Simplificando o Gerenciamento do Ciclo de Vida de Funções Virtualizadas de Rede

Giovanni Venâncio[1], Vinícius Fulber Garcia[2], Leonardo da Cruz Marcuzzo[2], Thales Nicolai Tavares[2], Muriel Figueredo Franco[3], Lucas Bondan[3], Alberto Egon Schaeffer-Filho[3], Carlos Raniery Paula dos Santos[2], Lisandro Zambenedetti Granville[3], Elias P. Duarte Jr.[1]

[1]Universidade Federal do Paraná (UFPR)
Caixa Postal 19.081 – 81.531-980 – Curitiba, PR – Brasil

[2]Universidade Federal de Santa Maria (UFSM)
Av. Roraima nº 1000 – 97.105-900 – Santa Maria – RS – Brasil

[3]Universidade Federal do Rio Grande do Sul (UFRGS)
Caixa Postal 15.064 – 91.501-970 – Porto Alegre – RS – Brasil

{gvsouza,elias}@inf.ufpr.br
{vfulber,lmarcuzzo,tntavares,csantos}@inf.ufsm.br
{mffranco,lbondan,alberto,granville}@inf.ufrgs.br

Abstract. *One of the main challenges in Network Functions Virtualization (NFV) is to properly manage the lifecycle of network functions. Currently, solutions for managing Virtualized Network Functions (VNFs) are complex and require a deep understanding of the underlying infrastructure. This work proposes an architecture for the lifecycle management of VNFs that leverages different platforms and VNF utilization scenarios, simplifying management operations and reducing the need for the network operator to know the details of the virtualized infrastructure. Experimental results demonstrate the effectiveness of the proposed architecture in terms of the performance of VNF lifecycle management operations.*

Resumo. *Um dos principais desafios da Virtualização de Funções de Rede (NFV) é o gerenciamento do ciclo de vida de funções de rede. Atualmente, as soluções para o gerenciamento de Funções Virtualizadas de Rede (VNFs) são complexas e necessitam de um entendimento profundo da infraestrutura subjacente. Este trabalho propõe uma arquitetura para a gerência do ciclo de vida das VNFs que fornece compatibilidade entre diferentes plataformas e cenários de utilização de VNFs, simplificando as operações de gerência na medida em que diminui a necessidade de que o operador das funções conheça detalhes da infraestrutura virtualizada. Resultados obtidos em um ambiente experimental demonstram a efetividade da arquitetura proposta na realização de todas as operações do gerenciamento do ciclo de vida de VNFs.*

1. Introdução

A Virtualização de Funções de Rede (*Network Function Virtualization* ou NFV) apresenta diversas vantagens em comparação com redes tradicionais, tais como a flexibilidade para

provisionamento de serviços e a redução de despesas operacionais (OPEX) e de capital (CAPEX) [Cotroneo et al. 2014]. NFV utiliza técnicas de virtualização para disponibilizar serviços de rede através de dispositivos virtuais que executam em hardware genérico (*e.g.,* arquitetura x86). Dessa forma, a inclusão de novos serviços, bem como a alteração e atualização de serviços já existentes não necessita da instalação de equipamentos de hardware especializados [Mijumbi et al. 2016], diminuindo custos e flexibilizando o provisionamento de serviços.

Existem diversos desafios para a utilização efetiva da tecnologia NFV, incluindo gerenciamento e a garantia de desempenho das Funções Virtualizadas de Rede (*Virtualized Network Functions* ou VNFs). Neste sentido o *European Telecommunications Standards Institute* (ETSI) vem propondo especificações para a padronização da arquitetura NFV. Um dos principais componentes desta arquitetura é o *VNF Manager* (VNFM), entidade responsável pela gerência do ciclo de vida das VNFs. A gerência consiste principalmente da instanciação, remoção e atualização das VNFs. Existem ainda outras operações importantes, como o ajuste automático de recursos utilizados e a recuperação da VNFs em casos de falha.

Atualmente, existem diversas plataformas que implementam o VNFM [Tacker 2017, OpenBaton 2017, ETSI 2017, ONAP 2017, Rift.io 2017]. Em geral, estas plataformas demandam uma série de procedimentos para as operações do ciclo de vida das VNFs, que vão desde a criação de descritores de máquinas virtuais (*Virtual Machines* ou VMs) até a configuração manual do software que irá executar a função virtualizada. Dessa forma, tais soluções se tornam complexas e exigem do operador das funções um grande entendimento da infraestrutura do sistema [Shen et al. 2015]. Além disso, as plataformas disponíveis são pouco flexíveis, fazendo com que a integração com outras tecnologias exija um grande esforço dos desenvolvedores para adaptar as ferramentas utilizadas. Por fim, algumas destas soluções não gerenciam VNFs de maneira completa, sendo necessária a utilização de ferramentas adicionais ou gerenciamento manual das funções em nível de software.

Para suprir tais deficiências, neste trabalho é proposta a arquitetura de um VNFM que simplifica o gerenciamento do ciclo de vida das VNFs. O VNFM proposto define uma API que permite o controle do ciclo de vida das funções virtualizadas através de requisições feitas pelo usuário. Além disso, são definidas outras duas APIs que flexibilizam e simplificam a compatibilidade do VNFM com outras plataformas e cenários no ambiente NFV. Isso reduz o conhecimento necessário da infraestrutura virtualizada por parte do operador das funções e simplifica os procedimentos de gerência. É proposto também um submódulo do VNFM que utiliza todas as APIs definidas para controlar de maneira autônoma todo o ciclo de vida das VNFs, desde o nível de hardware até o nível de software.

Como prova de conceito da arquitetura proposta, é apresentado um protótipo, denominado de *vCommander*, que implementa a arquitetura do VNFM proposto. Em especial, é descrito como o *vCommander* utiliza o *Event Manager* e as APIs de comunicação para implementar a inserção, remoção e atualização das VNFs. Resultados obtidos em um ambiente experimental comprovam a efetividade do *vCommander* na realização de todas as operações de gerenciamento. Em especial, foi analisado o desempenho individual de cada operação.

Este trabalho está organizado da seguinte forma. A Seção 2 apresenta uma breve fundamentação sobre plataformas de gerência e orquestração de NFV. A Seção 3 apresenta a arquitetura conceitual da nossa proposta. A Seção 4 descreve a implementação do protótipo *vCommander* e resultados experimentais são apresentados na Seção 5. Por fim, a Seção 6 apresenta as conclusões e os trabalhos futuros.

2. Fundamentação e Trabalhos Relacionados

Nesta seção serão apresentados os principais conceitos e esforços relacionados ao controle, gerência e orquestração de funções virtualizadas de rede. Na Subseção 2.1, a arquitetura de referência proposta pelo ETSI para NFV é detalhada, com uma atenção especial ao bloco de gerência de VNFs. Na Subseção 2.2 diferentes plataformas que implementam o *NFV Management and Network Orchestration* e, mais especificamente, o módulo operacional de gerência (*i.e. VNF Manager*) são discutidas.

2.1. Arquitetura NFV

Como apresentado em [ETSI 2015], o paradigma NFV deve suportar a execução de funções virtualizadas de rede provenientes de diferentes desenvolvedores. Essas VNFs são coordenadas, gerenciadas e providas pelo NFV-MANO, que oferece interfaces de comunicação padronizadas e abstrai os recursos computacionais necessários para a execução das mesmas [ETSI 2014], como apresentado na Figura 1. A arquitetura NFV é composta por três blocos principais: *Virtualized Networks Functions* (VNFs) - elementos operacionais virtualizados responsáveis pela execução de funções de rede; *NFV Infrastrucuture* (NFVI) - abstração de recursos de armazenamento, computação e rede; *NFV Management and Orchestration* (NFV-MANO) - módulos operacionais de controle e orquestração de VNFs, responsáveis pelo ciclo de vida e gerenciamento de recursos das funções virtualizadas.

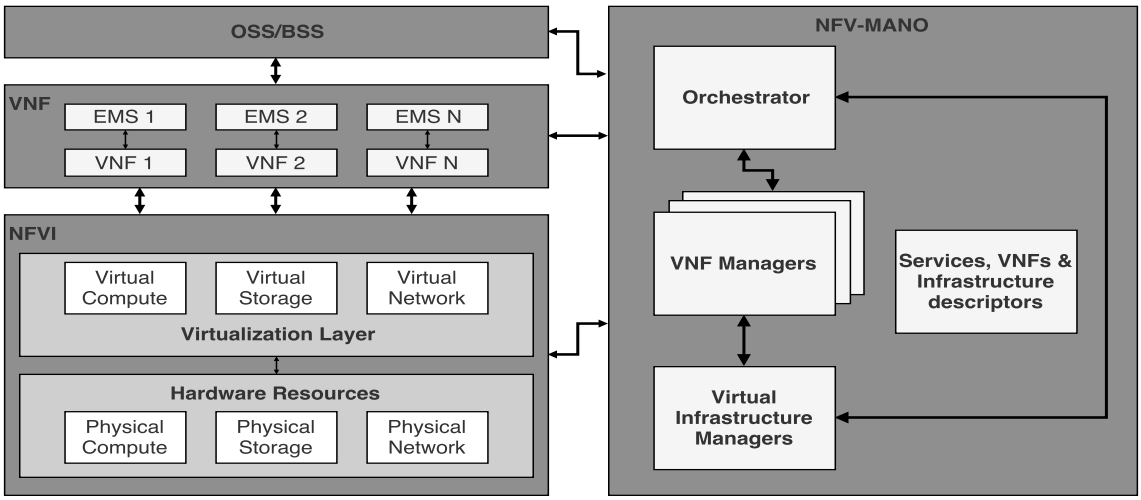

Figura 1. Arquitetura NFV.

O NFV-MANO também realiza a gerência dos recursos disponíveis na infraestrutura de virtualização (*Virtualized Infrastructure Manager* ou VIM) e das VNFs em operação (*VNF Manager* ou VNFM). Em especial, o VNFM é o módulo cuja a responsabilidade principal é realizar a gerência do ciclo de vida das VNFs [Bernardos et al. 2017].

As operações relacionadas ao VNFM devem ser suportadas por qualquer VNF, independe da sua implementação [ETSI 2014]. Entre as atribuições deste módulo estão: instanciação, configuração inicial, atualização e escalonamento das instâncias de funções virtualizadas. Finalmente, o VNFM deve ter acesso aos repositórios e às diferentes versões de VNFs, além de total conhecimento dos descritores das mesmas.

2.2. Trabalhos Relacionados

Diferentes soluções capazes de realizar funções destacadas no módulo de gerência e orquestração de VNFs, de acordo com as especificações ETSI, foram propostas [ETSI 2017, Tacker 2017, OpenBaton 2017, ONAP 2017, Rift.io 2017]. Apesar da diversidade de plataformas que implementam parcialmente responsabilidades do MANO, nenhuma é totalmente adequada ao conjunto de operações previstas para este bloco.

Tacker [Tacker 2017], por exemplo, é uma plataforma capaz de executar funções de VNFM e NFVO, abstraindo a utilização do VIM (*e.g. OpenStack* [OpenStack 2017]) com funções de mais alto nível. Entre as funções implementadas relacionadas ao VNFM estão o controle do ciclo de vida básico, monitoramento, escalonamento e simplificação da configuração inicial das VNFs. O módulo que atua como NFVO é capaz de aplicar políticas de instanciação e escalonamento, bem como manipular estruturas complexas de serviços de rede, descritas através de *Service Function Chains* (SFCs) [Halpern and Pignataro 2015]. Apesar de implementar grande parte das funções previstas para o MANO, o *Tacker* não conta com uma interface gráfica padronizada, módulos de simulação e nem mesmo com a possibilidade de implantação de algoritmos de *placement* otimizados [Mechtri et al. 2017].

Open Source MANO (OSM) [ETSI 2017] consiste da implementação dos três módulos operacionais do NFV-MANO (*i.e.,* VNFO, VNFM e VIM) para realizar a configuração e abstração de VNFs, orquestração de recursos e elementos para controle da infraestrutura. Entre as operações suportadas pelo OSM estão um serviço de orquestração responsável por controlar diversos aspectos do ciclo de vida das VNFs e possibilitando a execução de serviços complexos, suporte a diferentes VIMs, controladores SDN e ferramentas de monitoramento. Além disso, conta com um VNFM genérico que possibilita a integração a VNFMs específicos, porém, exigindo modificações no seu orquestrador de serviço para total compatibilidade [Dmitriy Andrushko 2017].

A plataforma *Open Baton* [OpenBaton 2017] também implementa um MANO baseado na especificação do ETSI. A plataforma consiste de um VNFO e um VNFM e possui suporte nativo a diferentes VIMs. Além do monitoramento e controle de ciclo de vida das VMs que executam as funções de rede, o *Open Baton* utiliza rotinas definidas através de um VNF *package* com intuito de configurar a função de rede propriamente dita, ou seja, é capaz de alterar internamente o estado do *software* executado pela VM. Apesar de ser capaz de suportar VIMs heterogêneos e apresentar uma interface de usuário amigável, o *Open Baton* não apresenta nenhum suporte nativo para instanciação e gestão de SFCs [Mechtri et al. 2017].

De maneira geral, as plataformas existentes implementam funções destinadas ao VNFM e NFVO, além de suportarem a utilização de VIMs externos (*e.g. OpenStack, OpenVIM* [OpenVIM 2017]). Entretanto, algumas funcionalidades destacadas para esses blocos operacionais de orquestração e gerência das VNFs, como a possibilidade de reali-

zar alterações internas a VNF (*i.e.* na função de rede em si) ou a aplicação e verificação de políticas complexas, não contam com suporte nativo.

3. Uma Arquitetura para a Gerência do Ciclo de Vida das VNFs

Nesta seção é apresentada a arquitetura conceitual e são detalhados os principais componentes do *VNF Manager* (VNFM) proposto. Em seguida, são descritas as interfaces necessárias para a comunicação entre os módulos da solução. Por fim, funcionalidades de cada uma das APIs propostas neste trabalho são detalhadas.

3.1. Componentes da Arquitetura

Este trabalho introduz uma arquitetura do módulo VNFM presente no componente de gerenciamento e orquestração de funções virtualizadas de rede. Foram definidas três APIs que são utilizadas para gerenciar o ciclo de vida das VNFs, fornecendo maior flexibilidade para o desenvolvimento de um VNFM. Em especial, é proposto um submódulo do VNFM, denominado de *Event Manager*, responsável por interagir com estas APIs. Os requisitos de cada uma das APIs definidas são descritas na Subseção 3.2.

O módulo VNFM é um subsistema de gerência responsável por controlar o ciclo de vida das VNFs. O VNFM permite a instanciação, remoção e atualização das máquinas virtuais que hospedam as VNFs, bem como a gerência e monitoramento das VNFs em nível de função, isto é, configuração e execução do software que executa a funcionalidade da VNF. O VNFM proposto neste trabalho é capaz de gerenciar as VNFs instanciadas independentemente dos seus propósitos (*e.g.,* segurança, desempenho e encaminhamento).

Em geral, as plataformas que oferecem mecanismos para gerenciar o ciclo de vida de VNFs demandam uma série de procedimentos exaustivos, como por exemplo, a criação e remoção de descritores de máquinas virtuais, configuração e instanciação em nível de software da VNF. Além disso, é exigida do usuário uma grande quantidade de informações, referentes ao funcionamento do sistema [Shen et al. 2015]. Estas informações incluem detalhes de configuração da rede até parâmetros específicos do sistema. Neste sentido, o VNFM proposto utiliza uma API que permite realizar a gerência das VNFs através de requisições geradas pelo usuário. Como exemplo, considere a instanciação de uma VNF. Para realizar essa tarefa em outras plataformas é comum a necessidade de executar uma série de comandos que envolvem o registro de descritores (*e.g., VNF Descriptor* ou *Network Service Descriptor*) e a instanciação da máquina virtual onde a função irá ser executada. Ainda, algumas plataformas não instanciam a função da VNF, sendo necessária a execução manual ou o desenvolvimento de ferramentas adicionais que realizem a execução da função de rede. Ao utilizar o VNFM proposto, uma única requisição é suficiente para que a VNF solicitada seja instanciada e executada.

A arquitetura proposta é apresentada na Figura 2. Esta arquitetura consiste de duas camadas: a camada de usuário (*User Layer*) e a camada NFV (*NFV Environment*). Na camada de usuário, o operador se comunica com módulos providos por plataformas genéricas que ofereçam serviços em nível de usuário (*e.g.,* Marketplaces [Xilouris et al. 2014] e ferramentas para visualização em NFV [Franco et al. 2016]). Tais plataformas se comunicam com os módulos disponíveis no NFV-MANO através de uma API de comunicação e podem se beneficiar dos recursos disponíveis, tais como velocidade

e facilidade na instanciação de VNFs, monitoramento da operação das VNFs e gerenciamento eficaz de recursos. O *NFV Environment* consiste de três componentes principais: NFV-MANO (*NFV MANagement and Orchestration*), NFVI (*NFV Infrastructure*) e *VNF Instances*. O componente NFV-MANO é responsável por gerenciar e orquestrar as funções virtualizadas de rede. O NFVI por sua vez representa a infraestrutura virtualizada onde as VNFs irão executar. Nesta infraestrutura estão presentes os recursos de processamento (*compute*), armazenamento (*storage*) e de rede (*network*). Por fim, as *VNF-Instances* representam as instâncias das funções virtualizadas de rede.

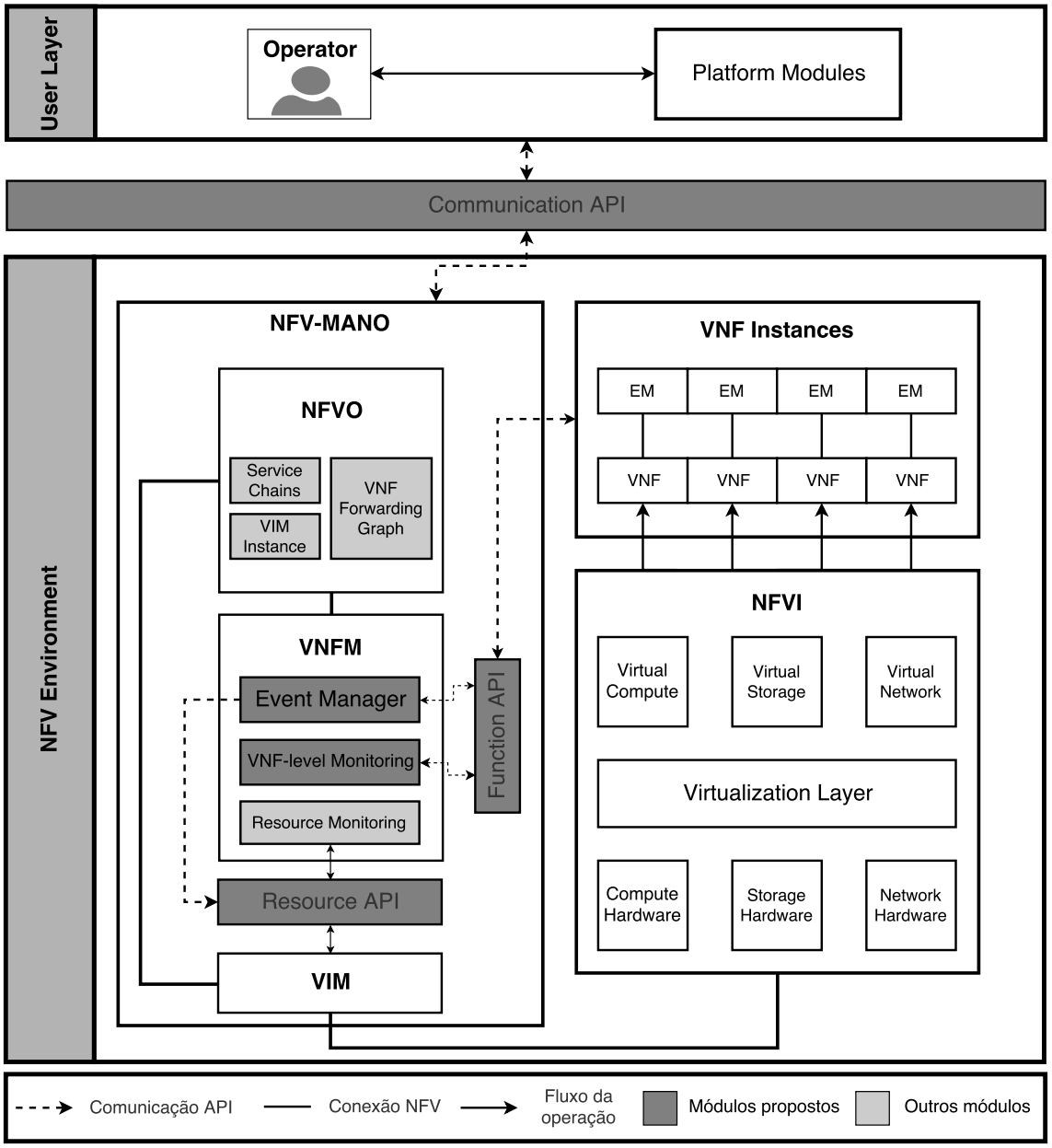

Figura 2. Arquitetura onde o VNFM está inserido.

Para prover a automação necessária foi implementado um módulo principal denominado *Event Manager* e um módulo de monitoramento denominado *VNF-level Monitoring*. O *Event Manager* é o módulo principal do VNFM responsável por receber di-

retamente as requisições originadas da camada de usuário. A partir destas requisições, o VNFM realiza, de maneira automática, todos os procedimentos necessários para a instanciação da VNF, desde a criação da VM até a execução da função. Os detalhes das funcionalidades da gerência fornecidas pelo *Event Manager* são descritos na Subseção 4.1. Para o monitoramento das funções das VNFs, um módulo especializado é introduzido. O *VNF-level Monitoring* é responsável por monitorar a função propriamente dita de cada VNF. Neste caso, as métricas coletadas são relacionadas ao uso da função, como por exemplo, a quantidade de pacotes processados e a latência das operações. Por fim, o módulo *Resource Monitoring* monitora as máquinas virtuais de cada VNF instanciada. Neste caso, as métricas de monitoramento são relacionadas aos recursos utilizados de cada VNF, tais como uso de CPU, memória e disco. Uma vez obtidas estas métricas é possível realizar rotinas para o gerenciamento das VNFs (*e.g.,* redistribuição de recursos sob demanda e recuperação de falhas).

3.2. Interfaces de Comunicação

A arquitetura proposta descrita na Subseção 3.1 utiliza três APIs para simplificar e flexibilizar as operações das funções virtualizadas. Uma destas APIs é a *Communication API*, a qual o VNFM utiliza para receber requisições da camada de usuário. As outras duas APIs são denominadas de *Resource* e *Function API* e são utilizadas para gerenciar o ciclo de vida das VNFs tanto em nível de hardware quanto em nível de software. Cada uma destas APIs são descritas a seguir.

Com o objetivo de facilitar a interação do usuário com o controle do ciclo de vida das VNFs, a *Communication API* define um conjunto de funções que permite ao usuário realizar operações sobre as funções virtualizadas. Estas funções não exigem do usuário a execução de vários procedimentos e evitam a necessidade de especificar múltiplos parâmetros do sistema. Todas as funcionalidades disponibilizadas pela *Communication API* estão descritas na Tabela 1.

Tabela 1. Funcionalidades da *Communication API*.

Função	Descrição
vnf_create(vnfd, vnf_function)	Cria uma VNF
vnf_delete(vnf_id)	Remove uma VNF
vnf_update(vnf_id, vnfd)	Atualiza os recursos de uma VNF
vnf_function_update(vnf_id, vnf_function)	Atualiza a função de uma VNF
sfc_create(vnffgd)	Cria um *Service Function Chaining*

Uma vez recebidas requisições pela *Communication API* da camada de usuário, o VNFM comunica-se com vários outros módulos que integram a arquitetura NFV definida pela ETSI. Um destes módulos é o VIM (*Virtualized Infrastructure Manager*), subsistema que gerencia os recursos e controla as interações da VNF com o ambiente de virtualização. Já os EMs (*Element Managers*), são responsáveis pelo FCAPS (*Fault management*, *Configuration*, *Accounting*, *Performance measurement* e *Security*) das VNFs [ETSI 2014].

As comunicações entre o VNFM e os demais módulos durante o controle do ciclo de vida das VNFs são realizadas por meio de requisições REST (*REpresentational State Transfer*) ou através de um *Message Broker* (*e.g., RabbitMQ*). Em geral, estas implementações são pouco flexíveis, onde cada plataforma de gerência de VNFs

fornece uma solução que implementa o conjunto de funcionalidades necessárias para comunicação do VNFM com estes módulos. Dessa maneira, utilizar outras tecnologias torna-se uma tarefa complexa, exigindo um grande esforço dos desenvolvedores para adaptar as ferramentas utilizadas. Nesse sentido, propomos duas APIs para simplificar a compatibilidade e flexibilidade do VNFM. Estas APIs são denominadas de *Resource API* e *Function API* e são descritas a seguir.

A *Resource API* define as funcionalidades necessárias para que o VNFM possa gerenciar os recursos das máquinas virtuais que irão hospedar as VNFs. A Tabela 2 descreve o conjunto mínimo de funções que deve ser implementado nesta API. Novas funções podem ser adicionadas à *Resource API* (*e.g.,* remoção e atualização de VNFD), conforme a necessidade do operador. Em geral, esta API é fornecida através de um *plugin* ou *driver* do próprio VIM.

Tabela 2. Conjunto de funcionalidades da *Resource API*.

Função	**Descrição**
vnfd_create(vnfd)	Cria um *VNF Descriptor*
vm_create(vnfd_id)	Cria uma máquina virtual baseada no VNFD
vm_delete(vnf_id)	Remove uma máquina virtual
vm_update(vnf_id, vnfd)	Atualiza uma máquina virtual

A *Function API* por sua vez define as funcionalidades que o VNFM deve executar em nível de função. Esta API é responsável por administrar o FCAPS das VNFs, fazendo o papel de EM. A Tabela 3 descreve as funções que devem ser implementadas nesta API. Da mesma forma como na *Resource API*, funções adicionais podem ser incluídas na *Function API*. Em especial, através dessa API pode-se definir métricas customizadas de monitoramento obtidas pelo módulo *VNF-level Monitoring*, como por exemplo, número de pacotes bloqueados em um *firewall*. Em geral, as próprias VNFs oferecem esta interface para que o VNFM possa coletar dados e executar instruções.

Tabela 3. Conjunto de funcionalidades da *Function API*.

Função	**Descrição**
write_function(vnf_ip, function)	Insere a função da VNF na máquina virtual
start_function(vnf_ip)	Inicia a função da VNF
stop_function(vnf_ip)	Para a função
get_log(vnf_ip)	Obtém o *log* da função
get_metrics(vnf_ip)	Obtém métricas do uso da função

Uma vez definidas as APIs de comunicação necessárias para gerenciamento do ciclo de vida das VNFs, é possível que diferentes tecnologias sejam utilizadas para implementar tais APIs. Com isso, reduz-se a complexidade para adoção de novas funções e também se facilita a implementação de novos serviços e rotinas em diferentes infraestruturas. Baseando-se na arquitetura proposta, na seção a seguir é apresentada e detalhada a implementação de uma prova de conceito, denominada *vCommander*.

4. *vCommander*: Implementação Prototípica de um *VNF Manager*

Nesta seção é apresentado o *vCommander*, um VNFM implementado como prova de conceito da arquitetura proposta na Seção 3. Na Subseção 4.1 são detalhadas as operações de criação, remoção e atualização de uma VNF. Em seguida, detalhes do protótipo *vCommander* implementado são descritos na Subseção 4.2.

4.1. Gerenciando o Ciclo de Vida das VNFs

A partir da *Resource API* e *Function API* é possível realizar as operações de gerência do ciclo de vida das VNFs, desde a criação da máquina virtual, até a instanciação da função propriamente dita. A seguir é descrito como o *vCommander* utiliza estas APIs para gerenciar as VNFs, detalhando as operações de instanciação, remoção e atualização, as quais são iniciadas a partir de requisições recebidas pela *Communication API*. É importante notar que todos os procedimentos descritos são realizados de maneira automática, sem a intervenção do usuário.

4.1.1. Instanciação de uma VNF

Para a instanciação de uma VNF são necessárias duas etapas: *(i)* criação e configuração da máquina virtual (*Resource API*) e *(ii)* execução da função virtualizada (*Function API*). A primeira etapa consiste de dois passos, onde o primeiro é utilizar a função *vnfd_create(vnfd)* para adicionar o VNFD recebido da *Communication API* para o catálogo de VNFs, gerando um identificador único denominado *vnfd_id*. O segundo passo é a chamada da função *vm_create(vnfd_id)* para criação da máquina virtual baseando-se no VNFD criado anteriormente. Dado que o tempo de criação varia para cada sistema, o *vCommander* inicia um mecanismo de *polling*, onde ele verifica a cada intervalo de tempo se a máquina virtual está ativa. O intervalo de tempo padrão de *polling* do *vCommander* é de um segundo, porém este valor é customizável e pode ser ajustado de acordo com o sistema. Quando o *vCommander* detecta que a criação da máquina virtual está pronta, inicia-se a segunda etapa.

A segunda etapa consiste em inserir a função da VNF na máquina virtual, também recebida como parâmetro da *Communication API*, através da chamada *write_function(vnf_ip, function)*. Uma vez inserida, a função da VNF é executada através da função *start_function(vnf_ip)* e neste momento está totalmente funcional. Por fim, o *vCommander* adiciona esta VNF ao catálogo de instâncias de VNFs para que ela possa ser monitorada pelos módulos *Resource Monitoring* e *VNF-level Monitoring*. Para cada operação realizada é verificado se ela foi concluída com sucesso, caso contrário, um mecanismo de *rollback* desfaz as modificações realizadas até o momento.

4.1.2. Remoção de uma VNF

O processo de remoção inicia-se quando o *vCommander* recebe um identificador único da VNF (*vnf_id*) pela *Communication API* e posteriormente remove a máquina virtual correspondente através da função *vm_delete(vnf_id)*. Da mesma forma como na instanciação, um mecanismo de *polling* é iniciado para verificar, a cada intervalo de tempo, se a máquina virtual foi removida com sucesso. Em seguida, a VNF é removida do catálogo de VNFs

instanciadas e não é mais monitorada pelos módulos de monitoramento. Por fim, o VNFD da VNF é removido do catálogo.

4.1.3. Atualização de uma VNF

A atualização de uma VNF pode ser dividida em dois níveis: *(i)* atualização de recursos (*Resource API*) e *(ii)* atualização da função virtualizada (*Function API*). No caso da atualização de recursos, a máquina virtual que hospeda a VNF pode utilizar a função *vm_update(vnf_id, vnfd)* para atualizar, por exemplo, o número de CPUs, a quantidade de memória RAM e quantidade de disco alocados para a VNF. Essa atualização ocorre da seguinte forma. Pela *Communication API*, é recebido o *vnf_id* e o novo descritor da VNF, contendo as modificações que devem ser realizadas. Por exemplo, este descritor pode modificar a quantidade de memória RAM para 512 MB ou alterar o número de CPUs para 8. O *vCommander* executa então a operação de atualização através da *Resource API* e, para concluir as alterações, a máquina virtual é reiniciada.

Para a atualização da função da VNF, o *vCommander* recebe o *vnf_id* e a nova implementação da função virtualizada. Da mesma forma como na instanciação, a nova função é inserida na máquina virtual pela função *write_function(vnf_ip, function)*. Após, a função antiga é parada através da chamada *stop_function(vnf_ip)* e a nova função é executada pela função *start_function(vnf_ip)*. Note que neste intervalo de tempo é possível que pacotes sejam perdidos.

4.2. Protótipo

Como descrito na Seção 3, a arquitetura do VNFM proposto consiste dos módulos *Event Manager* e *VNF-level Monitoring* e utiliza as APIs *Communication*, *Resource* e *Function* para a comunicação entre os demais componentes. O *Event Manager* foi desenvolvido em *Python* e é responsável por receber requisições de alto nível da *Communication API* e realizar as operações necessárias através da *Resource* e *Function API*. Todas estas requisições são feitas por meio de de requisições REST utilizando a biblioteca *Requests*[1] e as informações são definidas em JSON. É importante notar que a comunicação entre os módulos pode ser interna (*i.e.,* execução centralizada dos componentes) ou externa (*i.e.,* execução distribuída dos componentes).

O *VNF-level Monitoring* por sua vez, realiza requisições às VNFs a cada intervalo de tempo. O *vCommander* utiliza por padrão um intervalo de tempo de 30 segundos, porém este valor pode ser customizado de acordo com a necessidade. Os dados coletados são enviados para um servidor central, onde é feito o processamento destes dados. Os dados são inseridos no banco de dados *InfluxDB*[2], que realiza em tempo real o processamento dos dados coletados do uso das funções virtualizadas de rede. Ambos os módulos foram também desenvolvidos em *Python*.

Para a *Resource API*, todas as funcionalidades descritas na Tabela 2 foram implementadas. Esta API deve ser disponibilizada para que o VNFM possa realizar as requisições e gerenciar as máquinas virtuais. Da mesma forma, a *Function API* implementa as funcionalidades descritas na Tabela 3. Em especial, a implementação desta API

[1]http://docs.python-requests.org
[2]https://www.influxdata.com/

foi feita de maneira genérica, de forma a utilizar a mesma API para diferentes VNFs. Isso é feito passando o endereço IP da VNF como parâmetro da requisição. Neste trabalho, a *Resource API* foi implementada utilizando o *Tacker* [Tacker 2017]. Para a *Function API*, cada VNF instanciada disponibiliza uma interface REST onde o *vCommander* realiza as requisições. A próxima seção apresenta resultados experimentais do *vCommander* avaliando o tempo e a quantidade de recursos utilizados por cada operação.

5. Resultados Experimentais

Com o objetivo de verificar o comportamento e o desempenho do protótipo *vCommander*, esta seção apresenta resultados experimentais. Na Subseção 5.1, o cenário utilizado para os experimentos é apresentado. A seguir, na Subseção 5.2, as justificativas para as métricas escolhidas e os resultados obtidos são apresentados e discutidos.

5.1. Cenário de Avaliação

Como descrito na Seção 4 o *vCommander* comunica-se com a *Resource API* para gerenciar os recursos virtuais das VNFs. No protótipo implementado a *Resource API* é disponibilizada pelo Tacker e OpenStack, e sua execução depende de um servidor de virtualização que esteja executando tais serviços. Assim, foi utilizado um servidor Dell PowerEdge com um processador Intel(R) Xeon(R) E3-1220v6 @ 3.00GHz, com 4 núcleos e 8 *threads* e memória de 8GB DDR4 @ 2400Mhz. Este servidor foi configurado com a distribuição Ubuntu 16.04, a versão Pike do OpenStack, bem como o protótipo desenvolvido – embora este possa ser executado remotamente.

O protótipo possui 4 funções principais que são disponibilizadas ao usuário: *Create*, *Delete*, *Update* e *Update Function*. Cada operação executa diversas suboperações que são efetivamente responsáveis por se comunicar com o Tacker para controlar a infraestrutura virtualizada.

A avaliação do protótipo foi realizada da seguinte forma: a operação *Create* foi executada instanciando uma VNF do tipo Click-on-OSv [da Cruz Marcuzzo et al. 2017] com um *firewall*, e recursos de 512 MB de RAM, 1 CPU e 1 GB de disco. A seguir, a operação *Update* foi executada, alterando a memória RAM alocada para a VNF de 512 MB para 1 GB, enquanto os demais recursos permaneceram iguais. Após, a operação *Update Function* foi avaliada, alterando a função em execução para um VNF *forwarder*. Por fim, a VNF foi removida utilizando a operação *Delete*. Todos os testes foram realizados 30 vezes e os resultados são apresentados com um intervalo de confiança de 95%.

5.2. Resultados e Discussão

Visto que as VNFs devem ser capazes de serem instanciadas, configuradas e escaladas dinamicamente, o tempo de execução destas funções é uma métrica relevante para medição, uma vez que o atraso na execução destas funções pode impactar negativamente o funcionamento da infraestrutura. Desta forma, o protótipo foi instrumentado de forma a obter o tempo de execução total de cada uma das operações, bem como o tempo de execução das diversas suboperações que as compõem.

Como pode ser observado na Figura 3, a operação *Create* foi a operação com maior duração, sendo que a suboperação de *polling* consome por volta de 90% do

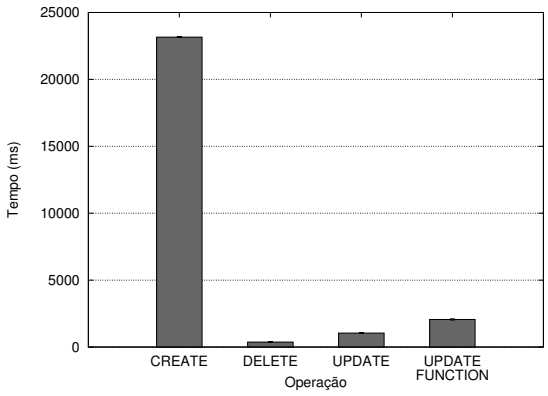

Figura 3. Tempo total de execução das operações.

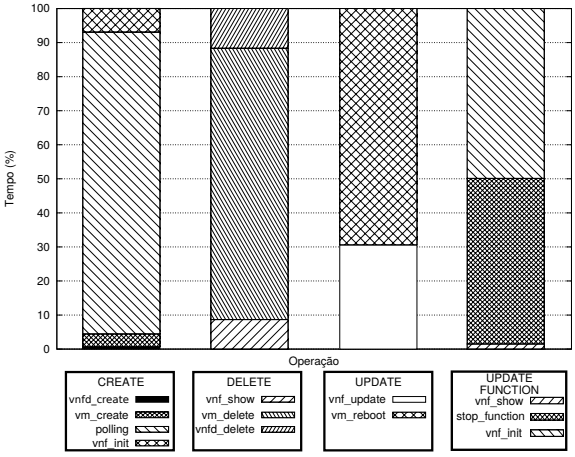

Figura 4. Tempo de execução das suboperações em porcentagem.

tempo total de execução da operação, conforme Figura 4. Isto ocorre pois, ao enviar a requisição para a criação da VM (suboperação *vm_create*), a suboperação *polling* deve verificar periodicamente e aguardar a infraestrutura terminar o processo de instanciação, para que a função de rede (configuração Click) possa ser configurada na próxima suboperação(*vnf_init*). Já a operação *Update Function* precisa realizar o *upload* de uma nova função de rede (uma nova configuração Click) e aguardar a reinicialização da VNF. Por fim, a operação *Update*, após atualizar a descrição da VM, deve aguardar sua reinicialização, enquanto a operação *Delete* envia comandos para que a VM seja removida.

O uso de NFV também depende da utilização de servidores de virtualização de alta densidade, onde pode ocorrer uma escassez de recursos computacionais se houverem muitas VNFs instanciadas. Assim, o consumo de recursos do protótipo deve ser reduzido, considerando que a própria plataforma OpenStack já possui requisitos elevados para sua execução. Desta forma, tanto o consumo de CPU como o consumo de memória do protótipo foram medidos, utilizando a biblioteca *psutil* do *Python*. A Figura 5 apresenta a utilização de memória e CPU de cada operação.

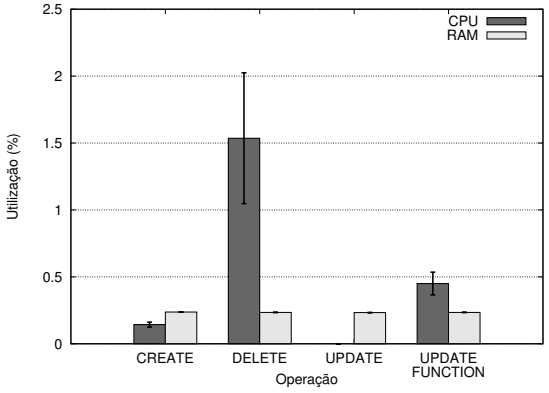

Figura 5. Uso de CPU e memória durante as operações.

Com relação a memória RAM, a memória utilizada é ocupada pelas bibliotecas

desenvolvidas para a comunicação com a *Resource API* e *Function API* e a gerência do ciclo de vida das VNFs. Como estas bibliotecas são utilizadas por todas as operações, o consumo de memória é semelhante.

Já para a utilização de CPU, é possível perceber que a operação *Delete* apresenta a maior utilização de CPU, enquanto que as operações *Create* e *Update Function*, mesmo possuindo suboperações mais longas, apresentam um menor consumo de CPU. Isto deve-se ao fato de que, no caso da operação *Create*, apesar da suboperação de *polling* ser a mais longa, ela apenas aguarda a resposta da *Resource API*, não realizando nenhum processamento neste intervalo. Já a operação *Update Function* precisa aguardar a VNF reinicializar para finalizar. Desta forma, como a operação *Delete* não é *I/O bound*, não é necessário aguardar nenhum tipo de chamada do sistema, portanto está realizando algum tipo de processamento durante todo o tempo da sua execução. Por fim, a operação *Update* não apresenta um consumo de CPU significativo, já que ela apenas envia o VNFD atualizado para a *Resource API* e aguarda a reinicialização, sem realizar nenhum tipo de processamento.

Apesar disso, o maior consumo, dentre todas as operações, foi de apenas 1.5% de uso da CPU durante uma duração menor que 1 segundo, enquanto que o consumo de memória RAM corresponde a 0.25% da memória disponível no ambiente testado, de forma que o impacto do protótipo nos recursos do servidor seja mínimo.

6. Conclusão

Neste trabalho foi introduzida uma arquitetura de um VNFM para simplificar o gerenciamento do ciclo de vida das VNFs. Através da utilização de APIs, a arquitetura permite compatibilidade entre diferentes plataformas além de simplificar a gerência do ciclo de vida das VNFs. Além disso, o gerenciamento das funções virtualizadas é feito tanto em nível de hardware quanto em nível de software.

Como prova de conceito foi implementado um protótipo da arquitetura proposta denominado de *vCommander*. Resultados experimentais demonstram que o *vCommander* consegue controlar e facilitar o gerenciamento do ciclo de vida das VNFs consumindo uma quantidade de recursos mínima, utilizando em média menos de 1% de CPU e 0.25% de memória RAM. Além disso, o tempo de operação das principais funcionalidades do *vCommander* foi medido e pode-se observar a efetividade da arquitetura.

Para trabalhos futuros planeja-se estender as funcionalidades da arquitetura proposta para incluir a composição e orquestração de VNFs. Além disso, planeja-se oferecer uma estratégia inteligente de migração de VNFs (*e.g., live migration*) com o objetivo de minimizar a quantidade de pacotes perdidos.

Agradecimentos

Este trabalho foi realizado dentro do projeto GT-FENDE da RNP.

Referências

Bernardos, C. J., Rahman, A., Zúñiga, J.-C., Contreras, L. M., Aranda, P. A., and Lynch, P. (2017). Network Virtualization Research Challenges. Internet-Draft draft-irtf-nfvrg-gaps-network-virtualization-07, Internet Engineering Task Force. Work in Progress.

Cotroneo, D., De Simone, L., Iannillo, A., Lanzaro, A., Natella, R., Fan, J., and Ping, W. (2014). Network function virtualization: Challenges and directions for reliability assurance. In *Software Reliability Engineering Workshops (ISSREW), 2014 IEEE International Symposium on*, pages 37–42. IEEE.

da Cruz Marcuzzo, L., Garcia, V. F., Cunha, V., Corujo, D., Barraca, J. P., Aguiar, R. L., Schaeffer-Filho, A. E., Granville, L. Z., and dos Santos, C. R. (2017). Click-on-osv: A platform for running click-based middleboxes. In *Integrated Network and Service Management (IM), 2017 IFIP/IEEE Symposium on*, pages 885–886. IEEE.

Dmitriy Andrushko, G. E. (2017). What is the best nfv orchestration platform? a review of osm, open-o, cord, and cloudify. `https://www.mirantis.com/blog/which-nfv-orchestration-platform-best-review-osm-open-o-cord-cloudify/`. Accessed: 2017-12-14.

ETSI (2017). Open source mano. `https://osm.etsi.org/`. Accessed: 2017-11-21.

ETSI, N. (2014). Network functions virtualisation (nfv); management and orchestration. *NFV-MAN*, 1:v0.

ETSI, N. (2015). Network functions virtualization (nfv) infrastructure overview. *NFV-INF*, 1:V1.

Franco, M. F., d. Santos, R. L., Schaeffer-Filho, A., and Granville, L. Z. (2016). Vision – interactive and selective visualization for management of nfv-enabled networks. In *2016 IEEE 30th International Conference on Advanced Information Networking and Applications (AINA)*, pages 274–281.

Halpern, J. and Pignataro, C. (2015). Service function chaining (sfc) architecture.

Mechtri, M., Ghribi, C., Soualah, O., and Zeghlache, D. (2017). Nfv orchestration framework addressing sfc challenges. *IEEE Communications Magazine*, 55(6):16–23.

Mijumbi, R., Serrat, J., Gorricho, J.-L., Bouten, N., De Turck, F., and Boutaba, R. (2016). Network function virtualization: State-of-the-art and research challenges. *IEEE Communications Surveys & Tutorials*, 18(1):236–262.

ONAP (2017). Onap. `https://www.onap.org/`. Accessed: 2017-11-21.

OpenBaton (2017). Openbaton. `https://openbaton.github.io/`. Accessed: 2017-11-21.

OpenStack (2017). Openstack. `https://www.openstack.org/`. Accessed: 2017-11-24.

OpenVIM (2017). Openvim. `https://www.sdxcentral.com/projects/openvim/`. Accessed: 2017-11-24.

Rift.io (2017). Rift.io. `https://riftio.com/`. Accessed: 2017-11-21.

Shen, W., Yoshida, M., Minato, K., and Imajuku, W. (2015). vconductor: An enabler for achieving virtual network integration as a service. *IEEE Communications Magazine*, 53(2):116–124.

Tacker (2017). Tacker. `https://wiki.openstack.org/wiki/Tacker`. Accessed: 2017-11-21.

Xilouris, G., Trouva, E., Lobillo, F., Soares, J. M., Carapinha, J., McGrath, M. J., Gardikis, G., Paglierani, P., Pallis, E., Zuccaro, L., Rebahi, Y., and Kourtis, A. (2014). T-nova: A marketplace for virtualized network functions. In *2014 European Conference on Networks and Communications (EuCNC)*, pages 1–5.