Uma Arquitetura de Alta Disponibilidade para Funções Virtualizadas de Rede

Giovanni Venâncio[1], **Elias P. Duarte Jr.**[1]

[1]Universidade Federal do Paraná (UFPR) – Curitiba – PR – Brasil

gvsouza@inf.ufpr.br, elias@inf.ufpr.br

Abstract. *Virtualization has represented a revolution on the way networks are built and managed. In particular, dedicated hardware can be replaced by Virtualized Network Functions (VNFs) which can be even downloaded from Internet marketplaces. However, it is undeniable that VNFs are more susceptible to failures. In this work we propose a high availability architecture for VNFs, which provides fault management and a variety of recovery techniques. As stateful VNFs are executed in virtualized environments, copying the entire state is an attractive strategy that does not require any code modifications to the VNFs. The architecture is based on Checkpoint/Restore and was designed within the NFV-MANO reference architecture. A proof-of-concept prototype was implemented and experimental results are presented.*

Resumo. *A virtualização vem revolucionando a forma como as redes são construídas e gerenciadas. Em especial, funções de rede implementadas em hardware dedicado podem ser substituídas por Virtual Network Functions (VNFs), obtidas inclusive em marketplaces na Internet. Entretanto, é inquestionável que as VNFs apresentam maior susceptibilidade a falhas. Este trabalho propõe uma arquitetura de alta disponibilidade para VNFs, englobando o gerenciamento de falhas e diversas estratégias de recuperação. Como as VNFs executam em ambientes virtualizados, para VNFs stateful é possível copiar todo o seu estado, uma estratégia atraente e que não exige modificações no código interno da VNF. A estratégia é baseada em Checkpoint/Restore e a arquitetura foi projetada de acordo com à arquitetura de referência NFV-MANO. Um protótipo foi implementado como prova de conceito e resultados experimentais são apresentados.*

1. Introdução

Os protocolos da Internet foram desenvolvidos há décadas, e não foram projetados prevendo o extraordinário crescimento que ocorreu, o que contribuiu para a chamada "ossificação da Internet". As tecnologias de virtualização representam uma estratégia concreta para a solução deste problema, permitindo a evolução da rede para múltiplas direções, melhorando a sua flexibilidade e gerenciamento. Com o paradigma NFV (*Network Function Virtualization*), torna-se possível disponibilizar novas funcionalidades na rede utilizando hardware genérico [Chiosi et al. 2012]. Através de técnicas de virtualização, serviços de rede são disponibilizados em dispositivos virtuais, denominados de Funções Virtualizadas de Rede (*Virtualized Network Functions* - VNFs) [Mijumbi et al. 2016]. Com o auxílio de NFV, os serviços de rede que antes eram produzidos apenas por um número limitado de desenvolvedores, agora podem ser obtidos de maneira virtualizada em *marketplaces* na Internet [Bondan et al. 2019].

Apesar das diversas vantagens que a tecnologia NFV oferece, é inquestionável que funções de rede executadas de maneira virtualizada são mais propensas a falhas em comparação com os dispositivos de hardware dedicados [Han et al. 2015]. A transição dos dispositivos de hardware para plataformas virtualizadas introduz diversos desafios relacionados à confiabilidade dos sistemas [Cotroneo et al. 2014, Mijumbi et al. 2016]. A complexidade em integrar diferentes softwares em múltiplas camadas, os desafios de interoperabilidade entre componentes de hardware e software fornecidos por diferentes desenvolvedores e a inexperiência na operação de ambientes virtualizados de rede [Schöller et al. 2015] são alguns dos fatores que contribuem para uma expectativa de aumento de riscos em termos da confiabilidade das redes baseadas em VNFs.

Funções de rede implementadas em dispositivos especializados de hardware normalmente são projetadas para oferecer alta disponibilidade, atingindo o nível exigido para sistemas comerciais de telecomunicações: *five nines* o que corresponde a 99.999% do tempo em estado correto [Gray and Siewiorek 1991]. Dessa forma, torna-se essencial prover mecanismos que minimizem o tempo de inatividade de VNFs na presença de falhas. Em especial, a *European Telecommunications Standards Institute* (ETSI) definiu vários requisitos de resiliência para serviços executados em um ambiente virtualizado [Schöller et al. 2015]. Entretanto, a disponibilidade e continuidade do serviço estão sendo negligenciados pelos desenvolvedores de VNFs [Han et al. 2017].

Apesar de que diversas propostas foram feitas para aumentar a disponibilidade de funções de rede em ambientes virtualizados, tais soluções possuem limitações, como a garantia de confiabilidade apenas para tecnologias específicas, modificações no código interno da VNF, ou não englobam todas as funcionalidades necessárias para garantir confiabilidade de ponta a ponta nos serviços virtualizados. O problema torna-se ainda mais complexo ao considerar que grande parte das funções de rede são *stateful* – o estado interno muda de acordo com os pacotes processados. Assim, a recuperação de uma VNF *stateful* só é concluída se a restauração do seu estado antes da falha ocorrer. Por fim, as soluções não são totalmente compatíveis com a arquitetura referência NFV-MANO (*NFV Management and Orchestration*) definida pela ETSI [Quittek et al. 2014].

Neste contexto, este trabalho propõe o NHAM (*NFV High Availability Module*), uma arquitetura de alta disponibilidade para VNFs, integrada como um módulo da arquitetura NFV-MANO, totalmente compatível com as especificações definidas pela ETSI. O NHAM permite que todas as VNFs instanciadas na plataforma NFV herdem as propriedades de alta disponibilidade de maneira transparente para o usuário. Para tanto, o NHAM realiza o gerenciamento de falhas, englobando a detecção e recuperação de VNFs. Assume-se falhas do tipo *crash-recovery* de uma única VNF. Além disso, de forma a preservar o estado interno de VNFs *stateful* em caso de falha, técnicas baseadas em *Checkpoint/Restore* são utilizadas.

O NHAM é agnóstico em relação à tecnologia NFV: qualquer VNF se torna altamente disponível sem a necessidade de alterações durante a sua implementação. Como as VNFs executam em ambientes virtualizados, copiar o estado do dispositivo virtual torna-se uma estratégia concreta e atrativa, uma vez que esta abordagem não exige modificações no código interno da VNF. A viabilização desta estratégia ocorre através da realização de *checkpoints* diretamente nos recursos virtuais em que a VNF está executando.

Um protótipo foi implementado como prova de conceito e experimentos mostram o desempenho e a disponibilidade para diferentes estratégias de resiliência. Em espe-

cial, dependendo da estratégia e dos parâmetros, é possível alcançar a disponibilidade de sistemas comerciais de telecomunicações.

O restante deste trabalho está organizado da seguinte forma. A Seção 2 apresenta conceitos de NFV e uma visão geral da arquitetura NFV-MANO. As Seções 3 e 4 detalham o NHAM e o sistema de gerenciamento de falhas, respectivamente. A Seção 5 apresenta a implementação e a avaliação experimental do NHAM. Por fim, a Seção 6 descreve os trabalhos relacionados e a Seção 7 conclui o trabalho.

2. Uma Visão Geral da Arquitetura NFV-MANO

O paradigma NFV surge como uma alternativa concreta baseada em virtualização para flexibilizar e simplificar serviços de rede tradicionalmente disponibilizados em hardware dedicado [Chiosi et al. 2012]. Através de técnicas de virtualização, funções de rede são implementadas como VNFs (*Virtualized Network Functions*) e executadas em hardware de prateleira. Além disso, 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 e a gerência dos serviços baseados em NFV, além de permitir o suporte à execução de VNFs provenientes de diferentes desenvolvedores, a ETSI vem coordenando esforços para propor uma arquitetura de referência para NFV [Quittek et al. 2014]. Esta arquitetura, denominada de NFV-MANO (NFV *Management and Orchestration*), possui os módulos para o controle e orquestração de VNFs, responsáveis pelo ciclo de vida e gerenciamento de recursos das funções virtualizadas.

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 entre VNFs, formando as 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. Para executar suas funcionalidades, o VNFM faz uso do VNF *Descriptor* (VNFD) – *template* responsável por especificar uma VNF em termos dos seus requisitos operacionais e de implantação. Além do VNFD, cada VNF possui um EM (*Element Management*), módulo responsável pelo FCAPS (*Fault, Configuration, Accounting, Performance, Security, Management*) de cada função virtualizada. 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 é responsável principalmente pela criação e remoção dos dispositivos virtuais.

Com o objetivo de prover alta disponibilidade e resiliência para as VNFs, a ETSI vem definindo requisitos para ambientes e plataformas NFV [Schöller et al. 2015, Nakamura et al. 2016]. Em particular, uma plataforma NFV deve oferecer suporte a alta disponibilidade e resiliência para VNFs de diferentes tipos e fornecidas por diferentes desenvolvedores. Além disso, espera-se que diferentes níveis de resiliência possam ser definidos, uma vez que as VNFs possuem requisitos distintos. Para garantir a alta disponibilidade, é essencial que a plataforma NFV forneça um sistema de gerenciamento de falhas completo, englobando a detecção e recuperação de falhas. Por fim, a plataforma NFV deve garantir que VNFs *stateful* preservem o seu estado interno em casos de falha.

Apesar de que várias plataformas NFV são compatíveis com a arquitetura NFV-

MANO, não há entre elas soluções que fornecem todo o conjunto necessário de funcionalidades para garantir a confiabilidade de ponta a ponta para serviços virtualizados. Dessa forma, este trabalho tem como objetivo preencher esta lacuna, propondo uma arquitetura de alta disponibilidade integrada à arquitetura NFV-MANO.

3. Uma Arquitetura de Alta Disponibilidade para NFV

O suporte para alta disponibilidade é essencial para garantir a confiabilidade de serviços virtualizados. Para ser considerada uma alternativa concreta, a tecnologia NFV deve apresentar disponibilidade no mesmo nível dos *middleboxes* implementados em hardware dedicado. Uma vez que a disponibilidade de um serviço aumenta conforme o tempo total de recuperação diminui, é necessário detectar e reagir a falhas de maneira eficiente, minimizando a indisponibilidade do serviço. No entanto, apenas solicitar recursos adicionais ou substituir os recursos existentes não é suficiente. Deve-se considerar que boa parte das VNFs são *stateful* – o estado interno da função de rede muda de acordo com os pacotes processados e a própria execução da função. Dessa forma, deve-se garantir que o estado da VNF antes da ocorrência de uma falha seja preservado.

Este trabalho propõe o NHAM (*NFV High Availability Module*): uma arquitetura de alta disponibilidade para VNFs integrada e disponibilizada como um módulo da arquitetura de referência NFV-MANO. Em especial, o NHAM possui gerenciamento de falhas eficiente, minimizando o tempo total de recuperação. O NHAM assume um sistema assíncrono, falhas do tipo *crash-recovery* de uma única VNF e canais de comunicação confiáveis. Destaca-se também que ao utilizar o NHAM, as VNFs *herdam* as propriedades de alta disponibilidade, evitando que os desenvolvedores necessitem realizar qualquer tipo de mudança no seu código interno.

A seguir é apresentada a arquitetura no contexto do NFV-MANO, após a qual é descrita a estratégia para VNFs *stateful* e então os mecanismos de resiliência.

3.1. Arquitetura Geral e Componentes Internos do NHAM

O NHAM é uma arquitetura de alta disponibilidade para VNFs, integrada com a arquitetura NFV-MANO. Para tanto, o NHAM se comunica com os módulos da arquitetura NFV-MANO como mostra a Figura 1.

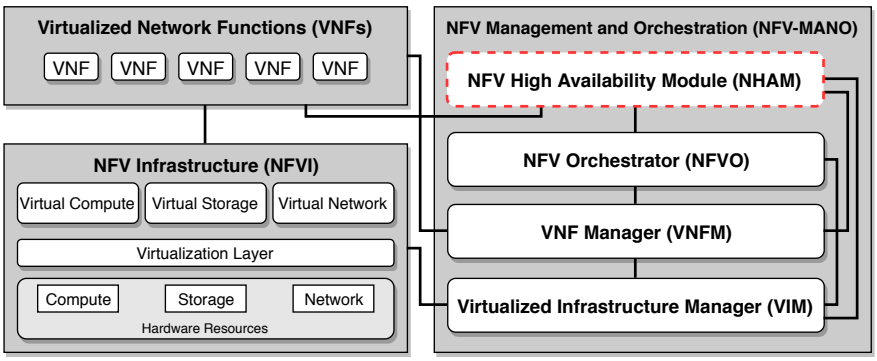


Figura 1. Arquitetura NFV-MANO e o NHAM.

Os componentes originais da arquitetura NFV-MANO possuem todas as funcionalidades necessárias para controlar o ciclo de vida dos serviços virtualizados. O NHAM se comunica com os componentes do NFV-MANO para realizar funções específicas para

garantir a alta disponibilidade. Por exemplo, durante os procedimentos de recuperação o NHAM solicita a criação de novas VNFs através do VNFM e faz requisições ao VIM para o gerenciamento dos recursos virtuais das VNFs.

A Figura 2 mostra os principais componentes da arquitetura NHAM: o Sistema de Gerenciamento de Falhas (*Fault Management System* - FMS) e o Gerenciador de Estados de VNFs (*VNF State Manager* - VSM). Ambos os componentes são descritos a seguir.

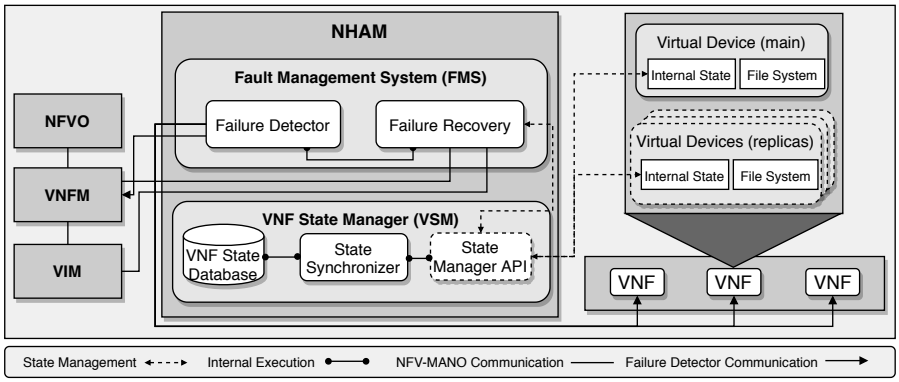


Figura 2. Componentes internos da arquitetura NHAM.

A gerência de falhas no contexto deste trabalho consiste de detecção e recuperação de falhas. Em ambientes baseados em NFV, a detecção consiste no monitoramento das VNFs. Para realizar a detecção, o NHAM possui um detector de falhas (*Failure Detector* - FD) que monitora periodicamente todas as instâncias das VNFs. Para a etapa de recuperação, o NHAM realiza requisições para o VNFM, VIM e o gerenciador de estados para garantir a execução correta da VNF. Além disso, o NHAM define quatro mecanismos de resiliência para garantir a execução da VNF mesmo na presença de falhas. Estas estratégias são descritas em detalhes na Seção 3.3.

De forma a preservar também o estado interno da VNF após a sua recuperação, o NHAM possui um gerenciador de estados (VSM). O VSM é composto pelo *State Synchronizer*, uma API para a manipulação do estado interno das VNFs e uma base de dados responsável por armazenar os estados das VNFs. A seguir o VSM é descrito em detalhes.

3.2. Gerenciando o Estado de VNFs *Stateful*

De maneira a permitir a recuperação de VNFs *stateful*, o NHAM define um gerenciador de estados (VSM). Como as VNFs executam em dispositivos virtualizados, que podem ser tanto máquinas virtuais como *containers*, copiar todo o estado do dispositivo virtual torna-se uma estratégia atrativa, uma vez que esta abordagem é genérica e não exige modificações no código interno da VNF. O *Checkpoint/Restore* é uma abordagem clássica para recuperar o estado de um sistema [Elnozahy et al. 2002]. Periodicamente, *checkpoints* do estado interno são capturados e salvos em memória não-volátil, contendo uma representação do estado do sistema (*e.g.* descritores de arquivos, mapeamento de memória). Quando uma falha é detectada, o sistema pode ser recuperado (*i.e. restore*) para um *checkpoint* anterior.

O estado de uma função de rede pode ser classificado como externo ou interno [Han et al. 2017, Kablan et al. 2017, Rajagopalan et al. 2013]. No estado externo são armazenadas informações estáticas que não variam com o tempo (*e.g.* regras de

um *firewall*). Em caso de falha, o estado externo pode ser facilmente recuperado durante a reinicialização da VNF. Por sua vez, o estado interno armazena informações que são atualizadas de acordo com o processamento de pacotes e a execução de processos. Como exemplo, processos em execução, mapeamento de memória e conexões TCP são informações do estado interno de uma função de rede. O principal desafio no gerenciamento de estados é recuperar o estado interno da VNF em caso de falha.

Neste sentido, o VSM possui como componente principal o *State Synchronizer*, que se baseia em técnicas de *Checkpoint/Restore* para gerenciar o estado interno das VNFs. O *State Synchronizer* coleta periodicamente o estado interno da VNF através de *checkpoints*. De forma a reduzir o impacto na sobrecarga causada pelas operações de sincronização, o *checkpoint* é feito de maneira assíncrona – após o estado ser capturado, a VNF continua a sua execução enquanto o *State Synchronizer* conclui o *checkpoint*.

O NHAM possui uma API (*Application Programming Interface*) para gerenciar o estado interno das VNFs de maneira transparente, sem exigir nenhuma modificação no código interno da VNF. Nesta API foram definidos um conjunto de funcionalidades que permitem o controle completo do estado interno da VNF. A API consiste de duas primitivas principais, ambas descritas a seguir.

export_vnf_state. Esta primitiva é utilizada para realizar o *checkpoint* de uma VNF específica. Para tanto, a VNF é parada momentaneamente para que um *checkpoint* do sistema seja gerado, salvando informações internas, tais como processos em execução, conexões TCP e mapeamento de memória. Note que neste intervalo os pacotes em trânsito são armazenados em *buffers* até que o *checkpoint* seja concluído. Após receber uma confirmação, o *checkpoint* é enviado para a base de dados (VNF *State Database*). Dependendo do mecanismo de resiliência adotado, o *checkpoint* pode ser enviado diretamente para as réplicas.

import_vnf_state. Esta primitiva é utilizada para restaurar um *checkpoint* em uma VNF. Nesta primitiva, três parâmetros são especificados: *(i) vnf_destination* especifica a VNF; *(ii) vnf_source*, que indica de qual VNF será importado o *checkpoint* e; *(iii) epoch*, que permite especificar um *checkpoint* específico. Se o parâmetro *epoch* não for especificado, o *State Synchronizer* utiliza o *checkpoint* mais recente. Para receber o novo estado a VNF é parada momentaneamente. A fim de verificar se a VNF de destino recebeu o *checkpoint* sem nenhum erro, é calculado o *hash* do *checkpoint*. Caso ocorra alguma inconsistência, uma mensagem de erro é retornada para o *State Synchronizer*. Caso contrário, a VNF de destino confirma a importação do novo *checkpoint*.

3.3. Mecanismos de Resiliência

Para tornar as VNFs altamente disponíveis é necessário definir estratégias para garantir a disponibilidade e continuidade do serviço. Entretanto, diferentes funções de rede podem ter requisitos distintos de resiliência [Schöller et al. 2015]. Por exemplo, funções de rede que processam tráfego em tempo real possuem requisitos mais restritos de resiliência do que funções que tenham menor prioridade no tráfego. Dessa forma, é essencial que a plataforma NFV ofereça suporte a diferentes níveis de resiliência.

Ao definir os requisitos de resiliência é possível classificar três diferentes métodos [Nakamura et al. 2016]: *Active-Standby*, *Active-Active* e *Load Balancing*. No método *Active-Standby*, a VNF possui uma réplica que já está instanciada, porém está em um estado de espera (*standby*). No método *Active-Active*, a réplica da VNF está instanciada

e ativa, recebendo periodicamente atualizações do estado interno da VNF principal. Por fim, o *Load Balancing* é uma generalização do método *Active-Active*, no qual a VNF possui N réplicas ativas. O termo "Load Balancing" foi proposto neste contexto pela própria ETSI [Schöller et al. 2015].

A arquitetura de alta disponibilidade do NHAM define quatro mecanismos de resiliência. Estes mecanismos possuem diferentes custos computacionais e tempo de recuperação, adaptando-se a diferentes circunstâncias e tipos de VNFs. Além disso, as estratégias possuem procedimentos de recuperação diferentes, descritos em detalhes na Seção 4. Cada mecanismo de resiliência é descrito a seguir.

Sem redundância (0R). Como o nome sugere, este mecanismo não emprega nenhum tipo de redundância. Neste caso, o *State Synchronizer* apenas exporta periodicamente o *checkpoint* da VNF para a base de dados. Esta abordagem é a mais simples e com menor utilização de recursos computacionais. Por outro lado, em caso de falha o tempo de recuperação será maior do que o dos outros métodos, uma vez que uma nova instância ainda precisa ser criada.

Réplica primária *Active-Standby* (1R-AS). Este mecanismo de resiliência utiliza o método *Active-Standby*. Assim como no mecanismo 0R, o *State Synchronizer* exporta periodicamente o *checkpoint* da VNF para a base de dados. Em caso de falha a réplica já está criada, sendo necessário apenas importar o *checkpoint* mais recente da VNF falha. Esta opção é mais custosa que o mecanismo 0R pois consome recursos da infraestrutura virtualizada para manter a réplica, porém possui menor tempo de recuperação.

Réplica primária *Active-Active* (1R-AA). Neste mecanismo o *State Synchronizer* atualiza periodicamente a réplica com o *checkpoint* da VNF. Dessa forma, a réplica é constantemente atualizada com o estado da VNF. Em caso de falha, o NHAM apenas indica que a réplica é agora a VNF principal. Naturalmente, o mecanismo 1R-AA é mais custoso que o mecanismo 1R-AS, porém o tempo de recuperação é ainda menor.

Múltiplas réplicas *Load Balancing* (NR-LB). Este caso é a generalização do mecanismo 1R-AA. A VNF possui N réplicas que são constantemente sincronizadas com a VNF principal. Além disso, o usuário pode acessar qualquer uma das réplicas deste grupo. Em caso de falha, este mecanismo não necessita de reconfiguração, uma vez que o usuário utiliza outra VNF do grupo. Esta opção é a mais custosa de todas, pois exige a sincronização de N réplicas, porém é a que possui o menor tempo de inatividade.

Os diferentes mecanismos de resiliência podem ser comparados entre si com base na relação entre a utilização de recursos computacionais e o tempo de recuperação. A escolha de qual mecanismo utilizar depende dos requisitos de resiliência definidos pela VNF. A Tabela 1 apresenta uma comparação entre as diferentes estratégias de resiliência.

Tabela 1. Comparação entre as diferentes estratégias de recuperação.

Mecanismo de Resiliência	Método	Réplicas	Base de Dados	Tempo de Recuperação	Custo Computacional	Reconfiguração
0R	Sem redundância	0	Sim	Muito alto	Muito baixo	Sim
1R-AS	*Active-Standby*	1	Sim	Médio	Baixo	Sim
1R-AA	*Active-Active*	1	Não	Baixo	Alto	Sim
NR-LB	*Load balancing*	N	Não	Muito baixo	Muito alto	Não

A estratégia 0R possui o menor custo computacional apesar de apresentar o maior tempo de recuperação, ideais para VNFs que podem tolerar um maior tempo de inatividade. Já a abordagem 1R-AS possui menor tempo de recuperação com relação ao 0R,

ideal para servidores que possuem alta capacidade de disco, uma vez que a réplica em *standby* não consome recursos de computação ou de rede. Já as estratégias 1R-AA e NR-LB possuem o menor tempo de recuperação, pois o estado das réplicas é constantemente atualizado. Apesar de consumirem mais recursos, elas possuem o menor tempo de recuperação, garantindo altas disponibilidade para as VNFs.

4. Sistema de Gerenciamento de Falhas

O NHAM possui um sistema de gerenciamento de falhas para detectar e recuperar falhas, bem como realizar a reconfiguração das VNFs. O primeiro passo é detectar que a VNF está falha, identificando também o local em que a VNF está executando. Uma vez que a falha é localizada, inicia-se o processo de recuperação, que consiste em substituir a VNF falha por outra sem falha. Por fim, é necessário realizar procedimentos de reconfiguração para refletir as mudanças da nova VNF no NFV-MANO, além de criar novas réplicas para substituir as que foram utilizadas durante a recuperação. Assume-se que os protocolos de rede utilizando a VNF são capazes de lidar com a perda e duplicação de pacotes que podem ocorrer entre a falha e a recuperação de uma VNF. A forma como cada componente do FMS do NHAM funciona é descrita a seguir.

A detecção de falhas no NHAM é feita a partir de um FD baseado em *polling* – mensagens enviadas periodicamente para as VNFs monitoradas. Nesta abordagem o EM de cada VNF é responsável por processar as mensagens enviadas pelo NHAM. Para cada mensagem enviada, o FD calcula um intervalo de *timeout*. Se a VNF não responder a mensagem dentro deste intervalo, o FD adiciona esta VNF em uma lista de suspeitos. Neste caso, o FD envia uma mensagem para o VNFM, notificando que a VNF está em processo de recuperação. Após a detecção, inicia-se o processo de recuperação. Estes processos são diferentes para cada mecanismo de resiliência empregado pelo NHAM. Os procedimentos de cada abordagem são descritos a seguir.

0R. Como o mecanismo 0R não utiliza nenhuma réplica, o processo de recuperação exige a criação de uma nova VNF, substituindo aquela que falhou. Em seguida, o *State Synchronizer* importa o último *checkpoint* da VNF que falhou na nova VNF, preservando assim o seu estado interno. Após, o processo de reconfiguração é iniciado, atualizando as informações da VNF recém criada (*e.g.* endereços IP, identificadores). Por fim, o NHAM envia mensagens para o NFV-MANO notificando que uma VNF foi substituída e que as informações correspondentes podem ser atualizadas.

1R-AS. Neste mecanismo, como a VNF já possui uma réplica em *standby*, o *State Synchronizer* apenas importa o último *checkpoint* da VNF para a réplica, atualizando assim o seu estado interno. Para a reconfiguração, a réplica se torna a VNF principal e o NHAM envia uma requisição para o NFV-MANO atualizando as informações necessárias. Por fim, uma nova réplica é criada em modo *standby* para substituir a réplica utilizada.

1R-AA. Neste mecanismo, como a VNF possui uma réplica que é periodicamente atualizada, não é necessário realizar nenhum procedimento relacionado à recuperação. Dessa forma, basta realizar a reconfiguração, de forma semelhante aos demais mecanismos. Por fim, uma nova réplica é criada.

NR-LB. O mecanismo NR-LB consiste de um grupo de réplicas que são periodicamente sincronizadas. Neste caso, como o usuário pode acessar qualquer uma das réplicas, não existe a necessidade de reconfiguração. No entanto, é possível especificar que o NHAM mantenha um número mínimo de réplicas no grupo. Dessa forma, se o

número de falhas exceder o limite mínimo, o NHAM cria novas réplicas.

O *State Synchronizer* é utilizado como componente da arquitetura responsável por garantir que não existam inconsistências ao atualizar um grupo de réplicas. Para tanto, o *State Synchronizer* aguarda uma confirmação de que cada réplica recebeu o estado atualizado. Só após receber N confirmações, o *State Synchronizer* realiza a próxima sincronização. Para não aguardar indefinidamente por uma resposta de uma réplica que possivelmente falhou, o *State Synchronizer* faz monitoramento baseado em *timeouts*.

Além disso, o NHAM garante a consistência dos estados das réplicas em outras duas situações específicas. A primeira é quando uma VNF é incorretamente suspeita de ter falhado. Neste caso, o NHAM simplesmente executa o procedimento de recuperação nesta VNF. O segundo caso é quando uma VNF falha enquanto o *State Synchronizer* está atualizando as réplicas. Neste caso, um *rollback* é realizado em todas as réplicas do grupo para o seu estado antes da atualização. A VNF falha é então removida do grupo e apenas na atualização seguinte as réplicas corretas recebem o *checkpoint* atualizado.

5. Implementação e Resultados Experimentais

Como prova de conceito, um protótipo do NHAM foi implementado e utilizado para avaliar o desempenho e disponibilidade da solução. As próximas subseções apresentam os detalhes de implementação e os resultados experimentais.

5.1. Implementação

A implementação da arquitetura NHAM é modular e integrada como um módulo da arquitetura de referência NFV-MANO. Dessa forma, o NHAM é facilmente integrado a qualquer plataforma NFV compatível com as especificações da ETSI. Os requisitos de disponibilidade de cada VNF são definidos no VNFD, especificando o tipo de mecanismo de resiliência adotado, o intervalo de *checkpoints* e a quantidade de réplicas.

O protótipo do NHAM é baseado em *containers* da plataforma Docker [Merkel 2014], uma vez que eles possuem baixo tempo de inicialização, além de apresentar pouca utilização de memória. Para prover a funcionalidade de *Checkpoint/Restore* em *containers*, o Docker possui um *plugin* que utiliza o CRIU (*Checkpoint/Restore in Userspace*) – implementação da funcionalidade de *Checkpoint/Restore* para sistemas baseados em Linux [CRIU 2019]. Note que o NHAM não é limitado a virtualização baseada em *containers*. Para VNFs baseadas em máquina virtual, ferramentas como o *Micro-Checkpointing*[1], e *live migration*[2], podem ser adotadas para salvar *checkpoints* da VNF.

5.2. Resultados Experimentais

O desempenho do NHAM foi avaliado em termos do tempo de recuperação, desempenho das estratégias de recuperação e a disponibilidade atingida pelas VNFs instanciadas com o suporte oferecido pelo NHAM. Cada experimento foi executado 10 vezes e a média dos resultados é apresentada com um intervalo de confiança de 95%.

O protótipo foi executado em um servidor com 32 processadores Intel Core i7@2.50GHz com 4 núcleos, 48 GB de RAM e Ubuntu 18.04. Cada VNF é composta por um Ubuntu *server* com 512 MB de RAM e 1 CPU executando uma função de encaminhamento (*forwarder*). Por padrão, o intervalo de *checkpoint* é de 500ms. Por fim, no mecanismo NR-LB são utilizadas, por padrão, 5 réplicas.

[1] `https://wiki.qemu.org/Features/MicroCheckpointing`
[2] `https://www.linux-kvm.org/page/Migration`

5.2.1. Tempo de Recuperação de Falhas

O baixo tempo na recuperação das falhas tem particular importância para garantir a alta disponibilidade das VNFs. A Figura 3 mede o tempo total para a recuperação das falhas, desde a detecção até a reconfiguração. As falhas foram injetadas através de *scripts* que removem todas as conexões das VNFs. Dessa forma, as mensagens de monitoramento não são recebidas pela VNF, consequentemente apontando para a suspeita de falha.

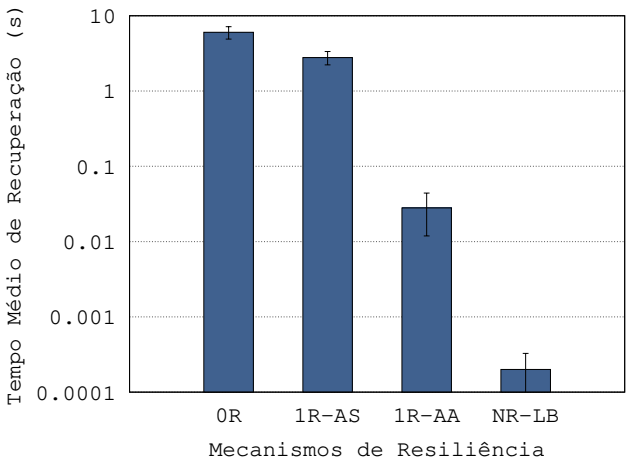


Figura 3. Tempo de recuperação para os diferentes mecanismos de resiliência.

Como previsto, o mecanismo 0R possui o maior tempo de recuperação, alcançando aproximadamente 6s. Nesse intervalo de tempo uma nova VNF foi instanciada e o VSM realizou a restauração do último *checkpoint* da VNF. O mecanismo 1R-AS apresenta resultados melhores, uma vez que a VNF já possui uma réplica instanciada. Por utilizar o método *Active-Standby*, é necessário apenas importar o *checkpoint* da VNF que falhou, totalizando um tempo de recuperação de 2.7s.

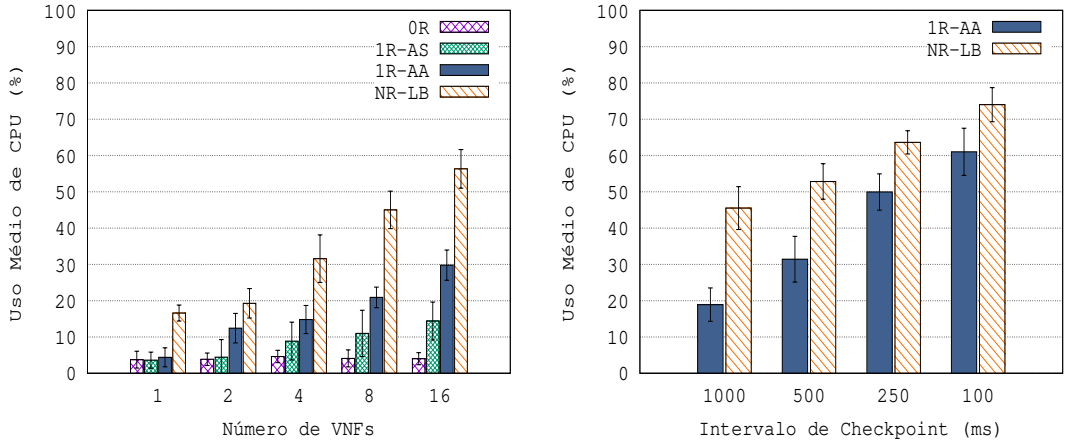
Os mecanismos baseados no método *Active-Active* possuem resultados mais adequados para VNFs com altos requisitos de disponibilidade. O mecanismo 1R-AA atingiu 0.02s no tempo total de recuperação, enquanto que o mecanismo NR-LB atingiu apenas 0.0002s, tempo necessário para o NHAM atualizar o grupo de réplicas.

5.2.2. Avaliando o Desempenho dos Mecanismos de Resiliência

O experimento anterior mostrou que as diferentes estratégias possuem vasta diferença com relação ao tempo de recuperação. Dependendo do mecanismo adotado, os custos computacionais são diferentes, em especial a utilização de CPU dos componentes do NHAM. A Figura 4(a) mostra a comparação na utilização de CPU de acordo com o mecanismo de resiliência adotado, conforme o número de VNFs varia de 1 a 16.

O mecanismo 0R possui maior tempo de recuperação em troca de um melhor desempenho. É possível perceber que o mecanismo 0R é facilmente escalável com relação ao número de VNFs, uma vez que a sua utilização de CPU permanece constante. Já o mecanismo 1R-AS apresenta maior utilização de recursos computacionais, aumentando em até 4 vezes o uso de CPU (14.4%).

Como previsto, os mecanismos baseados no método *Active-Active* possuem maior utilização de recursos, uma vez que estão constantemente sincronizando o estado de suas



(a) Desempenho de cada mecanismo de resiliência variando a quantidade de VNFs.

(b) Desempenho dos mecanismos baseados em *Active-Active* variando o intervalo de *checkpoints*.

Figura 4. Utilização de CPU dos mecanismos de resiliência.

réplicas. O mecanismo 1R-AA apresenta uso de CPU de até 30%, enquanto que o mecanismo NR-LB chega a 56% para sincronizar as suas réplicas.

Quanto menor o intervalo de *checkpoint*, menor a perda de pacotes em caso de falha. A Figura 4(b) apresenta o custo computacional conforme o intervalo de *checkpoints* diminui, uma vez que os mecanismos 1R-AA e NR-LB apresentaram os menores tempos de recuperação. Neste experimento, 16 VNFs foram utilizadas enquanto o intervalo de *checkpoint* varia de 1000ms até 100ms. Conforme aumenta a frequência em que as réplicas são sincronizadas, aumenta também a utilização de CPU. Em especial, para um intervalo de 100ms, a utilização de CPU aumenta até 79% para o mecanismo NR-LB.

5.2.3. Avaliando a Disponibilidade das VNFs

O tempo médio entre a ocorrência de falhas (*Mean Time Between Failures* - MTBF) é uma medida utilizada para especificar o tempo previsto até que uma falha aconteça [Kshemkalyani and Singhal 2011]. Para avaliar a disponibilidade das VNFs com o suporte do NHAM, este experimento injeta falhas consecutivas a cada intervalo de tempo. Por exemplo, um MTBF de 300 indica que uma falha ocorre a cada 5 minutos. A Tabela 2 apresenta os resultados de disponibilidade para cada mecanismo de resiliência, variando o MTBF. Cada experimento durou 3 horas e o MTBF indica a frequência em que as falhas ocorrem. Por exemplo, um MTBF de 60 indica que 180 falhas ocorreram.

Tabela 2. Disponibilidade das VNFs de acordo com o MTBF.

MTBF	Availability (%)			
	0R	1R-AS	1R-AA	NR-LB
60 (1 min)	90.869	95.563	99.952	99.999
300 (5 min)	98.029	99.080	99.990	99.999
600 (10 min)	99.005	99.537	99.995	99.999
900 (15 min)	99.334	99.691	99.996	99.999
1200 (20 min)	99.500	99.768	99.997	99.999
1500 (25 min)	99.599	99.814	99.998	99.999
1800 (30 min)	99.666	99.845	99.998	99.999

O mecanismo 0R apresenta resultado insatisfatório para VNFs que exigem alta

disponibilidade, uma vez que atingiu apenas 99.666% de disponibilidade para o maior MTBF. O mecanismo 1R-AS também não atinge os níveis de disponibilidade necessários, apesar de obter resultados melhores que o 0R.

O mecanismo 1R-AA apresenta resultados melhores, mesmo no ambiente mais propenso a falhas. As VNFs alcançaram 99.9% de disponibilidade com um MTBF de 60 e 99.99% a partir de um MTBF de 300. Já o mecanismo NR-LB possui os melhores resultados, atingindo 99.999% disponibilidade em todos os casos.

Em geral, as plataformas em nuvem utilizadas para ambientes NFV oferecem uma disponibilidade de aproximadamente 99.9% [Han et al. 2017]. Os resultados desta seção permitem concluir que o NHAM consegue garantir a alta disponibilidade das VNFs através dos mecanismos baseados no método *Active-Active*, melhorando as taxas de disponibilidade das plataformas em nuvem disponíveis atualmente.

6. Trabalhos Relacionados

Em [Kablan et al. 2017] é proposta uma arquitetura que separa o estado interno das VNFs do seu processamento. Em casos de falha, a VNF recupera o estado atualizado de um banco de dados distribuído. Entretanto, a proposta não introduz nenhum tipo de redundância para as VNFs, prejudicando o tempo de recuperação em casos de falha. Além disso, as extensas mudanças na arquitetura interna das funções limitam a solução, além de dificultar a integração com outras plataformas.

REINFORCE é um *framework* proposto em [Kulkarni et al. 2018] para oferecer suporte à resiliência através de *Checkpoint/Restore*. Similar ao NHAM, a sincronização do REINFORCE é assíncrona. Enquanto o NHAM fornece diferentes mecanismos de resiliência, o REINFORCE utiliza uma única técnica que envia periodicamente o estado para uma réplica *standby*. Por fim, o REINFORCE não é compatível com a arquitetura NFV-MANO, dificultando a sua integração com as plataformas NFV existentes.

Remus [Cully et al. 2008] é um sistema para prover alta disponibilidade em hardware genérico. O Remus envia periodicamente *checkpoints* de uma máquina virtual para um *backup*. Os pacotes são armazenados em *buffers* até que a sincronização de um novo estado seja finalizada. Apesar da diminuição no número de pacotes perdidos em caso de falha, a latência é prejudicada em situações em que não há falhas. O NHAM não interfere no processamento dos pacotes e o desempenho é mantido quando não há falhas.

Fault-Tolerant Middlebox (FTMB) [Sherry et al. 2015] é um sistema baseado em *rollback-recovery* para salvar o estado de *middleboxes* através de mecanismos que reproduzem as entradas no sistema em caso de falha. Além disso, o FTMB garante a ordem em que estas entradas são reproduzidas, visto que os pacotes possuem dependências entre si. No entanto, o FTMB exige extensas modificações no código fonte da VNF, o que representa uma grande limitação.

Em [Gember-Jacobson et al. 2014] uma arquitetura é proposta para garantir a sincronização do estado interno das VNFs. A arquitetura gerencia o estado e minimiza a perda de informação através da transferência dos fluxos de pacotes pelo controlador. Assim como as demais soluções, a solução exige mudanças no código interno da VNF.

No trabalho proposto em [Li et al. 2015], os autores utilizam *middlewares* que realizam periodicamente o *checkpoint* do estado interno das aplicações. Em caso de falha o *middleware* tenta reinicializar o *container*, o que pode comprometer o tempo de

recuperação caso essa abordagem não seja suficiente. Ao contrário do NHAM, os autores não propõem estratégias para diminuir a sobrecarga da sincronização dos estados, ou diferentes níveis de resiliência para diferentes tipos de VNFs.

Além dos trabalhos descritos acima, a maior parte das plataformas NFV e de nuvem, tais como o OpenStack [OpenStack 2019] e Kubernetes [Bernstein 2014], oferecem suporte a algum tipo de tolerância a falhas. Entretanto, estas soluções não conseguem garantir a disponibilidade de VNFs *stateful*, uma vez que não possuem mecanismos para preservar o estado interno das funções.

Os trabalhos apresentados oferecem apenas parte do suporte necessário para garantir a alta disponibilidade de VNFs *stateful*. Em particular, existem três grandes desvantagens nas soluções atuais. A primeira é a falta de suporte a diferentes níveis de resiliência, de forma a adaptar-se a diferentes requisitos de disponibilidade. A segunda é a necessidade em mudar o código interno das VNFs, limitando os tipos de VNFs que podem ser utilizadas. Por fim, nenhuma solução é totalmente compatível com a arquitetura de referência NFV-MANO, dificultando a integração com as demais plataformas NFV.

7. Conclusão

Apesar dos diversos avanços em NFV, a confiabilidade das VNFs é essencial para a sua ampla adoção em larga escala. Este trabalho propõe o NHAM: uma arquitetura integrada à arquitetura de referência NFV-MANO para garantir a alta disponibilidade de VNFs. O NHAM possui um sistema de gerenciamento de falhas do tipo *crash-recovery* para uma única VNF, detecção e recuperação de falhas, além da reconfiguração das VNFs. Além disso, o NHAM possui um gerenciador de estados que salva periodicamente o estado interno das VNFs *stateful* através de *Checkpoint/Restore* sem a necessidade de mudanças no código interno das VNFs. O NHAM define múltiplas estratégias de resiliência, adaptando-se a diferentes tipos de VNFs. Resultados experimentais mostram que o NHAM consegue garantir a alta disponibilidade dos serviços virtualizados.

Trabalhos futuros incluem a criação de estratégias para garantir o estado consistente de uma cadeia de VNFs (*i.e.* SFCs) em casos de falha. Planeja-se também a definição de mecanismos para tornar a própria arquitetura NFV-MANO tolerante a falhas, *i.e.* permitindo a falha dos componentes do NFV-MANO, além de estender os mecanismos de resiliência do NHAM, permitindo a tolerância a falhas no nível de infraestrutura. Por fim, outro trabalho promissor é analisar estratégias concretas para melhorar a prevenção e predição de falhas.

Referências

Bernstein, D. (2014). Containers and cloud: From lxc to docker to kubernetes. *IEEE Cloud Computing*, 1(3):81–84.

Bondan, L., Franco, M. F., Marcuzzo, L., Venancio, G., Santos, R. L., Pfitscher, R. J., Scheid, E. J., Stiller, B., De Turck, F., Duarte, E. P., et al. (2019). Fende: Marketplace-based distribution, execution, and life cycle management of vnfs. *IEEE Communications Magazine*, 57(1):13–19.

Chiosi, M., Clarke, D., Willis, P., Reid, A., Feger, J., Bugenhagen, M., Khan, W., Fargano, M., Cui, C., Deng, H., et al. (2012). Network functions virtualisation: An introduction, benefits, enablers, challenges and call for action. In *SDN and OpenFlow World Congress*, pages 22–24.

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

CRIU (2019). Checkpoint/Restore In Userspace. `https://criu.org/`. Dezembro de 2019.

Cully, B., Lefebvre, G., Meyer, D., Feeley, M., Hutchinson, N., and Warfield, A. (2008). Remus: High availability via asynchronous virtual machine replication. In *Proceedings of the 5th USENIX Symposium on Networked Systems Design and Implementation*, pages 161–174. San Francisco.

Elnozahy, E. N., Alvisi, L., Wang, Y.-M., and Johnson, D. B. (2002). A survey of rollback-recovery protocols in message-passing systems. *ACM Computing Surveys (CSUR)*, 34(3):375–408.

Gember-Jacobson, A., Viswanathan, R., Prakash, C., Grandl, R., Khalid, J., Das, S., and Akella, A. (2014). Opennf: Enabling innovation in network function control. In *ACM SIGCOMM Computer Communication Review*, pages 163–174. ACM.

Gray, J. and Siewiorek, D. P. (1991). High-availability computer systems. *Computer*, 24(9):39–48.

Han, B., Gopalakrishnan, V., Ji, L., and Lee, S. (2015). Network function virtualization: Challenges and opportunities for innovations. *IEEE Communications Magazine*, 53(2):90–97.

Han, B., Gopalakrishnan, V., Kathirvel, G., and Shaikh, A. (2017). On the resiliency of virtual network functions. *IEEE Communications Magazine*, 55(7):152–157.

Kablan, M., Alsudais, A., Keller, E., and Le, F. (2017). Stateless network functions: Breaking the tight coupling of state and processing. In *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, pages 97–112.

Kshemkalyani, A. D. and Singhal, M. (2011). *Distributed computing: principles, algorithms, and systems*. Cambridge University Press.

Kulkarni, S. G., Liu, G., Ramakrishnan, K., Arumaithurai, M., Wood, T., and Fu, X. (2018). Reinforce: Achieving efficient failure resiliency for network function virtualization based services. In *Proceedings of the 14th International Conference on emerging Networking EXperiments and Technologies*, pages 41–53. ACM.

Li, W., Kanso, A., and Gherbi, A. (2015). Leveraging linux containers to achieve high availability for cloud services. In *2015 IEEE International Conference on Cloud Engineering*, pages 76–83. IEEE.

Merkel, D. (2014). Docker: lightweight linux containers for consistent development and deployment. *Linux Journal*, 2014(239):2.

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.

Nakamura, H., Adams, R., and et al (2016). Network Functions Virtualisation (NFV); Reliability; Report on Models and Features for End-to-End Reliability. GS NFV-REL 003 V1.1.1. Technical report, ETSI.

OpenStack (2019). OpenStack - open source software for creating private and public clouds.

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. Technical report, ETSI.

Rajagopalan, S., Williams, D., Jamjoom, H., and Warfield, A. (2013). Split/merge: System support for elastic execution in virtual middleboxes. In *Presented as part of the 10th USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, pages 227–240.

Schöller, M., Khan, N., and et al (2015). Network Function Virtualisation (NFV); Resiliency Requirements. GS NFV-REL 001 V1.1.1. Technical report, ETSI.

Sherry, J., Gao, P. X., Basu, S., Panda, A., Krishnamurthy, A., Maciocco, C., Manesh, M., Martins, J., Ratnasamy, S., Rizzo, L., et al. (2015). Rollback-recovery for middleboxes. In *ACM SIGCOMM Computer Communication Review*, pages 227–240. ACM.