != None:
        #   my.ovim.net_update_ofc_thread(bind_net_id)
    except ovim.ovimException as e:
        my.logger.error(str(e), exc_info=True)
        bottle.abort(e.http_code, str(e))
    except Exception as e:
        my.logger.error(str(e), exc_info=True)
        bottle.abort(HTTP_Bad_Request, str(e))

    return

@bottle.route(url_base + '/networks/<network_id>/<bind_net_id>', method='DELETE')
def removeNetworkBind(network_id=None, bind_net_id=None):
    """
    Change bind of a network_id to bind_net_id
    If bind_net_id=None then bindness is deleted
    """
    my = config_dic['http_threads'][threading.current_thread().name]

    try:
        my.ovim.changeNetworkBind(network_id, None)
        my.ovim.net_update_ofc_thread(network_id)
        if bind_net_id != None:
            my.ovim.net_update_ofc_thread(bind_net_id)
    except ovim.ovimException as e:
        my.logger.error(str(e), exc_info=True)
        bottle.abort(e.http_code, str(e))
    except Exception as e:
        my.logger.error(str(e), exc_info=True)
        bottle.abort(HTTP_Bad_Request, str(e))

    return

@bottle.route(url_base + '/networks/<network_id>/bind', method='GET')
```

```
def http_get_network_bind(network_id):
    '''
    Reply with network id of bind network for <network_id>
    '''

    my = config_dic['http_threads'][ threading.current_thread().name ]

    try:
        net_id_conn = my.ovim.get_net_id_bind(network_id)
        delete_nulls(net_id_conn)
        net_id_conn = net_id_conn[1]
        data = {'bind_id':net_id_conn}
        return format_out(data)
    except ovim.ovimException as e:
        my.logger.error(str(e), exc_info=True)
        bottle.abort(e.http_code, str(e))
    except Exception as e:
        my.logger.error(str(e), exc_info=True)
        bottle.abort(HTTP_Bad_Request, str(e))

@bottle.route(url_base + '/networks/<network_id>/connection', method='GET')
def http_get_network_connection(network_id):
    '''
    Reply with network id in which network_id is connected
    '''
    my = config_dic['http_threads'][ threading.current_thread().name ]

    try:
        net_id_conn = my.ovim.get_net_id_connection(network_id)
        delete_nulls(net_id_conn)
        data = {'net_conn':net_id_conn}
        return format_out(data)
    except ovim.ovimException as e:
        my.logger.error(str(e), exc_info=True)
        bottle.abort(e.http_code, str(e))
    except Exception as e:
        my.logger.error(str(e), exc_info=True)
        bottle.abort(HTTP_Bad_Request, str(e))
```

Código inserido em **ovim.py**, classe principal do OpenVIM em que métodos implementam as funções de gerenciamento necessárias para o suporte à alta disponibilidade.

```
    def changeNetworkBind(self, network_id, bind_net_id):

        if bind_net_id == None:
            self.db.update_rows("nets", UPDATE={"bind_net": None}, \
                                        WHERE={"uuid": network_id})
        else:
            self.db.update_rows("nets", UPDATE={"bind_net": bind_net_id}, \
                                        WHERE={"uuid": network_id})

        return

    def get_net_id_bind(self, network_id, limit=None):

        result, content = self.db.get_table(SELECT={'uuid':'uuid'}, FROM='nets', \
                                                WHERE={'bind_net':network_id})
        self.logger.info("get_net_id_bind: network %s is bind to %s",
                                                        \content, network_id)
        if result < 0:
```

```
            raise ovimException(str(content), -result)
        return result, content

    def get_resource_ports(self, columns=None, port_name=None, limit=None):

        result, content = self.db.get_table(SELECT=columns, \
                                            FROM='resources_port', \
                                    WHERE={'source_name':port_name}, LIMIT=limit)
        self.logger.info("get_net_id_connection: resource port - %s", content)
        if result < 0:
            raise ovimException(str(content), -result)

        return result, content

    def get_net_id_connection(self, net_id):
        """This function finds a network id for an abstract network attached
        to a switch port in which net_id is physically connected
        """
        result, content = self.db.get_table(SELECT={'name', 'type'}, \
                                        FROM='nets', WHERE={'uuid':net_id})

        if result < 0:
            raise ovimException(str(content), -result)

        for net in content:
            self.logger.info("get_net_id_connection: net %s type %s", \
                                                net['name'], net['type'])
            if net['type'] == 'bridge_data':
               #get  host interface name for port out
               net_name = net['name']
               port_name = net_name[net_name.find('macvtap:')+8:len(net_name)]
               self.logger.info("get_net_id_connection: port_name - %s", port_name)
            try:
                #search for network linked to switch port
                result, content1 = self.get_resource_ports({'switch_port':'switch_port'}, \
                                                                    port_name)
                if result < 0:
                   raise ovimException(str(content1), -result)
                self.logger.info("get_net_id_connection: switch port - %s", content1)
                result, content2 = self.db.get_table(SELECT={'net_id'}, \
                                                FROM='ports', WHERE=content1[0])
                self.logger.info("get_net_id_connection: net connection - %s", content2)
                if result < 0:
                   raise ovimException(str(content2), -result)

            except Exception as e:
                self.logger.error("get_net_id_connection: %s", e)
                return None
        return content2
```

Referências

AVIZIENIS, A. et al. Basic concepts and taxonomy of dependable and secure computing. *IEEE transactions on dependable and secure computing*, IEEE, v. 1, n. 1, p. 11–33, 2004. Citado na página 35.

ETSI. Etsi, nfvisg - network functions virtualization, white paper. *(2012-10-20)[2014-02-14]. http://www.etsi.org/technologies-clusters/technologies/nfv*, 2012. Citado na página 36.

ETSI. Technologies and clusters;technologies;nfv. 2012. Disponível em: <http://www.etsi.org/technologies-clusters/technologies/nfv>. Citado na página 39.

ETSI. Etsi gs nfv-rel 002 - reliability; report on scalable architectures for reliability management. 2014. Disponível em: <http://www.etsi.org/deliver/etsi_gs/NFV-REL/001_099/002/01.01.01_60/gs_nfv-rel002v010101p.pdf>. Citado na página 66.

ETSI. Gs nfv-man 001 v1.1.1 (2014-12) - network functions virtualisation (nfv); management and orchestration. p. 45, 2014. Disponível em: <http://www.etsi.org/deliver/etsi_gs/NFV-MAN/001_099/001/01.01.01_60/gs_NFV-MAN001v010101p.pdf>. Citado na página 39.

ETSI. Network functions virtualisation (nfv); management and orchestration. In: _____. *ETSI GS NFV-MAN 001 V1.1.1*. 650 Route des Lucioles F-06921 Sophia Antipolis Cedex - FRANCE: [s.n.], 2014. p. 45. Citado 4 vezes nas páginas 43, 49, 51 e 66.

ETSI. Gs nfv-rel 001 v1.1.1 (2015-01) - network functions virtualisation (nfv); resiliency requirements. p. 14, 2015. Disponível em: <http://www.etsi.org/deliver/etsi_gs/NFV-REL/001_099/001/01.01.01_60/gs_NFV-REL001v010101p.pdf>. Citado na página 41.

ETSI. Gs nfv 002 v1.2.1 (2014-12) - network functions virtualisation (nfv); architectural framework. p. 14, 2018. Disponível em: <http://www.etsi.org/deliver/etsi_gs/NFV/001_099/002/01.02.01_60/gs_NFV002v010201p.pdf>. Citado 2 vezes nas páginas 39 e 40.

ETSI ISG NFV. "network functions virtualization industry specification group etsi site". 2012. Disponível em: <http://www.etsi.org/technologies-clusters/technologies/nfv>. Citado na página 29.

GLOSSARY, T. *Document T1. 523-2007, Alliance for Telecommunications Industry Solutions*. 2007. Citado na página 28.

GONZALEZ, A. et al. Service availability in the nfv virtualized evolved packet core. In: IEEE. *Global Communications Conference (GLOBECOM), 2015 IEEE*. [S.l.], 2015. p. 1–6. Citado 2 vezes nas páginas 37 e 38.

HAUTAKORPI, J. *RFC 5853:" Requirements from SIP Session Border Control Deployments", 2010 April*. [S.l.]: ISSN, 2010. Citado na página 38.

KIM, T.; KOO, T.; PAIK, E. Sdn and nfv benchmarking for performance and reliability. In: IEEE. *Network Operations and Management Symposium (APNOMS), 2015 17th Asia-Pacific*. [S.l.], 2015. p. 600–603. Citado 2 vezes nas páginas 33 e 34.

LOPEZ, D. R. Network functions virtualization: Beyond carrier-grade clouds. In: IEEE. *Optical Fiber Communications Conference and Exhibition (OFC), 2014*. [S.l.], 2014. p. 1–18. Citado 2 vezes nas páginas 37 e 41.

MAKAYA, C. et al. Policy-based nfv management and orchestration. In: IEEE. *Network Function Virtualization and Software Defined Network (NFV-SDN), 2015 IEEE Conference on*. [S.l.], 2015. p. 128–134. Citado 2 vezes nas páginas 32 e 34.

MIJUMBI, R. et al. Management and orchestration challenges in network functions virtualization. *IEEE Communications Magazine*, IEEE, v. 54, n. 1, p. 98–105, 2016. Citado na página 29.

MONTELEONE, G.; PAGLIERANI, P. Session border controller virtualization towards"service-defined"networks based on nfv and sdn. In: IEEE. *Future Networks and Services (SDN4FNS), 2013 IEEE SDN for*. [S.l.], 2013. p. 1–7. Citado 3 vezes nas páginas 31, 34 e 38.

Open Source MANO. Open source mano project site. 2016. Disponível em: <https://osm.etsi.org/>. Citado na página 29.

OPENBATON. Openbaton git site. 2016. Disponível em: <http://openbaton.github.io/index.html#about>. Citado 2 vezes nas páginas 32 e 34.

OPNFV. High availability for opnfv. 2016. Disponível em: <https://wiki.opnfv.org/display/availability/High+Availability+For+OPNFV>. Citado na página 31.

OPNFV. Open platform for nfv (opnfv). 2016. Disponível em: <https://www.opnfv.org/>. Citado 2 vezes nas páginas 31 e 34.

ORDONEZ-LUCENA, J. et al. Network slicing for 5g with sdn/nfv: Concepts, architectures, and challenges. *IEEE Communications Magazine*, IEEE, v. 55, n. 5, p. 80–87, 2017. Citado na página 66.

OSM Projects. 2015. Disponível em: <https://osm.etsi.org/gitweb/>. Citado na página 41.

QIU, Q. et al. Design a high availability platform for the carrier grade applications. In: *2011 IEEE 3rd International Conference on Communication Software and Networks*. [S.l.: s.n.], 2011. Citado 2 vezes nas páginas 28 e 35.

RAJAGOPALAN, S.; WILLIAMS, D.; JAMJOOM, H. Pico replication: A high availability framework for middleboxes. In: ACM. *Proceedings of the 4th annual Symposium on Cloud Computing*. [S.l.], 2013. p. 1. Citado na página 66.

RIFT.IO. Virtual network function descriptor (vnfd:vnfd) - vdu data model (vnfd:vdu). In: _____. *MANO Descriptor Reference Guide*. Burlington, US: [s.n.], 2017. Disponível em: <https://open.riftio.com/documentation/riftware/4.3/a/descriptor/vnfd/vnfd-vdu-data-model.htm>. Acesso em: 20 mar. 2018. Citado 2 vezes nas páginas 43 e 50.

ROSSEM, S. V. et al. Deploying elastic routing capability in an sdn/nfv-enabled environment. In: IEEE. *Network Function Virtualization and Software Defined Network (NFV-SDN), 2015 IEEE Conference on*. [S.l.], 2015. p. 22–24. Citado 2 vezes nas páginas 33 e 34.

SOENEN, T. et al. Optimising microservice-based reliable nfv management & orchestration architectures. In: IEEE. *Resilient Networks Design and Modeling (RNDM), 2017 9th International Workshop on*. [S.l.], 2017. p. 1–7. Citado 2 vezes nas páginas 32 e 34.

TALEB, T.; KSENTINI, A.; SERICOLA, B. On service resilience in cloud-native 5g mobile systems. *IEEE Journal on Selected Areas in Communications*, IEEE, v. 34, n. 3, p. 483–496, 2016. Citado 2 vezes nas páginas 31 e 34.

"TELEFóNICA NFV Reference Lab". 1988. Disponível em: <http://www.tid.es/long-term-innovation/network-innovation/telefonica-nfv-reference-lab>. Citado na página 41.

TRIVEDE, K. S. et al. Modeling high availability. In: IEEE. *2006 12th Pacific Rim International Symposium on Dependable Computing (PRDC'06)*. [S.l.], 2006. p. 154–164. Citado na página 28.

VIDEOLAN. Videolan website. 2015. Disponível em: <https://www.videolan.org/>. Citado na página 60.

YRJO, R.; RUSHIL, D. Cloud computing in mobile networks—case mvno. In: IEEE. *Intelligence in Next Generation Networks (ICIN), 2011 15th International Conference on*. [S.l.], 2011. p. 253–258. Citado na página 27.