# Uma Arquitetura de Alta Disponibilidade para Serviços Virtualizados de Rede

**Giovanni Venâncio, Elias P. Duarte Jr.**

Universidade Federal do Paraná – Depto. Informática – Curitiba, PR

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

***Abstract.*** *Network Functions Virtualization (NFV) allows the implementation in software of services that run in the network core. Complex services can be formed by composing multiple VNFs (Virtual Network Functions) in SFCs (Service Function Chains). It is essential to guarantee the reliability of these services, since they are responsible for critical tasks. This work proposes the NHAM (NFV High Availability Module): a high availability architecture for virtualized network services. NHAM is proposed as a module of the NFV-MANO architecture and guarantees the continuous availability of stateful SFCs. NHAM performs fault management and offers a choice of different recovery mechanisms which are applied according to specific service requirements. The high-availability strategy combines SFC buffer management with Checkpoint/Restore. A prototype was implemented and experimental results are presented.*

***Resumo.*** *A Virtualização de Funções de Rede (NFV) permite a implementação em software de serviços diversos que executam no núcleo da rede. Serviços complexos podem ser formados a partir da composição de múltiplas VNFs (Virtual Network Functions) em SFCs (Service Function Chains). É imprescindível garantir a confiabilidade destes serviços, uma vez que eles são responsáveis por tarefas críticas na rede. Este trabalho propõe o NHAM (NFV High Availability Module): uma arquitetura de alta disponibilidade para serviços virtualizados de rede. O NHAM é integrado como um módulo da arquitetura NFV-MANO e garante o funcionamento correto e contínuo de SFCs stateful. O NHAM realiza o gerenciamento de falhas e permite a escolha de diferentes mecanismos de recuperação, aplicados de acordo com as necessidades dos serviços. A estratégia de alta disponibilidade combina o gerenciamento de buffers das SFCs em conjunto com técnicas de Checkpoint/Restore. Um protótipo foi implementado e resultados experimentais são apresentados.*

## 1. Introdução

A virtualização é uma das tecnologias mais importantes que podem levar a uma solução para o problema da "Ossificação da Internet": quando protocolos da Internet foram desenvolvidos há décadas, era difícil prever o crescimento extraordinário que ocorreria nas redes desde então. As tecnologias de virtualização permitem que a rede torne-se programável, garantindo a possibilidade de evolução para múltiplas direções. A tecnologia NFV (*Network Function Virtualization*) permite a implementação de serviços diversos no núcleo da rede em software, que é executado em hardware de prateleira [Mijumbi et al. 2016]. Os serviços de rede complexos podem ser implementados através

da orquestração de *Virtualized Network Functions* (VNFs) em *Service Function Chains* (SFCs). Em uma SFC, VNFs são conectadas em uma ordem predefinida pela qual o tráfego é direcionado [Halpern and Pignataro 2015]. A tecnologia NFV amplia o mercado de desenvolvedores de serviços de rede, antes limitados a um número pequeno de fornecedores, agora podendo inclusive ser obtidos em *Marketplaces* da Internet [Bondan et al. 2019].

Apesar das diversas vantagens, é inegável que os serviços virtualizados de rede estão mais sujeitos a falhas do que as alternativas tradicionais construídas com hardware especializado [Han et al. 2017, Cotroneo et al. 2014, Mijumbi et al. 2016]. Além disso, serviços do núcleo da rede têm requisitos estritos de resiliência. O nível de disponibilidade dos sistemas comerciais de telecomunicações é de *five nines* (99.999%), o que corresponde a um pouco mais de 5 minutos de inatividade por ano. A disponibilidade dos serviços virtualizados de rede têm sido pouco abordada pelos desenvolvedores NFV [Han et al. 2017]. Os desafios encontrados para manter níveis altos de disponibilidade aumentam ao considerar que grande parte das VNFs são *stateful*: seu estado interno muda de acordo com o tráfego processado pela função.

Este trabalho propõe uma arquitetura de alta disponibilidade para serviços virtualizados de rede baseados em NFV. A arquitetura, denominada NHAM (NFV *High Availability Module*), é integrada como um módulo da arquitetura de referência NFV-MANO (NFV *Management and Orchestration*) [Quittek et al. 2014], garantindo o funcionamento correto e contínuo de VNFs e SFCs. A estratégia adotada pelo NHAM é definida no nível de virtualização, de forma que qualquer serviço possa herdar as propriedades de alta disponibilidade e resiliência de maneira transparente. Além disso, o NHAM permite a recuperação consistente de serviços *stateful* – todo o estado do serviço é preservado após a ocorrência de uma ou múltiplas falhas. Destaca-se também que não conhecemos outra solução totalmente compatível com a arquitetura de referência NFV-MANO.

Para garantir a confiabilidade e alta disponibilidade ponto-a-ponto dos serviços virtualizados, uma estratégia é proposta para construir e gerenciar SFCs resilientes. A estratégia combina o uso de *checkpoint/restore* [Elnozahy et al. 2002] para preservar o estado interno de cada VNF com o gerenciamento de *buffers* das SFCs. Esta estratégia possibilita a sincronização do tráfego processado por cada VNF da SFC com os *checkpoints* correspondentes. Consequentemente, o NHAM garante a recuperação completa e correta dos serviços, tolerando inclusive múltiplas falhas simultâneas de múltiplas VNFs da SFC. Essa estratégia evita também perdas e duplicações de pacotes em decorrência das falhas. O NHAM fornece um cardápio de funcionalidades e estratégias para garantir a alta disponibilidade dos serviços NFV. Além de prover o gerenciamento de falhas, o NHAM permite que o operador da rede escolha, para cada VNF que compõe o serviço, diferentes mecanismos de resiliência, de acordo com os requisitos de cada função.

Um protótipo do NHAM foi implementado e experimentos foram executados para avaliar o desempenho e a disponibilidade dos serviços de rede tolerantes a falhas. Os experimentos demonstram que, dependendo da estratégia e dos parâmetros adotados, é possível atingir os níveis de disponibilidade dos sistemas comerciais de telecomunicações.

O restante deste trabalho está organizado da seguinte forma. A Seção 2 apresenta uma visão geral da tecnologia NFV incluindo as SFCs. O NHAM é apresentado na Seção 3, enquanto que a estratégia de tolerância a falhas para SFCs é descrita na Seção 4. A

Seção 5 apresenta a implementação e os experimentos. Por fim, a Seção 6 descreve os trabalhos relacionados e a Seção 7 conclui o trabalho.

## 2. *Network Function Virtualization*: Uma Visão Geral

O paradigma NFV foi proposto como uma alternativa baseada em software para implementar serviços de rede tradicionalmente implementados em hardware dedicado como *middleboxes*. O objetivo é melhorar a flexibilidade, simplificar o projeto, desenvolvimento e gerenciamento de serviços de rede [Mijumbi et al. 2016]. Através de tecnologias de virtualização, serviços de rede passam a ser implementados como VNFs, executadas em hardware genérico. A fim de padronizar a execução e gerenciamento de serviços baseados em NFV, o ETSI propôs a arquitetura de referência NFV-MANO (NFV *Management and Orchestration*) [Quittek et al. 2014]. Esta arquitetura inclui módulos de controle e orquestração de VNFs, responsáveis pelo ciclo de vida e gerenciamento de recursos de serviços virtualizados. A Figura 1 ilustra a arquitetura NFV-MANO, que também inclui a NFVI (NFV *Infrastructure*) e as próprias VNFs.

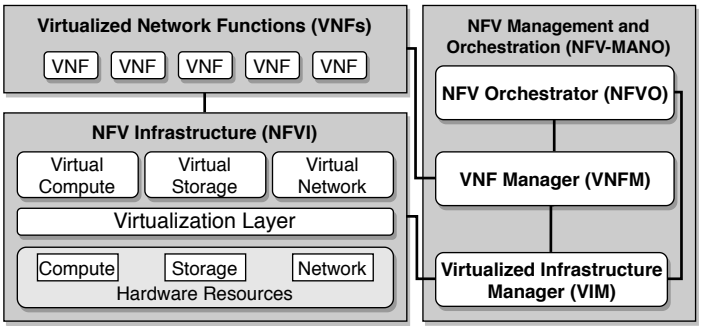



**Figura 1. Arquitetura de referência NFV-MANO.**

A NFVI corresponde à infraestrutura virtualizada na qual as VNFs são instanciadas, gerenciadas e executadas. A NFVI inclui o armazenamento físico e os recursos computacionais e de rede, abstraídos por meio de uma camada de virtualização. As VNFs na Figura 1 representam as instâncias de cada VNF em execução na NFVI.

O NFV-MANO por sua vez é organizado em três módulos principais. O primeiro módulo é o NFV *Orchestrator* (NFVO), responsável por gerenciar o ciclo de vida dos serviços de rede, permitindo a composição de VNFs em SFCs. O segundo módulo é o VNF *Manager* (VNFM), responsável por gerenciar o ciclo de vida das VNFs, incluindo a instanciação, remoção e configuração das VNFs. Por fim, o *Virtualized Infrastructure Manager* (VIM) gerencia os recursos de computação da NFVI, incluindo dispositivos virtuais.

Enquanto que VNFs executam funções específicas, elas podem ser orquestradas para compor serviços de rede complexos, as SFCs. Uma SFC consiste de várias VNFs conectadas em uma ordem predefinida através da qual o tráfego é direcionado [Halpern and Pignataro 2015]. De acordo com o IETF (*Internet Engineering Task Force*), a arquitetura de uma SFC (ilustrada na Figura 2) consiste de *Classifiers*, *Service Function Forwarders* (SFFs), além das próprias VNFs. Cada componente é descrito a seguir.

O *Classifier* é responsável por receber o tráfego da rede como entrada e determina para qual SFC o tráfego deve ser encaminhado. O caminho ao longo do qual o tráfego

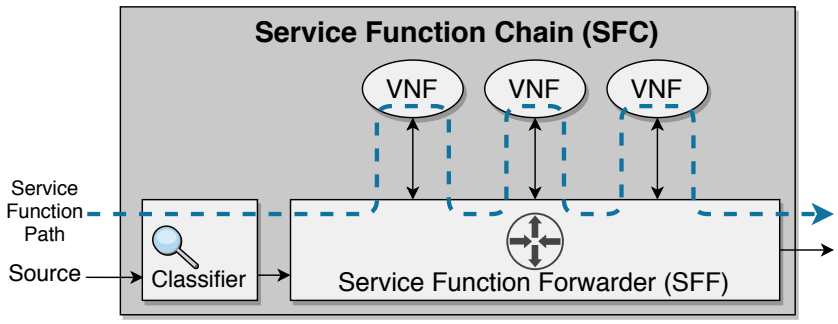



**Figura 2. Arquitetura interna de uma SFC.**

passa dentro de uma SFC é denominado de *Service Function Path* (SFP). Uma vez classificado, o tráfego é encapsulado para que possa ser roteado para o SFP correto. O SFF é responsável por encaminhar pacotes do *Classifier* para as VNFs em uma ordem predefinida, de acordo com as informações de encapsulamento. Depois que uma VNF processa o tráfego de entrada, os pacotes processados são encaminhados de volta ao SFF. Em seguida, o SFF encaminha o tráfego para a próxima VNF do SFP, e este processo é repetido até que o tráfego chegue ao destino final.

Documentos do ETSI especificam vários requisitos de resiliência para ambientes e plataformas NFV [Schöller et al. 2015, Nakamura et al. 2016]. Em particular, uma plataforma NFV deve oferecer suporte à alta disponibilidade e resiliência para VNFs de diferentes tipos e fornecidos por diferentes fornecedores. Além disso, espera-se que diferentes níveis de resiliência possam ser definidos, uma vez que diferentes VNFs possuem requisitos distintos. Para garantir a alta disponibilidade, uma plataforma NFV deve fornecer um sistema completo de gerenciamento de falhas.

## 3. NHAM: Arquitetura

O suporte para alta disponibilidade em ambientes NFV deve garantir a execução correta dos serviços virtualizados de rede, mesmo na presença de falhas. Até que a disponibilidade destes serviços seja comparável à de *middleboxes* implementados em hardware especializado, é improvável que eles sejam amplamente adotados.

A disponibilidade de um serviço aumenta à medida em que o tempo de recuperação diminui. Para reduzir o tempo de recuperação, é essencial detectar e reagir às falhas rapidamente, minimizando o tempo de inatividade. No entanto, o problema não pode ser resolvido apenas reservando recursos adicionais de forma a substituir ou criar uma nova VNF. A maioria das VNFs são *stateful* – o estado interno da função de rede muda de acordo com os pacotes processados e o fluxo da sua execução. Portanto, a recuperação de uma VNF deve também restaurar o estado da função em casos de falha.

Este trabalho propõe uma arquitetura de alta disponibilidade para SFCs *stateful* baseados em NFV, integrada como um módulo da arquitetura NFV-MANO. A arquitetura proposta estende o NHAM (NFV *High Availability Module*), proposto em [Venâncio et al. 2020] exclusivamente para a disponibilidade de VNFs individuais. O NHAM permite que as VNFs *herdem* as propriedades de alta disponibilidade, evitando alterações no seu código fonte. Além do gerenciamento de falhas, o NHAM realiza a recuperação de falhas de múltiplas VNFs por SFC, garantindo a confiabilidade de ponta-a-ponta das SFCs. O NHAM assume falhas por parada (falhas *crash*). Em seguida é

apresentada a arquitetura do NHAM para garantir a alta disponibilidade de VNFs individuais seguida da estratégia que combina o gerenciamento de *buffers* das SFCs com os *checkpoints* realizados pelo NHAM para garantir a alta disponibilidade de toda a SFC.

A Figura 3 ilustra a arquitetura interna do NHAM, que consiste de dois componentes principais: o *Fault Management System* (FMS) e o VNF *State Manager* (VSM), descritos a seguir. O FMS é o componente responsável pela gerência de falhas de todas as VNFs instanciadas na rede. Este componente fornece funcionalidades de detecção de falhas (*Failure Detector*), criação automatizada de réplicas (*Replica Manager*) e recuperação de falhas (*Failure Recovery*).

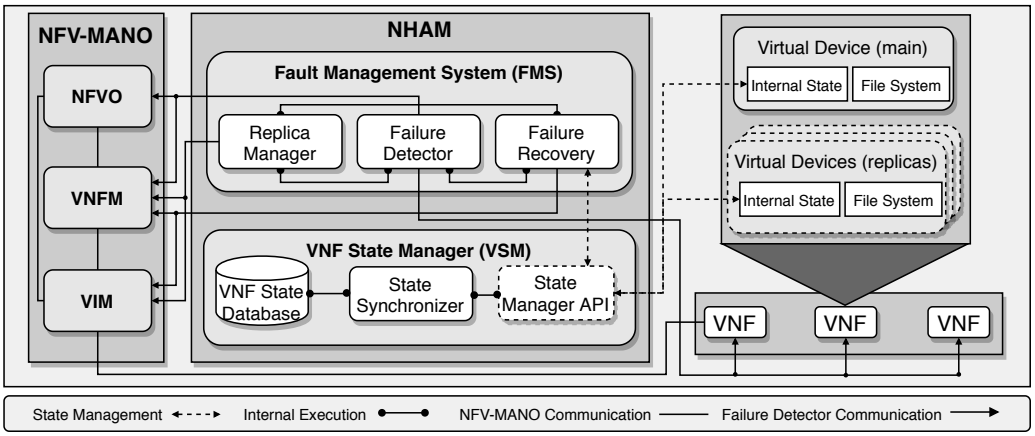



**Figura 3. NHAM: arquitetura interna.**

O VNF *State Manager* (VSM) é o componente responsável pela recuperação de VNFs *stateful*. Dessa forma, informações como mapeamento de memória, conexões TCP e cache, podem ser recuperadas após a ocorrência de uma falha. Para tanto, o VSM baseia-se em estratégias de *Checkpoint/Restore*, uma abordagem clássica para recuperar o estado de um sistema [Elnozahy et al. 2002]. Periodicamente, *checkpoints* contendo uma representação do estado da VNF são capturados e salvos em memória não-volátil. Quando uma falha é detectada, o NHAM pode restaurar o *checkpoint* mais recente.

O principal componente do VSM é o *State Synchronizer*, responsável pela realização dos *checkpoints*. Para cada VNF, o *State Synchronizer* cria um agente que coleta os *checkpoints* de cada VNF. Para realizar o *checkpoint*, o dispositivo virtual é pausado momentaneamente, para que as informações de estado interno possam ser obtidas de maneira consistente (*i.e.,* nenhuma modificação será feita na VNF durante a operação de *checkpoint*). Depois que o *checkpoint* é concluído, a execução da VNF é retomada.

A escolha de uma estratégia para garantir a alta disponibilidade de VNFs depende dos requisitos de resiliência de cada função de rede [Schöller et al. 2015]. O NHAM permite a escolha de diferentes mecanismos de resiliência, utilizados para a configuração de réplicas, acionadas em caso de falha da instância original. Os mecanismos de resiliência são baseados em dois métodos de replicação já definidos pelo ETSI [Nakamura et al. 2016]. O primeiro método é o *Active-Standby*, em que a réplica da VNF foi instanciada, porém está em modo *standby* (não operacional). O segundo método é o *Active-Active*, em que a réplica foi instanciada e já está ativa.

O componente *Replica Manager* é responsável por oferecer os diferentes meca-

nismos de resiliência para cada VNF. Quatro opções são definidas, de acordo com os respectivos requisitos de disponibilidade. Cada mecanismo de resiliência emprega um procedimento de recuperação diferente, descritos a seguir.

**Sem Redundância (0R).** Nenhum tipo de redundância é empregado: o *State Synchronizer* exporta os *checkpoints* da VNF para a VNF *State Database*. Após a ocorrência de uma falha, o NHAM instancia uma nova VNF, substituindo aquela que falhou. Em seguida, o *State Synchronizer* atualiza o estado interno da nova VNF, importando o *checkpoint* mais recente da VNF falha para a VNF recém criada.

**Réplica Primária *Active-Standby* (1R-AS).** Emprega o método *Active-Standby* com uma réplica que é instanciada em modo *standby*. Assim como no mecanismo 0R, o *State Synchronizer* exporta os *checkpoints* da VNF para a VNF *State Database*. Em casos de falha, uma réplica já está criada e o *State Synchronizer* precisa apenas importar o *checkpoint* mais recente para a réplica, de forma a atualizar o seu estado interno.

**Réplica Primária *Active-Active* (1R-AA)**. O mecanismo 1R-AA emprega o método *Active-Active*, no qual a VNF possui uma réplica ativa. Dessa forma, o *State Synchronizer* deve atualizar periodicamente a réplica com os *checkpoints* da VNF. Portanto, a réplica está constantemente recebendo atualizações sobre o estado da VNF primária. Após a ocorrência de uma falha, nenhum procedimento de recuperação específico é necessário, além de uma reconfiguração para indicar que a réplica agora é a VNF primária.

**Múltiplas Réplicas Active-Active (MR-AA).** O mecanismo de resiliência MR-AA é uma generalização do mecanismo 1R-AA, no qual a VNF pode ser vista como um membro de um grupo de $M$ réplicas que são continuamente sincronizadas. O *State Synchronizer* é responsável por manter a consistência dos estados das $M$ réplicas. Além disso, qualquer uma das réplicas neste grupo pode ser acessada para utilizar o serviço. Em caso de falha, este mecanismo não requer nenhuma reconfiguração, pois o usuário pode simplesmente acessar qualquer réplica do grupo.

Na próxima seção é apresentada a estratégia para garantir a alta disponibilidade de SFCs.

## 4. Serviços Virtualizados Altamente Disponíveis

A Seção 3 descreveu a arquitetura NHAM e a estratégia empregada para garantir a alta disponibilidade de VNFs individuais. Nesta seção é descrita a estratégia utilizada pelo NHAM para tornar as SFCs altamente disponíveis.

O NHAM adota uma estratégia de alta disponibilidade para serviços virtualizados que depende dos componentes da arquitetura interna de SFCs proposta pela IETF, descrita na Seção 2. É importante notar que nenhuma outra estratégia proposta para fornecer alta disponibilidade para SFCs é totalmente compatível com a arquitetura IETF. O NHAM aproveita as funcionalidades de componentes como o SFF e o SFP para permitir que SFCs *stateful* continuem operacionais mesmo após a falha de múltiplas VNFs.

### 4.1. Gerenciamento de *Buffers* e Premissas

O NHAM adota uma estratégia de alta disponibilidade que garante a recuperação de ponta-a-ponta de SFCs *stateful* após um número arbitrário de VNFs falharem. Na arquitetura original de uma SFC, cada VNF é precedida por um *buffer*, utilizado pelo SFF para armazenar pacotes, antes de serem entregues para a VNF. Na estratégia proposta neste

trabalho, ao invés de empregar um único *buffer*, dois *buffers* são empregados para cada VNF da SFC, conforme mostra a Figura 4. Esses *buffers* são denominados de *buffer_rx* e *buffer_tx*. O *buffer_rx* precede a VNF e recebe o tráfego a ser entregue para a VNF. Dessa forma, o *buffer_rx* armazena os pacotes que ainda não foram processados pela VNF. Já o *buffer_tx* sucede a VNF, recebendo o tráfego já processado na saída.

O fluxo de dados inicia quando o *buffer_rx* da primeira VNF da SFC recebe do SFF os pacotes que serão processados pela VNF. Em seguida, o SFF encaminha os pacotes do *buffer_rx* para a VNF. Depois de processar o tráfego de acordo com a função de rede, a VNF envia os pacotes para o *buffer_tx*. O SFF atua novamente movendo pacotes do *buffer_tx* de uma VNF para o *buffer_rx* da próxima VNF da SFC. Este processo é repetido para todas as demais VNFs da SFC. Após a última VNF processar o tráfego e inserir os pacotes no último *buffer_tx*, o SFF entrega adequadamente esse tráfego ao destino.

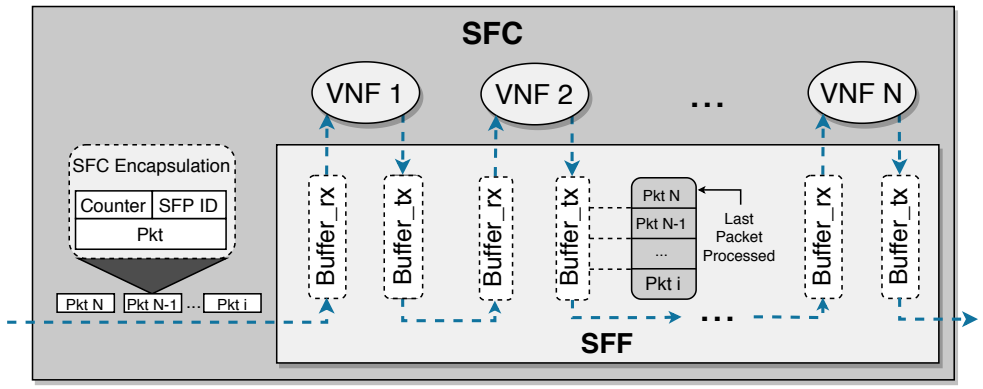



**Figura 4. A arquitetura para SFCs altamente disponíveis.**

O NHAM assume que tanto os *buffers* quanto o SFF não falham. Portanto, é recomendável executar as VNFs e os *buffers* em hardwares fisicamente separados. O NHAM também assume que os demais componentes da arquitetura NFV-MANO não falham. Além disso, a estratégia de recuperação do NHAM assume que todas as VNFs da SFC processam tráfego considerando a ordem FIFO (*First In First Out*). Assim, sempre que dois pacotes são enviados em uma determinada ordem para a VNF, se eles não forem descartados pela VNF, eles são enviados preservando essa ordem.

A estratégia *Hold/Release* é proposta a seguir para garantir a recuperação consistente de uma SFC após uma ou mais falhas. A estratégia *Hold/Release* é baseada em uma combinação de *Checkpoint/Restore* de VNFs juntamente com o gerenciamento de *buffers* de uma SFC.

### 4.2. A Estratégia *Hold/Release*

Para tornar as SFCs tolerantes a falhas e garantir a sua alta disponibilidade, a recuperação deve considerar serviços *stateful*. Os serviços *stateful* consistem de SFCs com uma ou mais VNFs *stateful*. Os estados destas VNFs devem ser tratados com cuidado, de forma a evitar inconsistências após falhas e recuperações.

A recuperação de uma SFC *stateful* consiste de duas partes principais: *(i)* a recuperação individual de cada VNF que falhou, incluindo a restauração do estado interno no caso de VNFs *stateful* e; *(ii)* a retransmissão do tráfego perdido durante a inatividade

de uma VNF em decorrência de uma falha. As técnicas propostas para a recuperação individual de VNFs *(i)* foram descritas na Seção 3. A estratégia *Hold/Release* descrita a seguir é proposta para resolver *(ii)*.

Antes que o SFF insira cada pacote no primeiro *buffer_rx* da primeira VNF da SFC, cada pacote é encapsulado para que possa ser roteado ao longo do SFP [Halpern and Pignataro 2015]. Além de transportar informações explícitas para identificar o SFP, o NHAM inclui no encapsulamento do SFF um *timestamp*, implementado como um contador e que funciona como um identificador de pacote sequencial.

Através do seu módulo de monitoramento, o NHAM verifica continuamente o estado das VNFs, identificando possíveis falhas. Caso alguma falha seja detectada, o NHAM envia imediatamente uma mensagem ao SFF, requisitando a alteração do estado da SFC para *recovering* (em recuperação). Quando uma SFC está neste estado, o tráfego não progride através do SFP até que as falhas sejam completamente recuperadas. Consequentemente, uma VNF em recuperação não recebe pacotes do *buffer_rx* nem envia pacotes para o *buffer_tx*.

Como mencionado acima, o SFF armazena no *buffer_rx* os pacotes a serem processados pela VNF. Na estratégia *Hold/Release*, após os pacotes do *buffer_rx* serem entregues à VNF, eles não são imediatamente removidos do *buffer*, *i.e.,* eles são temporariamente retidos (*Hold*). A VNF então recebe e processa os pacotes e armazena o tráfego resultante no *buffer_tx*. Periodicamente, *checkpoints* das VNFs são realizados pelo NHAM. A ideia é manter o tráfego no *buffer_rx* até que um *checkpoint* da VNF seja realizado após ter processado uma sequência de pacotes. Se um *checkpoint* for executado após um determinado pacote ter sido processado pela VNF, dizemos que o *checkpoint* inclui esse pacote.

Durante a realização de um *checkpoint*, a VNF é momentaneamente pausada (*i.e.,* nenhum tráfego é recebido ou enviado) e o SFF aguarda até que este procedimento seja finalizado. A partir deste momento, é possível concluir que o último pacote no *buffer_tx* foi processado pela VNF e incluído no *checkpoint*. Antes de liberar a VNF para retomar a sua execução, o SFF remove todos os pacotes até o último pacote do *buffer_rx* que foi incluído no *checkpoint* (*Release*). Caso a VNF falhe antes que o *checkpoint* seja executado, a VNF é revertida para o *checkpoint* anterior e os procedimentos de recuperação descritos na Seção 3 são realizados. Além disso, todos os pacotes que a VNF recebeu a partir do último *checkpoint* (que ainda estão no *buffer_rx*) devem ser enviados novamente e processados pela VNF. Este último passo é essencial para garantir a consistência do serviço, visto que cada VNF recebe e processa tráfego entre *checkpoints*.

Considere como exemplo que todos os pacotes até o pacote $i$ foram processados por uma VNF quando um *checkpoint* é iniciado. Considere que o pacote $i + 1$ também foi enviado do *buffer_rx* para a VNF, mas não foi incluído no *checkpoint*. Quando o *checkpoint* é concluído, o SFF confirma que o último pacote que já estava no *buffer_tx* é o pacote $i$ que, portanto, foi incluído no *checkpoint*. Neste momento, todos os pacotes até o $i$ podem ser removidos do *buffer_rx*. Observe que o pacote $i + 1$ ainda não pode ser removido, pois ele não foi incluído no *checkpoint* e pode ser necessário reenviá-lo e reprocessá-lo pela VNF novamente em casos de falha.

A comunicação entre VNFs consecutivas trás desafios adicionais para a gerência de pacotes. Considere por exemplo uma situação em que os pacotes $i$, $i + 1$ e $i + 2$ foram enviados do *buffer_rx* para a VNF $1$. No entanto, a VNF $1$ processou apenas até o pacote

$i$, quando um *checkpoint* foi concluído. O SFF pode então remover todos os pacotes até o $i$ do *buffer_rx*. Considere, entretanto, que após a conclusão do *checkpoint*, que a VNF 1 continua o seu processamento e envia os pacotes $i + 1$ e $i + 2$ para o *buffer_tx*, e que em seguida já são encaminhados para a VNF 2. O SFF mantém o registro do último pacote de cada VNF no *buffer_tx*, neste caso o pacote $i + 2$. Considere agora que a VNF 1 falha neste momento. Como os pacotes $i + 1$ e $i + 2$ não foram incluídos no último *checkpoint*, eles devem ser reprocessados após a recuperação da VNF 1. No entanto, eles já foram encaminhados para a VNF 2. Neste caso, o SFF evita pacotes duplicados encaminhando para a VNF 2 apenas pacotes com identificador maior que o último pacote registrado, *i.e.,* pacotes $i + 3$ em diante.

A recuperação da VNF ocorre de acordo com o seu respectivo mecanismo de resiliência, conforme descrito na Seção 3. Após a VNF retomar o seu funcionamento com o seu respectivo estado interno restaurado, o próximo passo é a retransmissão de todo o tráfego em *buffer_rx*, incluindo os pacotes que a VNF recebeu desde que o último *checkpoint* foi salvo.

A estratégia *Hold/Release* garante que a SFC se recupere independentemente do número e de quais VNFs falharam ao longo do SFP. Múltiplas VNFs podem inclusive falhar ao mesmo tempo, por exemplo, devido a uma queda de energia. Observe que o *buffer_rx* de uma determinada VNF somente é esvaziado após um *checkpoint* ser salvo e os pacotes correspondentes estarem no *buffer_tx*. Dessa forma, nem o tráfego processado entre *checkpoints* de cada VNF, nem o tráfego entre VNFs é perdido em decorrência de uma falha, garantindo a consistência da SFC de ponta-a-ponta.

## 5. Implementação e Avaliação Experimental

Um protótipo da arquitetura NHAM foi implementado dentro de uma plataforma NFV compatível com o modelo de referência NFV-MANO. O protótipo foi implementado em Python e é baseado em contêineres Docker [Merkel 2014]. Uma API REST foi implementada para facilitar o gerenciamento do estado interno das VNFs. A ferramenta CRIU (*Checkpoint/Restore in Userspace*) [CRIU 2019] foi empregada para realizar o *checkpoints* das VNFs. Os *checkpoints* incluem somente as informações mínimas necessárias para restaurar a VNF, incluindo a própria função de rede e alguns recursos relacionados, como mapeamentos de memória e árvore de processo (*process tree*). Os *checkpoints* são executados a cada 500ms. O tamanho dos *buffers* na SFC é de 64 KB.

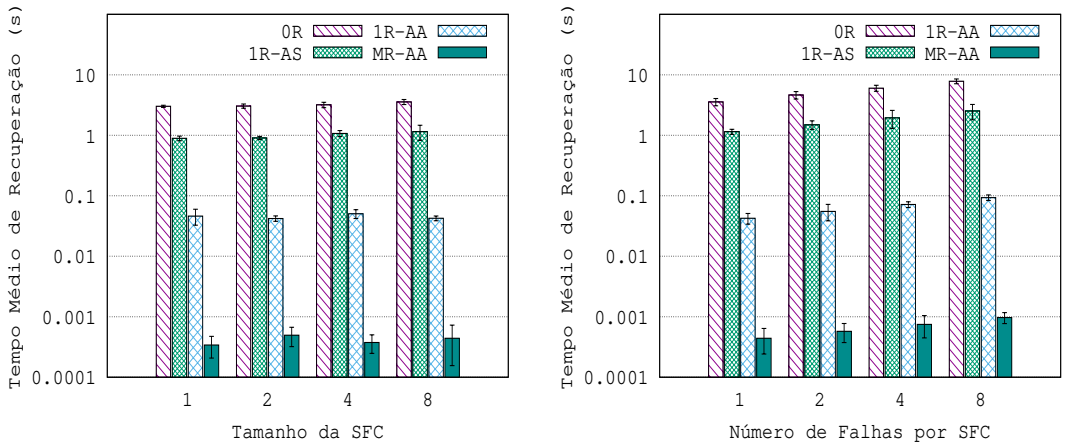
O protótipo do NHAM foi executado em um processador Intel Core i7 a 2.50 GHz com 8 núcleos, 16 GB de RAM, uma placa de rede Ethernet de 1 Gbps e Linux Ubuntu 20.04. Cada VNF consiste em um Ubuntu *server* com 256 MB de RAM e 1 CPU. O mecanismo MR-AA usa 3 réplicas por padrão. Note que o NHAM não requer nenhum *patch* no *kernel* para funcionar.

Um conjunto de experimentos foi executado para avaliar o desempenho do NHAM e a disponibilidade atingida pelas VNFs com o suporte da solução proposta. O primeiro conjunto de experimentos avalia o tempo de recuperação conforme o tamanho da SFC aumenta. O segundo conjunto de experimentos avalia o impacto do NHAM na vazão, enquanto que o último experimento mede a disponibilidade dos serviços baseados em NFV suportados pelo NHAM. Cada experimento foi repetido 10 vezes e as médias dos resultados são apresentadas com um intervalo de confiança de 95%.

### 5.1. Tempo de Recuperação

A redução no tempo de inatividade é particularmente importante para aumentar a disponibilidade dos serviços virtualizados. O primeiro conjunto de experimentos mede o tempo total de recuperação, desde a detecção da falha até a recuperação total da SFC. As falhas foram injetadas através de *scripts* que removem todas as conexões das VNFs, impedindo que as mensagens de monitoramento sejam recebidas, gerando as suspeitas de falha.

O primeiro experimento, ilustrado na Figura 5(a), mede o tempo de recuperação após a falha de uma única VNF na SFC. Este experimento compara os quatro diferentes mecanismos de resiliência, enquanto que o número de VNFs na SFC aumenta de 1 até 8.




(a) Tempo de recuperação após a falha de uma única VNF.

(b) Tempo de recuperação após a falha de múltiplas VNFs.

**Figura 5. Tempo de recuperação de uma SFC.**

Como previsto, o mecanismo 0R apresenta o maior tempo de recuperação, atingindo até 3.6s de inatividade para uma SFC com 8 VNFs. Este é o tempo que leva para instanciar uma nova VNF (2.4s) e para o VSM restaurar o *checkpoint* mais recente. O mecanismo 1R-AS apresenta resultados melhores, uma vez que a réplica já está instanciada em modo *standby*. Com o método *Active-Standby*, o NHAM somente importa um *checkpoint* para a réplica, reduzindo o tempo total de recuperação para 1.14s para uma SFC com 8 VNFs.

Mecanismos baseados no método *Active-Active* apresentam resultados ainda melhores. O mecanismo 1R-AA atingiu uma média de até 0.05s no tempo total de recuperação, uma vez que as réplicas ativas mantém o seu estado atualizado. Já o mecanismo MR-AA obteve os melhores resultados, devido ao seu grupo de réplicas sincronizadas. O seu tempo de recuperação é de apenas 0.0002s, o que corresponde à atualização do grupo de réplicas. É importante ressaltar que, para todas as estratégias, o tempo de recuperação permanece inalterado conforme o tamanho da SFC aumenta.

O próximo experimento avalia o impacto no tempo de recuperação quando várias falhas ocorrem ao mesmo tempo. O experimento exibido na Figura 5(b) utiliza SFCs com 8 VNFs e o número de falhas por SFC varia de 1 (falha de uma única VNF) a 8 (falha de toda a SFC).

O NHAM detecta e recupera múltiplas VNFs com falha em paralelo. Neste sen-

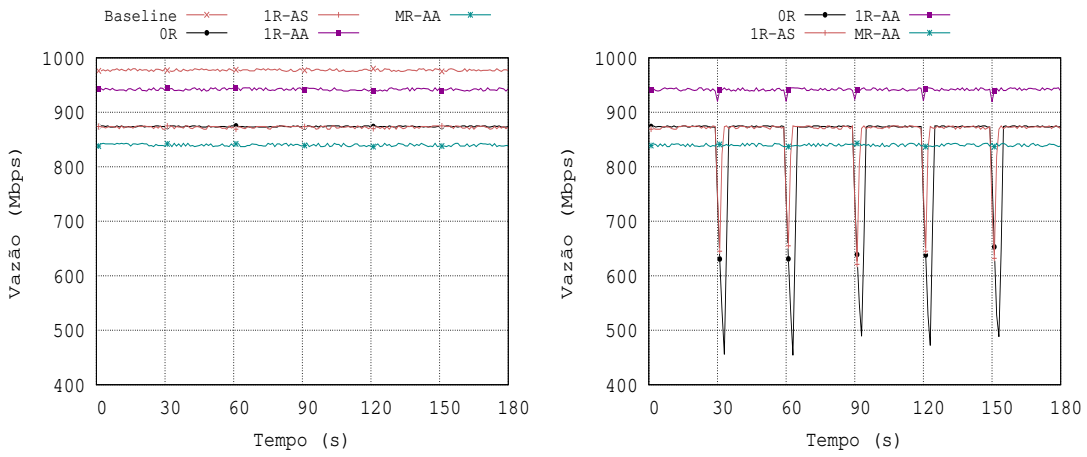
tido, foi medido o impacto no tempo de recuperação conforme a quantidade de falhas simultâneas aumenta. Para o mecanismo de resiliência 0R, o tempo de recuperação aumenta de 3.56s (uma falha) para 7.8s (8 falhas). Para o mecanismo 1R-AS, a diferença entre recuperar uma única falha e recuperar toda a SFC é menor: 1.14s para 2.52s.

Para os mecanismos 1R-AA e MR-AA a diferença é ainda menor, já que o tempo de recuperação de uma única falha é significativamente menor do que nas outras estratégias. Para a estratégia 1R-AA, o tempo de recuperação varia de 0.004s até 0.009s, enquanto que para o MR-AA o tempo de recuperação varia de 0.0004s até 0.0009s. A partir destes resultados, é possível concluir que todos os mecanismos de recuperação são escaláveis no que diz respeito ao número de falhas de VNFs por SFC.

## 5.2. Vazão da SFC

O NHAM emprega a estratégia *Hold/Release* para garantir a alta disponibilidade de serviços virtualizados. Os experimentos descritos nesta seção medem o impacto dessa estratégia com relação à vazão obtida pela SFC. O primeiro experimento, mostrado na Figura 6, avalia o impacto na vazão em dois cenários diferentes: o primeiro cenário é sem falhas enquanto que no segundo falhas ocorrem a cada 30 segundos. As SFCs nestes experimentos consistem de 4 VNFs.

No cenário sem falhas (Figura 6(a)), cada mecanismo de resiliência é comparado ao *baseline* – uma SFC sem o suporte do NHAM. Os mecanismos 0R e 1R-AS apresentam vazão semelhante, já que ambos os mecanismos realizam *checkpoints* da mesma maneira. Esses mecanismos diminuem a taxa de transferência em aproximadamente 11.5%. Isso se deve ao fato de que os *checkpoints* são obtidos, compactados e salvos em memória não-volátil em um banco de dados.




(a) Vazão da SFC durante um cenário sem falhas. (b) Vazão da SFC durante um cenário com falhas.

**Figura 6. Impacto dos diferentes mecanismos de resiliência na vazão da SFC.**

Para o mecanismo 1R-AA, a vazão diminui em aproximadamente 4.7%. O mecanismo 1R-AA realiza as suas operações diretamente em memória, melhorando significativamente a vazão da SFC. Por outro lado, o mecanismo MR-AA apresenta a maior degradação na vazão. Apesar do tempo de recuperação muito baixo, a vazão diminui em 14.1%. Assim como o 1R-AA, o mecanismo MR-AA também executa em memória, en-

tretanto, todo o processamento feito para garantir que o grupo de réplicas de cada VNF permaneça consistente tem impacto significativo na vazão.

No cenário com falhas na SFC, mostrado na Figura 6(b), as falhas são injetadas a cada 30 segundos. É possível observar que a redução na vazão é maior para os mecanismos 0R e 1R-AS do que para os métodos baseados em *Active-Active*, consequência dos seus maiores tempos de recuperação. É importante observar que, mesmo neste cenário, a vazão permanece constante para os mecanismos 1R-AA e MR-AA.

### 5.3. Avaliando a Disponibilidade de SFCs

Neste experimento, a disponibilidade de serviços baseados em NFV com o suporte do NHAM foi calculada através de um MTBF (*Mean Time Between Failures*) variável. O MTBF indica o tempo médio entre falhas. A Tabela 1 apresenta os resultados para cada mecanismo de resiliência. Cada experimento durou 3 horas e o MTBF indica a frequência com que as falhas foram injetadas. As SFCs nestes experimentos consistem de 8 VNFs.

**Tabela 1. Disponibilidade das SFCs conforme o MTBF varia.**

| MTBF | Disponibilidade (%) | | | |
|---|---|---|---|---|
| | **0R** | **1R-AS** | **1R-AA** | **MR-AA** |
| 60 (1 min) | 98.057 | 98.240 | 99.916 | 99.999 |
| 300 (5 min) | 99.605 | 99.643 | 99.983 | 99.999 |
| 600 (10 min) | 99.802 | 99.821 | 99.991 | 99.999 |
| 900 (15 min) | 99.868 | 99.880 | 99.994 | 99.999 |
| 1200 (20 min) | 99.901 | 99.910 | 99.995 | 99.999 |
| 1500 (25 min) | 99.920 | 99.928 | 99.996 | 99.999 |
| 1800 (30 min) | 99.934 | 99.940 | 99.997 | 99.999 |

O mecanismo 0R apresenta os níveis de disponibilidade mais baixos, embora seu custo possa justificar sua utilização para serviços que toleram tempos de recuperação mais longos. Mesmo para testes com um maior MTBF, o 0R atingiu apenas 99.3% de disponibilidade. O mecanismo 1R-AS também não atinge os níveis de disponibilidade necessários para garantir a alta disponibilidade de SFCs, apesar de alcançar resultados melhores. O mecanismo 1R-AA apresenta um desempenho melhor, mesmo no cenário mais sujeito a falhas: com um MTBF de 60, as SFCs obtiveram 99.9% (*three nines*) de disponibilidade e 99.99% (*four nines*) com um MTBF de 600. Naturalmente, o mecanismo MR-AA apresenta os melhores resultados, atingindo uma disponibilidade de 99.999% (*five nines*) em todos os casos, o nível exigido de sistemas comerciais de telecomunicações.

## 6. Trabalhos Relacionados

REINFORCE [Kulkarni et al. 2018] é um *framework* para prover resiliência para VNFs e SFCs. O estado das VNFs é continuamente replicado para réplicas locais ou remotas. Dentre as diversas alternativas do NHAM, o REINFORCE emprega um único método *Active-Standby*. Além disso, o REINFORCE exige que os próprios desenvolvedores das VNFs indiquem quais operações causam mudanças em seu estado interno, etapa desnecessária no NHAM. Por fim, o REINFORCE não é compatível com o NFV-MANO.

Remus [Cully et al. 2008] é um sistema de alta disponibilidade para VMs. O Remus salva periodicamente os *checkpoints* de uma VM, atualizando um backup. Após a ocorrência de falha, o *backup* pode assumir. O Remus sincroniza *checkpoints* por meio

de *buffers*, mantendo os pacotes em um *buffer* até que a sincronização de um novo estado seja concluída. A combinação do gerenciamento de *buffers* e *checkpoints* é semelhante ao NHAM, embora os contextos em que são empregados sejam diferentes.

Em outra proposta baseada em *buffers* [Rajagopalan et al. 2013], os autores propõem o *Pico Replication* (PR), um *framework* de alta disponibilidade para *middleboxes*. Ao invés de capturar o estado interno dos *middleboxes*, o PR executa *checkpoints* diretamente em fluxos de dados específicos, enquanto o *middlebox* continua a processar outros fluxos. O PR requer extensas modificações para garantir a alta disponibilidade de *middleboxes*, incluindo mudanças de *kernel* e no controlador SDN.

Outra estratégia proposta em [Kablan et al. 2017] e [Khalid and Akella 2019] consiste em desacoplar o estado interno das VNFs do seu processamento, salvando o estado interno em um banco de dados distribuído. Dessa forma, a tolerância a falhas é alcançada para funções de rede *stateful*, uma vez que em casos de falha uma nova instância é criada e o estado é recuperado pelo banco de dados.

Uma abordagem baseada em *rollback-recovery* é proposta em [Sherry et al. 2015]. O sistema FTMB salva o estado de *middleboxes* registrando as informações necessárias para reproduzir as entradas do sistema em caso de falha. O sistema contempla dependências entre pacotes. A estratégia proposta requer modificações no código fonte de cada VNF.

Em [Gember-Jacobson et al. 2014] é proposta a arquitetura de um plano de controle que redireciona pacotes de VNFs falhas para VNFs corretas, ao passo em que garante a sincronização do estado interno das VNFs. O plano de controle, denominado de OpenNF, gerencia os estados e minimiza a perda de informações transferindo os pacotes através de um controlador SDN. Além disso, o OpenNF propõe uma API para gerenciar os estados das VNF, semelhante à API proposta para o NHAM. No entanto, a API OpenNF requer alterações no código fonte da VNF.

É possível afirmar, em geral, que o NHAM tem três vantagens sobre as soluções existentes. A primeira é o suporte para diferentes mecanismos de resiliência. A segunda vantagem: o NHAM não requer qualquer modificação no código fonte das VNFs para que elas se tornem altamente disponíveis. Finalmente, nenhuma das soluções é totalmente compatível com a arquitetura de referência NFV-MANO, dificultando a integração com outras plataformas NFV.

## 7. Conclusão

A tecnologia NFV tem múltiplas vantagens e a alta disponibilidade é essencial para sua ampla adoção. Este trabalho propõe o NHAM: uma arquitetura de alta disponibilidade para NFV, definida como um módulo da arquitetura de referência NFV-MANO. O NHAM garante a alta disponibilidade tanto de VNFs quanto de SFCs *stateful*, sem a necessidade de modificações no código-fonte das funções de rede. Quatro mecanismos de resiliência diferentes são definidos, os quais podem ser escolhidos de acordo com os recursos e requisitos dos diferentes tipos de VNFs. O NHAM emprega um gerenciador de estados que combina técnicas de *Checkpoint/Restore* com o gerenciamento de *buffers* de uma SFC, permitindo a recuperação de serviços virtualizados *stateful*, mesmo após a falha de uma ou mais VNFs. Desta forma, o NHAM garante a recuperação completa e correta do serviço ponto-a-ponto. Um protótipo foi implementado e resultados experimentais foram

executados para avaliar o desempenho e a disponibilidade de VNFs e SFCs com o suporte do NHAM. Os resultados mostram que o NHAM é uma solução eficaz para melhorar a robustez dos serviços virtualizados que podem atingir níveis de disponibilidade similares aos de sistemas comerciais de telecomunicações. Trabalhos futuros incluem a ampliação do NHAM com estratégias para a prevenção e previsão de falhas no contexto de NFV.

## Referências


Bondan, L. et al. (2019). Fende: Marketplace-based distribution, execution, and life cycle management of vnfs. *IEEE Communications Magazine*, 57(1):13–19.

Cotroneo, D. et al. (2014). Network function virtualization: Challenges and directions for reliability assurance. In *IEEE International Symp. on Software Reliability Engineering Workshops*, pages 37–42.

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

Cully, B. et al. (2008). Remus: High availability via asynchronous virtual machine replication. In *Proceedings of the 5th USENIX Symp. on NSDI*, pages 161–174.

Elnozahy, E. et al. (2002). A survey of rollback-recovery protocols in message-passing systems. *ACM Computing Surveys (CSUR)*, 34(3):375–408.

Gember-Jacobson, A. et al. (2014). Opennf: Enabling innovation in network function control. In *ACM SIGCOMM Computer Communication Review*, pages 163–174. ACM.

Halpern, J. and Pignataro, C. (2015). Service Function Chaining (SFC) Architecture. RFC 7665, IETF.

Han, B. et al. (2017). On the resiliency of virtual network functions. *IEEE Communications Magazine*, 55(7):152–157.

Kablan, M. et al. (2017). Stateless network functions: Breaking the tight coupling of state and processing. In *14th USENIX Symp. on NSDI*, pages 97–112.

Khalid, J. and Akella, A. (2019). Correctness and performance for stateful chained network functions. In *16th USENIX Symposium on Networked Systems Design and Implementation (NSDI 19)*, pages 501–516.

Kulkarni, S. et al. (2018). Reinforce: Achieving efficient failure resiliency for network function virtualization based services. In *Proceedings of the 14th International CoNEXT*, pages 41–53. ACM.

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

Mijumbi, R. et al. (2016). Network function virtualization: State-of-the-art and research challenges. *IEEE Communications Surveys & Tutorials*, 18(1):236–262.

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

Quittek, J. et al. (2014). Network Functions Virtualisation (NFV); Management and Orchestration. GS NFV-MAN V1.1.1. Technical report, ETSI.

Rajagopalan, S. et al. (2013). Pico replication: A high availability framework for middleboxes. In *Proceedings of the 4th annual Symposium on Cloud Computing*, pages 1–15.

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

Sherry, J. et al. (2015). Rollback-recovery for middleboxes. In *ACM SIGCOMM Computer Communication Review*, pages 227–240. ACM.

Venâncio, G. et al. (2020). Uma arquitetura de alta disponibilidade para funções virtualizadas de rede. In *Anais do XXXVIII SBRC*, pages 407–420. SBC.