NFV*inc*: Disponibilizando Serviços Inovadores na Rede com Tecnologia NFV

Giovanni Venâncio[1], **Rogério C. Turchetti**[2], **Elias P. Duarte Jr.**[1]

[1]Departamento de Informática – Universidade Federal do Paraná (UFPR)

[2]CTISM - Universidade Federal de Santa Maria (UFSM)

gvsouza@inf.ufpr.br, turchetti@redes.ufsm.br, elias@inf.ufpr.br

Abstract. *Network Function Virtualization (NFV) allows the replacement by software of middleboxes traditionally implemented as specialized hardware. Although most VNFs implement classic middlebox functionalities, such as firewalls or intrusion detection, NFV technology can be used to provide services offered by the network itself that are employed by end-user applications. In this way, applications and systems normally executed by users on edge hosts, are now fully executed within the network, in the paradigm that has been called In-Network Computing (INC). We argue that using NFV to implement INC services has several advantages such as flexibility and cost. In this work we describe NFVinc, an architecture that allows developers to make available both their INC services and APIs that allow user applications to access those services. NFVinc is compliant with the popular NFV-MANO reference architecture proposed by the ETSI. Three case studies are presented demonstrating the flexibility and effectiveness of the NFVinc architecture.*

Resumo. *A Virtualização de Funções de Rede (Network Function Virtualization - NFV) permite a substituição de funções de rede tradicionalmente implementadas em hardware dedicado por software. Apesar de que a maioria das VNFs implementa funcionalidades de middleboxes clássicas, como firewall ou detector de intrusão, a tecnologia NFV pode ser utilizada para disponibilizar serviços oferecidos pela própria rede que são utilizados por aplicações de usuários finais. Desta forma, aplicações e sistemas normalmente executados pelos usuários em hosts da borda, passam a ser inteiramente executados dentro da rede, no paradigma que tem sido chamado de In-Network Computing (INC). Implementar INC em NFV tem diversas vantagens, incluindo aumento da flexibilidade e diminuição de custos. Neste trabalho descrevemos a arquitetura NFVinc que permite, de forma compatível com a arquitetura de referência NFV-MANO, a disponibilização tanto de serviços INC pelos desenvolvedores como a exportação de APIs que são usadas por aplicações de usuários finais para o acesso a estes serviços INC. São apresentados três estudos de caso demonstrando a flexibilidade e eficácia da arquitetura NFVinc.*

1. Introdução

A virtualização tem revolucionado as redes de computadores, permitindo sua evolução para múltiplas direções. Uma das tecnologias que mais tem se destacado neste contexto é a NFV (*Network Function Virtualization*). Utilizando NFV é possível disponibilizar novas

funcionalidades através de serviços de rede implementados com Funções Virtualizadas de Rede (*Virtualized Network Functions* – VNFs), que são executadas em hardware genérico [Mijumbi et al. 2016]. As vantagens em comparação com alternativas clássicas são muitas: maior flexibilidade e facilidade de gerenciamento, diminuição de custos e economia de energia. Apesar de que a maioria das VNFs implementa funcionalidades de *middleboxes* clássicas, como *firewall* ou detector de intrusão, a tecnologia NFV pode ser utilizada para disponibilizar serviços inovadores oferecidos pela própria rede. Desta forma, serviços normalmente executados em *hosts* da borda, passam a ser inteiramente executados dentro da rede, no paradigma que tem sido chamado de *In-Network Computing* (INC). O foco deste trabalho está na sinergia resultante da união das tecnologias NFV e INC, que chamamos de NFV*inc*. Diversos esforços já foram realizados para disponibilizar serviços na rede com NFV*inc*. Ainda em 2015 [Turchetti and Duarte 2015], foi implementado um serviço NFV*inc* baseado nos detectores de falhas clássicos de sistemas distribuídos; o trabalho foi expandido mais tarde [Turchetti and Duarte Jr 2017]. Outro serviço distribuído clássico implementado com NFV*inc* foi o consenso, utilizado para manter a consistência de um grupo de controladores SDN (*Software-Defined Networks*) [Venâncio et al. 2019]. Por fim, o trabalho [Venâncio et al. 2019] permite que a rede ofereça nativamente serviços de difusão confiável e ordenada de mensagens.

O uso de hardware programável é outra estratégia que tem sido utilizada para INC [Tokusashi et al. 2019]. Soulé e outros [Dang et al. 2016, Dang et al. 2020] implementaram o algoritmo de consenso Paxos em um *switch* utilizando P4 [Bosshart et al. 2014]. Outros serviços INC implementados em hardware programável incluem sistemas de armazenamento de dados na rede [Bressana et al. 2020]; um sistema de cache baseado em INC, chamado IncBricks [Liu et al. 2017]; aplicações como MapReduce [Dean and Ghemawat 2008], baseadas no paradigma de partição/agregação de dados [Sapio et al. 2017]. Estes serviços foram implementados utilizando tecnologias como ASICs (*Application Specific Integrated Circuits*) e FPGA (*Field Programmable Gate Array*). Em comparação com NFV*inc*, as implementações em hardware tem certamente níveis de desempenho maiores. Entretanto, acreditamos que as duas abordagens atendem a nichos distintos. Os serviços candidatos a serem implementados com NFV*inc* são aqueles já implementados em software na borda da rede.

As vantagens da tecnologia NFV*inc* são muitas. Talvez a principal seja a facilidade de implementar e gerenciar novos serviços usando tecnologias de virtualização em comparação com a alternativa de realizar a mesma implementação em hardware. Nesse sentido, outra vantagem da NFV*inc* é a diminuição do CAPEX (*CAPital EXpenditures*) e OPEX (*OPerational EXpenditures*), uma vez que ela é baseada em virtualização, portanto executada como software em hardware de prateleira. Do ponto de vista do operador da rede, a principal vantagem é a flexibilidade. Basta imaginar, por exemplo, a facilidade de fazer mudança de versão de um serviço implementado em software versus mudar versão de hardware. Para oferecer um novo serviço de rede basta fazer o download a partir de um *marketplace* da Internet e iniciar o serviço. Para deixar de oferecer o serviço, basta encerrar os processos correspondentes. Economia de energia, de espaço, facilidade de aumentar/reduzir os recursos computacionais alocados para o serviço. Do ponto de vista do usuário, talvez a principal vantagem seja a facilidade de simplesmente usar serviços que são nativamente oferecidos pela rede. A alternativa seria o próprio usuário implementar ou obter e manter/executar o serviço no seu próprio sistema computacional, tanto como

middleware ou aplicação. Alguns serviços inclusive podem ser reprojetados para utilizar informações da própria rede para serem oferecidos de forma mais eficiente, como o detector de falhas mencionado acima, que usa informações disponíveis no próprio OpenFlow para realizar sua função.

A arquitetura de referência NFV-MANO (NFV *MANagement and Orchestration*) [Quittek et al. 2014] tem sido amplamente adotada para disponibilizar, executar e gerenciar os serviços virtualizados baseados em NFV. Essa arquitetura também suporta a integração de serviços provenientes de diferentes desenvolvedores. Portanto, é essencial definir a arquitetura do NFV*inc* no contexto do NFV-MANO.

Deve ficar bem clara a diferença dos serviços tradicionais de rede que já são implementados usando a tecnologia NFV para os serviços NFV*inc*. Os serviços tradicionais, orquestrados pelo NFV-MANO como ele existe, não são usados integrados a aplicações de usuários finais. Eles recebem fluxos de pacotes e processam estes fluxos. Por outro lado, no caso dos serviços NFV*inc*, eles são acessados por aplicações finais que usam funções definidas na API correspondente. Desta forma, um dos principais desafios do NFV*inc* é sua interação com o usuário final, que é uma lacuna do NFV-MANO. Em outras palavras, deve ser definido de forma precisa como os serviços são ofertados, configurados, mantidos e acessados pelos usuários.

Assim, o presente trabalho discute os componentes necessários para construir uma arquitetura NFV*inc* integrada ao NFV-MANO. Os componentes desta arquitetura incluem: *(i)* uma biblioteca importada pelo usuário que contém as primitivas necessárias para o acesso e utilização dos serviços NFV*inc*; *(ii)* a interface do sistema que disponibiliza um conjunto de primitivas que permitem o acesso aos serviços NFV*inc*; e, por fim, *(iii)* o módulo que permite criar e disponibilizar serviços virtualizados, além de promover o acesso seguro e a gerência dos serviços.

O restante deste trabalho está organizado da seguinte forma. A Seção 2 descreve a tecnologia NFV e a arquitetura de referência NFV-MANO. A Seção 3 apresenta estudos de caso NFV*inc*. A Seção 4 trata dos desafios de pesquisa e discute requisitos de uma arquitetura genérica NFV*inc*. As conclusões seguem na Seção 5.

2. *Network Function Virtualization*: Uma Visão Geral

A Virtualização de Funções de Rede surge como uma alternativa concreta para implementar em software os diversos serviços de rede tradicionalmente disponibilizados em hardware dedicado [Mijumbi et al. 2016]. Serviços complexos de rede podem ser disponibilizados através da composição de múltiplas VNFs em uma *Service Function Chain* (SFC). Com o objetivo de padronizar a execução, interação e a gerência dos serviços baseados em NFV, a arquitetura de referência NFV-MANO foi proposta pela ETSI (*European Telecommunications Standards Institute*). A arquitetura NFV-MANO é responsável principalmente pelo controle do ciclo de vida e gerenciamento dos serviços virtualizados. A Figura 1 ilustra os principais blocos presentes em um rede baseada em NFV: a Infraestrutura NFV (NFVI), as VNFs, e o próprio NFV-MANO, descritos a seguir.

A NFVI representa a infraestrutura virtualizada onde as VNFs são instanciadas, gerenciadas e executadas. A parte física da NFVI consiste de recursos computacionais (*Compute*, *Storage* e *Network*). A NFVI realiza a abstração de recursos físicos em recursos virtuais através de uma camada de virtualização, que por sua vez é composta por

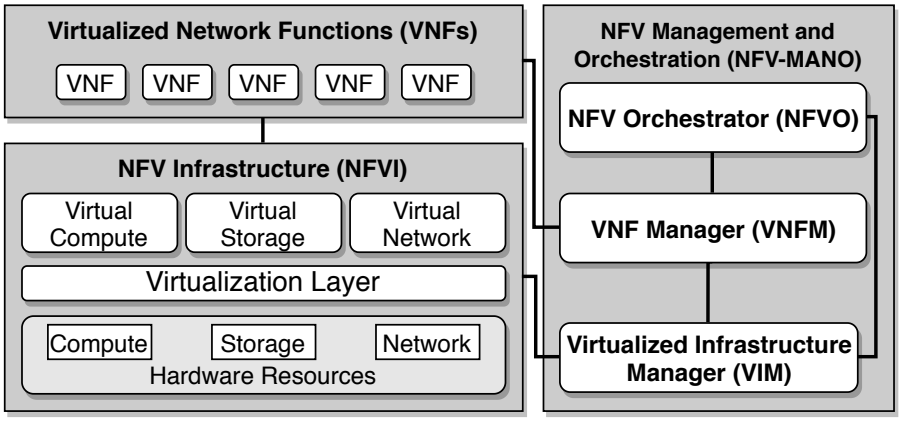

Figura 1. Arquitetura NFV-MANO proposta pela ETSI.

um *hypervisor* responsável por criar dispositivos virtualizados, como máquinas virtuais ou *containers*.

O bloco das VNFs representa as instâncias de cada elemento operacional virtualizado responsável pela execução das funções de rede que estão executando sobre a NFVI. As VNFs são funções de rede capazes de executar em uma infraestrutura virtualizada e que possuem interfaces de comunicação bem definidas para serem gerenciadas pelo NFV-MANO. Este bloco também compreende as SFCs – conjunto de VNFs que formam um serviço completo de rede.

A arquitetura NFV-MANO pode ser dividida em três módulos principais. O primeiro módulo é o NFV *Orchestrator* (NFVO), responsável por gerenciar e orquestrar os recursos alocados na infraestrutura virtualizada e pela gerência do ciclo de vida dos serviços de rede. Dessa forma, o NFVO coordena a composição de VNFs em SFCs. O segundo módulo é o VNF *Manager* (VNFM), que realiza a gerência do ciclo de vida das VNFs. Entre as atribuições deste módulo estão: instanciação, remoção, atualização, configuração e escalonamento das VNFs. Por fim, o módulo *Virtualized Infrastructure Manager* (VIM) realiza o controle e gerenciamento dos recursos computacionais da infraestrutura virtualizada (*e.g.,* computação, armazenamento e rede). O VIM, que normalmente corresponde a uma plataforma de nuvem, é responsável principalmente pela criação e remoção dos dispositivos virtuais.

3. NFV*inc*: Três Estudos de Caso

Esta seção descreve três estudos de caso de serviços INC disponibilizados através da tecnologia NFV. Os três serviços implementam abstrações clássicas de sistemas distribuídos tolerantes a falhas. A primeira abstração implementada é um detector de falhas de processos distribuídos. O segundo serviço implementa o consenso, aplicado na sincronização consistente do plano de controle distribuído de uma rede SDN. Por fim, o terceiro estudo de caso apresenta um serviço que permite à rede oferecer nativamente a difusão confiável e ordenada.

3.1. Serviço de Detecção de Falhas: NFV-FD

Os detectores de falhas foram originalmente propostos para auxiliar na resolução do consenso em sistemas distribuídos sem garantias temporais, *i.e.,* assíncronos. O consenso é

um problema fundamental de sistemas distribuídos que garante que todos os processos do sistema entrem em acordo sobre um único valor dentro de um conjunto de valores propostos inicialmente [Borran et al. 2012]. Além disso, o consenso deve ser garantido mesmo que processos participantes falhem. No entanto, em [Fischer et al. 1985] provou-se que é impossível a execução do consenso em sistemas distribuídos assíncronos sujeitos a falhas por parada. Visando contornar esta impossibilidade, os detectores de falhas não confiáveis [Chandra and Toueg 1996] foram propostos como uma abstração que é acessada pelos processos para obter informações sobre o estado (correto/suspeito de falha) de outros processos no sistema. Uma estratégia simples para implementar um detector de falhas é através do monitoramento de processos, que periodicamente enviam mensagens de *heartbeat*: se o detector não recebe uma destas mensagens dentro de um intervalo de tempo adequado, o processo é suspeito de ter falhado.

Em [Turchetti and Duarte Jr 2017] foi proposta a implementação de um detector de falhas baseado em NFV. A solução, denominada de NFV-FD (NFV *Failure Detector*), utiliza informações obtidas através do controlador SDN para monitorar os processos e determinar os seus estados (*i.e.,* correto ou suspeito de falha). O NFV-FD é baseado em um módulo denominado FDMod, que é um filtro aplicado sobre os pacotes recebidos pelo controlador, responsável por extrair informações de estado dos processos monitorados. O FDMod inspeciona pacotes examinando campos do cabeçalho e envia informações relevantes ao NFV-FD. Ao receber estas informações, o NFV-FD atualiza adequadamente os estados dos processos monitorados.

O NFV-FD também detecta falhas nos enlaces de comunicação. Para tanto, é utilizado o mecanismo de descoberta de topologia (*Link Layer Discovery Protocol* – LLDP). Os próprios *switches* enviam pacotes do tipo LLDP para informar mudanças na topologia da rede. Todos os pacotes LLDP são capturados, filtrados e processados pelo FDMod, que posteriormente envia as informações relevantes para o NFV-FD. Por exemplo, a mensagem *port s2-eth2 changed: DOWN* indica que a porta *eth2* do *switch s2* está inativa. O NFV-FD por sua vez verifica se existem processos conectados nesta porta e marca estes processos como inatingíveis (*unreachable*).

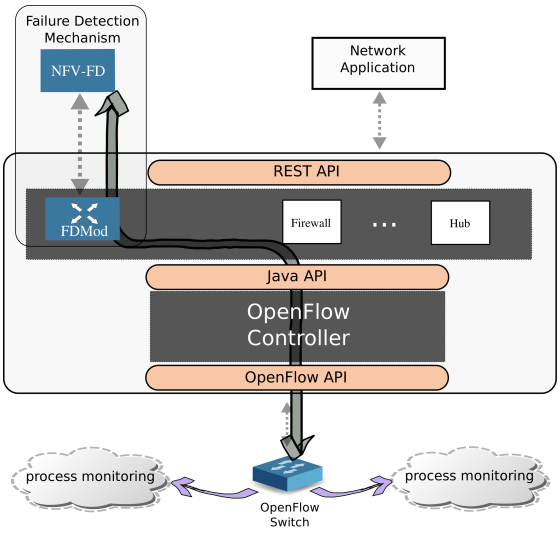

Figura 2. Arquitetura da NFV-FD.

Resultados experimentais demonstram que a abordagem baseada em NFV apresenta baixa utilização de recursos computacionais, em comparação com uma alternativa que implementa o detector no próprio controlador SDN. Por outro lado, o tempo de detecção do NFV-FD é levemente maior, visto que o detector de falhas não está disponível no mesmo *host* que o controlador SDN e há uma pequena latência introduzida pela comunicação com o controlador para detectar mudanças de estado dos processos monitorados.

3.2. Consistência do Plano de Controle: VNF-Consensus

Por padrão, o plano de controle em redes SDN é centralizado [Ho et al. 2016], o que afeta diretamente a disponibilidade, escalabilidade e desempenho da rede [Canini et al. 2015]. Apesar de que um plano de controle distribuído pode resolver estes problemas, alcançar uma sincronização consistente entre múltiplos controladores SDN não é uma tarefa trivial. As soluções existentes empregam os próprios controladores para realizar as tarefas de sincronização [Ho et al. 2016] ou sincronizam o próprio plano de dados [Dang et al. 2015]. Em [Venâncio et al. 2019] é proposta a *VNF-Consensus* que implementa o algoritmo de consenso Paxos [Lamport 1998] para manter a consistência de um plano de controle distribuído, composto por múltiplos controladores SDN. O objetivo é evitar o aumento da demanda de recursos dos controladores além de não implicar em qualquer modificação do plano de dados nem do protocolo SDN (no caso o OpenFlow [McKeown et al. 2008]).

A Figura 3 mostra o funcionamento da *VNF-Consensus*. Para garantir a consistência, antes que uma nova regra OpenFlow seja aplicada no plano de dados, deve ser replicada para os múltiplos controladores SDN. Dessa forma, para toda regra não existente na tabela de fluxos de um *switch*, o respectivo controlador encaminha esta regra para a *VNF-Consensus*. A *VNF-Consensus*, por sua vez, executa o algoritmo de consenso e retorna a decisão (*i.e.,* a regra a ser instalada) aos controladores, que atualizam seu estado local, mantendo o plano de controle consistente. Os controladores podem então atualizar o plano de dados adequadamente. Caso algum controlador falhe, outro controlador pode assumir as suas tarefas, uma vez que eles possuem uma mesma visão da rede.

Resultados experimentais demonstram a eficiência da *VNF-Consensus* em comparação com a implementação do consenso nos controladores. A utilização de CPU por um controlador reduz em até 50%. Este resultado é particularmente importante, porque implica em uma maior disponibilidade do controlador, que desta forma, pode aumentar o número máximo de requisições que processa por segundo. Ao mesmo tempo, a latência para instalar uma regra consistente sobre os múltiplos controladores diminui em até 18%.

3.3. Difusão Confiável e Ordenada de Mensagens: NFV-RBCast

Uma abstração importante para o desenvolvimento de aplicações distribuídas é a difusão confiável (*reliable broadcast*), que garante a entrega das mensagens por todos os processos corretos [Défago et al. 2004]. Além disso, o serviço pode entregar mensagens ordenadas de acordo com diferentes critérios. Tradicionalmente, usuários ou implementam difusão na própria aplicação, ou instalam/mantêm *middleware* com esta funcionalidade. A solução NFV-RBCast [Venâncio et al. 2019] utiliza tecnologia NFV*inc* para permitir que a rede ofereça difusão confiável e ordenada como serviços nativos. Para as difusões

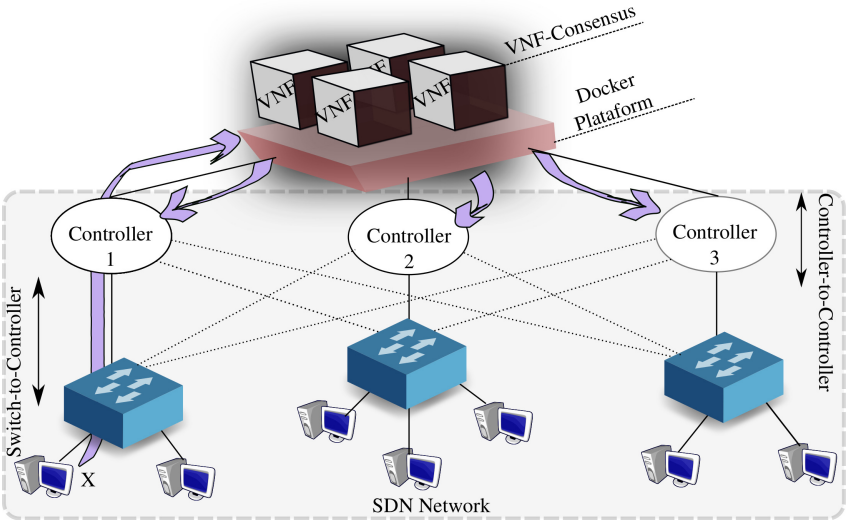

Figura 3. Arquitetura da *VNF-Consensus*.

ordenadas a NFV-RBCast considera duas sequências de mensagens. A primeira respeita a ordem FIFO (*First-In First-Out*) local de transmissão de cada processo. A outra é a ordem global, que garante a entrega atômica. Concretamente, são oferecidas as seguintes modalidades de difusão: difusão confiável, difusão atômica, difusão atômica FIFO e difusão atômica causal. As aplicações acessam as primitivas de difusão através de uma API denominada *RBCast*.

Na NFV-RBCast, a ordem das mensagens é garantida através de um sequenciador implementado como uma VNF, denominada de *VNF-Sequencer*, como mostra a Figura 4. Do funcionamento padrão do protocolo *OpenFlow*, todo pacote que não possui uma respectiva entrada na tabela de fluxos do *switch* SDN é encaminhado ao controlador SDN através de uma mensagem *packet_in*. Para cada *packet_in* recebido, o controlador SDN verifica se a mensagem foi originada a partir de uma difusão. Em caso afirmativo, o controlador instala uma regra no *switch* para que todas as mensagens originadas pela NFV-RBCast para este tipo de fluxo sejam encaminhadas diretamente para a *VNF-Sequencer*. Para o restante do fluxo, todas as mensagens são enviadas diretamente para a *VNF-Sequencer*, sem a necessidade da intervenção do controlador.

Ao receber um fluxo de pacotes, a *VNF-Sequencer* executa o algoritmo de ordenação especificado pelo processo emissor. Ao enviar as mensagens para os processos da difusão, a *VNF-Sequencer* insere na mensagem um valor de sequência global. A mensagem é enviada aos processos seguindo o funcionamento padrão do protocolo *OpenFlow*. Ao receber uma mensagem de difusão, o processo entrega a mensagem à aplicação levando em conta o valor de sequência global inserido pela *VNF-Sequencer*.

Resultados experimentais demonstram que a *VNF-Consensus* atinge uma vazão de até 15000 mensagens/s na difusão atômica, para 50 processos, e um aumento de até 32% na latência – valor decorrente do fato que a *VNF-Sequencer* garante a ordem total das mensagens.

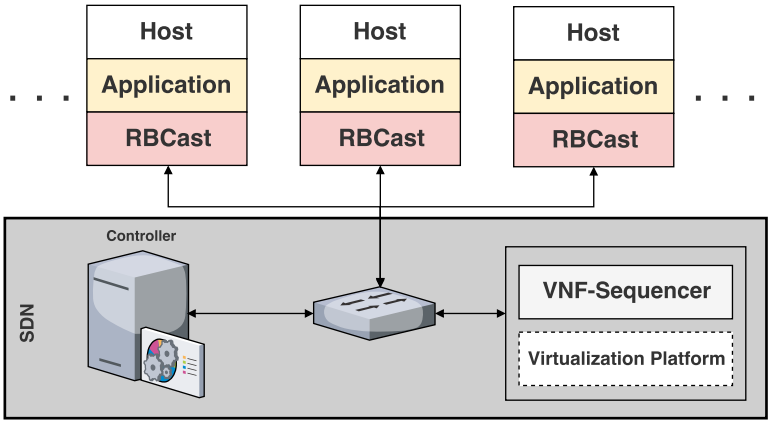

Figura 4. Arquitetura da NFV-RBCast.

4. A Arquitetura NFV*inc*

Com os diversos avanços da tecnologia de virtualização torna-se possível customizar os serviços oferecidos nativamente pela rede. Neste trabalho, nosso foco está na utilização da tecnologia NFV como suporte para que novos serviços e funcionalidades possam ser oferecidos dentro da própria rede. A Figura 5 representa uma arquitetura genérica de uma rede virtualizada. A figura mostra na parte superior um conjunto de servidores físicos localizados no núcleo da rede. Estes servidores são dotados de uma camada de virtualização, em que dispositivos tais como máquinas virtuais e *containers* abstraem os recursos físicos dos servidores (*e.g., Compute, Storage, Network*) em recursos virtuais. Serviços de rede de diversos tipos podem ser construídos com funções virtuais de rede executando neste ambiente. A figura mostra diversos exemplos tradicionais de funções de rede que já têm sido executadas de forma virtualizada: DPI, roteador, *proxy*, e *firewall*. Os serviços implementados por estas funções podem ser acessados a partir de qualquer *host* em uma das redes da borda, mostradas na parte inferior da figura.

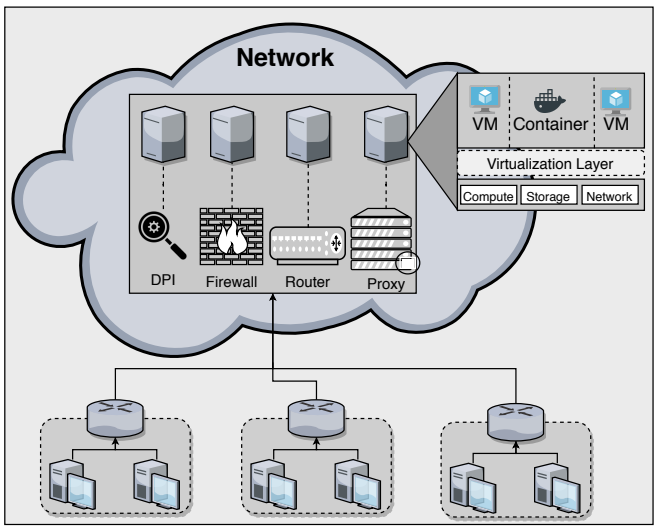

Figura 5. Arquitetura genérica de uma rede virtualizada.

Apesar da arquitetura NFV-MANO oferecer diversas funcionalidades que são indispensáveis para a execução dos serviços virtualizados, ainda existem diversas lacunas

para que a tecnologia NFV possa ser utilizada para oferecer serviços arbitrários na rede. Neste trabalho, temos por objetivo complementar a arquitetura NFV-MANO através da arquitetura NFV*inc*.

Conforme mostra a Figura 6, a arquitetura NFV*inc* é dividida em duas camadas, denominadas de *User Layer* e *NFV Layer*. Na camada *User Layer* estão todos os componentes relacionados ao usuário, tais como servidores/nodos físicos, aplicações que são hospedadas nesses servidores, além da biblioteca libNFV*inc*.

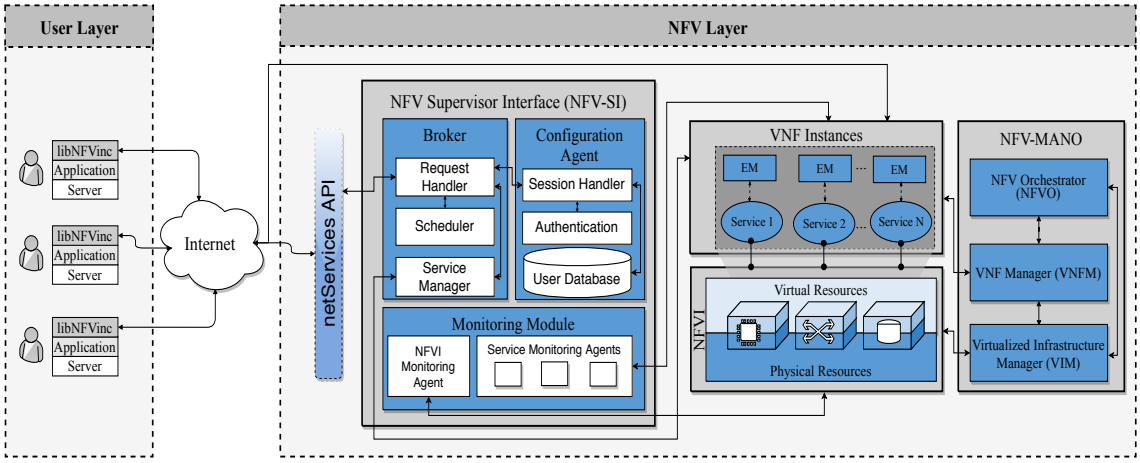

Figura 6. Componentes internos da arquitetura NFV*inc*.

A biblioteca libNFV*inc* é responsável pela comunicação com a *NFV Layer*. Para tanto, a libNFV*inc* possui todas as primitivas necessárias para acessar e gerenciar os serviços virtualizados, possibilitando a criação, utilização e remoção dos serviços NFV*inc*. A libNFV*inc* permite que os usuários acessem os serviços virtualizados de maneira transparente e unificada, garantindo a facilidade de integração de diferentes tipos de serviços em uma aplicação.

A *NFV Layer*, por sua vez, contém os componentes originais da arquitetura NFV-MANO (*i.e.,* NFVO, VNFM, VIM), o servidor físico responsável pela virtualização dos serviços (*i.e.,* NFVI), os próprios serviços virtualizados e também o NFV-SI (NFV *Supervisor Interface*). O NFV-SI é componente principal da arquitetura NFV*inc*, e é composto pelos módulos *Broker*, *Configuration Agent* e *Monitoring Module*, além da API *netServices*. Todos os componentes são descritos a seguir.

O componente NFV-SI atua como elemento intermediário entre o usuário final (*i.e.,* aplicação), os serviços virtualizados e a camada de virtualização. Todas as requisições feitas pela aplicação através da libNFV*inc* são recebidas pela *netServices* API – uma API genérica através da qual os usuários/aplicações podem acessar e manipular os serviços virtualizados que estão disponibilizados na rede. Essa API é genérica no sentido que: *(i)* permite que as aplicações utilizem os serviços disponíveis na rede através de uma chamada padronizada; *(ii)* a API é compatível com diferentes tipos de tecnologias de virtualização, tais como NFV, SDN, P4 e computação em nuvem; *(iii)* é compatível com a maior parte das linguagens de programação, permitindo a interoperabilidade entre diferentes desenvolvedores.

A *netServices* API permite que as aplicações acessem os serviços virtualizados de

maneira unificada e independente da tecnologia com que a aplicação foi implementada, visto que a API é baseada em uma interface REST (*REpresentational State Transfer*). Consequentemente, a integração de aplicações fornecidas por diferentes desenvolvedores também é facilitada. A *netServices* API é composta por um conjunto de primitivas utilizadas pelas aplicações para executarem ações específicas. Por fim, a *netServices* API também possui uma primitiva *discovery*, responsável por retornar ao usuário um catálogo dos serviços virtualizados que estão disponíveis na rede.

Todas as requisições recebidas pela *netServices* API são encaminhadas para o módulo *Broker*, elemento responsável por gerenciar as requisições e permitir o tráfego entre a aplicação e os serviços virtualizados. Para tanto, o *Broker* possui três submódulos. O primeiro submódulo é o *Request Handler*, responsável por receber as requisições encaminhadas pela *netServices* API. O *Request Handler* possui informações que permitem identificar o tipo da requisição, para então encaminhá-la ao submódulo correspondente (*i.e., Service Manager* ou *Scheduler*).

O *Service Manager* é o submódulo que realiza requisições ao NFV-MANO para o gerenciamento dos serviços, tais como novas instanciações de serviços, monitoramento, reconfiguração ou remoção (quando o serviço não é mais necessário). O submódulo *Scheduler* por sua vez é utilizado para agendar requisições quando elas não podem ser executadas no momento, devido a, por exemplo, uma indisponibilidade no serviço (mais detalhes na Seção 4.1).

O segundo módulo do NFV-SI é o *Configuration Agent* (CA), responsável tanto pela configuração geral dos serviços virtualizados, quanto pela configuração da sessão do usuário com a rede NFV, realizando o estabelecimento de conexões seguras entre o usuário e os serviços. Além disso, o CA cria uma rede virtual dedicada para cada usuário, permitindo que instâncias individuais dos serviços virtualizados sejam criadas. Essa abordagem garante o isolamento e segurança; além disso, como não há competição por recursos evita que surjam gargalos mesmo quando aumenta o número de usuários. O CA também mantém informações em um banco de dados local sobre os usuários e suas respectivas conexões, *e.g.,* os serviços que cada usuário possui, qual rede virtual está associada a qual usuário e conexões ativas.

Por fim, o terceiro submódulo do NFV-SI é o módulo de monitoramento, responsável por monitorar o estado dos serviços virtualizados instanciados e também da NFVI. O módulo de monitoramento suporta diversas métricas, tais como a utilização de CPU, de banda (*in/out*) e de memória. Além disso, o módulo de monitoramento monitora o estado dos canais de comunicação. Visto que ambientes reais NFV normalmente utilizam um *deploy* distribuído, *i.e.,* as instâncias podem estar geograficamente separadas. Portanto, uma queda no *switch*/roteador/nodo pode impedir a comunicação com uma determinada instância. Os recursos virtuais disponíveis na NFVI também são monitorados.

4.1. Funcionamento da Arquitetura NFV*inc*

A arquitetura NFV*inc* é composta pelos serviços virtualizados, a biblioteca libNFV*inc*, a NFV-SI e a *netServices* API. O conjunto de funcionalidades oferecido por estes componentes permite que a maior parte dos serviços virtualizados possam ser oferecidos dentro da rede e utilizados por diversas aplicações distribuídas pela Internet.

Dessa forma, a arquitetura NFV*inc* disponibiliza quatro operações fundamentais:

(i) estabelecer conexões seguras e autenticadas com serviços virtualizados; *(ii)* criar e disponibilizar novos serviços virtualizados; *(iii)* gerenciar os serviços virtualizados e; *(iv)* agendar requisições para os serviços virtualizados. A maneira como cada operação é executada é descrita a seguir.

4.1.1. Conectando com um Serviço Virtualizado

Para que uma aplicação possa incorporar um serviço virtualizado é necessário realizar o estabelecimento de uma conexão. Neste sentido, o primeiro passo consiste na execução da chamada *discovery* para a *netServices* API, de forma a obter uma lista com todos os serviços disponíveis na rede. A aplicação pode exigir um ou mais serviços virtualizados, portanto com o resultado da primitiva *discovery* o usuário/aplicação verifica se o serviço requisitado/necessário está disponível (*e.g.,* uma aplicação banco de dados distribuído que precisa de um serviço de consenso). Caso esteja disponível, a libNFV*inc* faz uma requisição através da *netServices* API para estabelecer uma nova conexão com este serviço.

A *netServices* API recebe esta requisição e a encaminha para o *Broker* (*i.e., Request Handler*), que irá identificar a requisição como um novo estabelecimento de conexão. O *Request Handler* então envia uma mensagem ao módulo CA para que uma nova sessão seja criada. Uma sessão consiste de uma conexão segura e autenticada, além de informações da aplicação, tais como endereço IP, serviço requisitado, quota, identificadores, etc. A criação de uma sessão é descrita a seguir.

O primeiro passo para criar uma nova sessão ocorre quando o CA faz uma requisição ao NFV-MANO para a criação de uma nova rede virtual para o usuário/aplicação em questão. Essa abordagem garante isolamento e segurança, evitando o uso não autorizado do serviço virtualizado requisitado. A segunda etapa consiste das respectivas autenticações (*token*, login/senha, etc.). Por fim, na terceira etapa o *Broker* grava no banco de dados do CA todas as informações relevantes (identificador do usuário com a sua respectiva rede virtual).

Após a sessão ser estabelecida, o *Broker* realiza uma requisição para instanciar os serviços virtualizados requisitados pela aplicação. Para tanto, o módulo *Service Manager* faz uma requisição ao NFV-MANO para criar uma nova instância do serviço virtualizado requerido dentro da rede virtual do usuário. O *Broker* aguarda até que a instanciação seja concluída, para então retornar uma mensagem à aplicação informando que o serviço está disponível.

4.1.2. Comunicação com os Serviços Virtualizados

Após a conexão com o serviço virtualizado ser estabelecida, a arquitetura NFV*inc* prevê dois métodos de comunicação da aplicação com os serviços: comunicação direta através da *netServices* API ou comunicação direta com o serviço. Ambos os métodos são descritos a seguir.

No método de comunicação direta através da *netServices* API, a aplicação envia os dados para o serviço virtualizado diretamente pela *netServices* API. Este método é

adequado para requisições que envolvem a comunicação de baixa quantidade de dados e que não exigem o melhor desempenho possível (*e.g.,* adicionar um processo no detector de falhas, enviar baixo volume de dados para sincronização). Visto que a *netServices* API é composta por um conjunto de primitivas que permitem ao usuário a manipulação dos serviços virtualizados (GET, POST), essa comunicação permite a execução de algumas ações de maneira simples e com baixa complexidade para a aplicação, visto que utiliza apenas requisições REST.

Por outro lado, no método de comunicação direta com o serviço virtualizado, a aplicação envia dados através de uma conexão dedicada com o serviço. Este método é recomendado para comunicações com alto volume de dados e/ou que exigem alto desempenho (*e.g.,* DPI, DPDK). Para tanto, o *Broker* retorna ao usuário uma conexão direta com o serviço, através da qual a libNFV*inc* (presente na aplicação) se comunica com o serviço. Além disso, o *Broker* cria novas rotas locais na rede virtual do usuário para que os dados que chegam desta aplicação sejam encaminhados imediatamente para o serviço virtual correspondente.

4.1.3. Gerenciamento de Serviços

Uma vez que os serviços virtualizados que serão utilizados pela aplicação estejam em operação, deve ser possível gerenciá-los de maneira dinâmica durante a execução da aplicação. Com a arquitetura NFV*inc*, este gerenciamento pode ser feito através da *netServices* API, realizando operações como remoção do serviço (caso não seja mais necessário), requisição de *autoscaling* (permitindo que a quantidade de recursos utilizados e/ou o número de instâncias aumente e/ou diminua de acordo com a demanda do sistema, até um limite máximo), além de reconfigurar o serviço de acordo com as primitivas disponibilizadas.

Ao receber uma requisição de gerenciamento, a *netServices* API encaminha esta requisição para o *Broker* (*Request Handler*). O *Broker* por sua vez identifica o tipo de requisição e envia para o *Service Manager*, em caso de remoção ou *autoscaling*, ou diretamente para os serviços virtualizados em caso de reconfiguração.

4.1.4. Agendamento de Requisições

Existem determinados casos em que uma ou mais requisições podem não ser executadas imediatamente pela plataforma NFV: *(i)* o serviço não possui mais recursos para executar a requisição; *(ii)* o serviço está em processo de recuperação/reconfiguração; *(iii)* a NFVI não possui mais recursos para executar novas instâncias de serviços; *(iv)* houve quedas de canal de comunicação, impedindo a comunicação com o serviço ou; *(v)* o serviço não está responsivo dentro do intervalo de tempo esperado.

Quando o módulo de monitoramento detecta alguma indisponibilidade em algum dos serviços virtualizados, este módulo envia o ID deste serviço para o *Request Handler*, informando que novas requisições com destino a este serviço devem aguardar. Ao receber esta mensagem, o *Request Handler* adiciona uma *flag* `REQUEST_TO_SCHEDULE` na sessão do usuário. Dessa foma, as novas requisições são encaminhadas ao *Scheduler*. O

Scheduler por sua vez, armazena em um *logger* todas as requisições recebidas.

Enquanto isso, o módulo de monitoramento aguarda até que as indisponibilidades com o serviço sejam resolvidas. Note que a arquitetura NFV*inc* não é responsável pela recuperação destes serviços, mas sim o NFV-MANO. Uma vez que o módulo de monitoramento detecta que o serviço está operacional novamente, uma mensagem é enviada ao *Request Handler* informando que o serviço está apto a receber novas requisições. O *Request Handler*, por sua vez, envia uma mensagem ao *Scheduler*, para que este retorne todas as requisições que foram armazenadas no *logger* (requisições agendadas). O *Request Handler* recebe estas mensagens pendentes e as envia para o serviço correspondente para serem processadas. Por fim, o *Request Handler* remove a *flag* da sessão do usuário e novas requisições podem ser encaminhadas diretamente para o serviço.

5. Conclusão

Este trabalho explora uma nova alternativa inovadora para a disponibilização de serviços arbitrários dentro da rede (*i.e.,* INC: *In-Network Computing*) baseada na tecnologia NFV. A principal diferença dos serviços tradicionais implementados com NFV para os serviços NFV*inc* é que estes são utilizados por aplicações dos usuários através de APIs disponibilizadas pela arquitetura proposta. Já foram implementados diversos serviços distribuídos dentro da rede usando NFV, o trabalho descreve brevemente três deles como estudos de caso que demonstram claramente a viabilidade da proposta: NFV-FD, VNF-Consensus e NFV-RBCast.

O principal serviço que a arquitetura NFV*inc* agrega ao NFV-MANO é justamente permitir os serviços que possam ser usados/integrados com facilidade pelas aplicações. A arquitetura consiste de quatro componentes principais: *(i)* a biblioteca libNFV*inc*, com primitivas para permitir a comunicação entre o usuário e os serviços virtualizados; *(ii) netServices* API, que funciona como ponto de entrada das requisições para os serviços; *(iii)* NFV-SI, o componente principal que atua como intermediário entre a aplicação, os serviços virtualizados e a infraestrutura virtualizada; além dos *(iv)* os próprios serviços virtualizados. A arquitetura NFV*inc* oferece quatro funcionalidades fundamentais: o estabelecimento de conexão segura da aplicação com os serviços virtualizados; a comunicação entre os dois; o gerenciamento dos serviços virtualizados e o agendamento de requisições para os serviços virtualizados, inclusive considerando possíveis indisponibilidades.

Trabalhos futuros serão direcionados para a divulgação da arquitetura NFV*inc*, com o objetivo de que um número maior de serviços inovadores sejam disponibilizados pelas redes, seguindo a estratégia proposta.

Referências

Borran, F., Hutle, M., Santos, N., and Schiper, A. (2012). Quantitative analysis of consensus algorithms. *IEEE Transactions on Dependable and Secure Computing*, 9(2).

Bosshart, P., Daly, D., Gibb, G., Izzard, M., McKeown, N., Rexford, J., Schlesinger, C., Talayco, D., Vahdat, A., Varghese, G., et al. (2014). P4: Programming protocol-independent packet processors. *ACM SIGCOMM Computer Communication Review*, 44(3):87–95.

Bressana, P., Zilberman, N., Vucinic, D., and Soulé, R. (2020). Trading latency for compute in the network. In *Proceedings of the 2020 Workshop on Network Application Integration/CoDesign, NAI@SIGCOMM 2020, Virtual Event, USA, August 14, 2020*, pages 35–40. ACM.

Canini, M., Kuznetsov, P., Levin, D., and Schmid, S. (2015). A distributed and robust SDN control plane for transactional network updates. In *IEEE Conference on Computer Communications (INFOCOM)*.

Chandra, T. D. and Toueg, S. (1996). Unreliable failure detectors for reliable distributed systems. *Journal of ACM*, 43(2).

Dang, H. T., Bressana, P., Wang, H., Lee, K. S., Zilberman, N., Weatherspoon, H., Canini, M., Pedone, F., and Soulé, R. (2020). P4xos: Consensus as a network service. *IEEE/ACM Transactions on Networking*.

Dang, H. T., Canini, M., Pedone, F., and Soulé, R. (2016). Paxos made switch-y. *ACM SIGCOMM Computer Communication Review*, 46(2):18–24.

Dang, H. T., Sciascia, D., Canini, M., Pedone, F., and Soulé, R. (2015). Netpaxos: Consensus at network speed. In *Symposium on Software Defined Networking Research, (SOSR'15/SIGCOMM)*.

Dean, J. and Ghemawat, S. (2008). Mapreduce: simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113.

Défago, X., Schiper, A., and Urbán, P. (2004). Total order broadcast and multicast algorithms: Taxonomy and survey. *ACM Comput. Surv.*, 36(4):372–421.

Fischer, M. J., Lynch, N. A., and Paterson, M. S. (1985). Impossibility of distributed consensus with one faulty process. *Journal of ACM*, 32(2).

Ho, C. C., Wang, K., and Hsu, Y. H. (2016). A fast consensus algorithm for multiple controllers in software-defined networks. In *18th International Conference on Advanced Communication Technology (ICACT)*.

Lamport, L. (1998). The part-time parliament. *ACM Transactions on Computer Systems (TOCS)*, 16(2).

Liu, M., Luo, L., Nelson, J., Ceze, L., Krishnamurthy, A., and Atreya, K. (2017). Incbricks: Toward in-network computation with an in-network cache. In *ACM ASPLOS*, pages 795–809.

McKeown, N., Anderson, T., Balakrishnan, H., Parulkar, G., Peterson, L., Rexford, J., Shenker, S., and Turner, J. (2008). Openflow: Enabling innovation in campus networks. *SIGCOMM Comput. Commun. Rev.*, 38(2):69–74.

Mijumbi, R., Serrat, J., Gorricho, J. L., Bouten, N., Turck, F. D., and Boutaba, R. (2016). Network Function Virtualization: State-of-the-Art and Research Challenges. *IEEE Communications Surveys Tutorials*, 18(1).

Quittek, J., Bauskar, P., BenMeriem, T., Bennett, A., Besson, M., and et al (2014). Network Functions Virtualisation (NFV); Management and Orchestration. GS NFV-MAN 001 V1.1.1. Technical report, ETSI.

Sapio, A., Abdelaziz, I., Aldilaijan, A., Canini, M., and Kalnis, P. (2017). In-network computation is a dumb idea whose time has come. In *Proceedings of the 16th ACM Workshop on Hot Topics in Networks*, pages 150–156.

Tokusashi, Y., Dang, H. T., Pedone, F., Soulé, R., and Zilberman, N. (2019). The case for in-network computing on demand. In *Proceedings of the Fourteenth EuroSys Conference 2019*, pages 1–16.

Turchetti, R. C. and Duarte, E. P. (2015). Implementation of a failure detector based on network function virtualization. In *IEEE DSNWorkshops*, pages 19–25.

Turchetti, R. C. and Duarte Jr, E. P. (2017). Nfv-fd: Implementation of a failure detector using network virtualization technology. *International Journal of Network Management*, 27(6):e1988.

Venâncio, G., Turchetti, R. C., de Camargo, E. T., and Jr., E. P. D. (2019). Vnf-consensus: A virtual network function for maintaining a consistent distributed SDN control plane. In *IFIP IEEE LANOMS*.

Venâncio, G., Turchetti, R. C., and Duarte, E. P. (2019). Nfv-rbcast: Enabling the network to offer reliable and ordered broadcast services. In *IEEE LADC*.