Autocura para Redes Definidas por Software

Natal Vieira de Souza Neto

Universidade Federal de Uberlândia
Faculdade de Computação
Programa de Pós-Graduação em Ciência da Computação

Uberlândia
2021

Natal Vieira de Souza Neto

Autocura para Redes Definidas por Software

Tese de doutorado apresentada ao Programa de Pós-graduação da Faculdade de Computação da Universidade Federal de Uberlândia como parte dos requisitos para a obtenção do título de Doutor em Ciência da Computação.

Área de concentração: Ciência da Computação

Orientador: Pedro Frosi Rosa
Coorientador: Luiz A. DaSilva
Coorientador: Flávio de Oliveira Silva

Uberlândia
2021

Ficha Catalográfica Online do Sistema de Bibliotecas da UFU
com dados informados pelo(a) próprio(a) autor(a).

S729 Souza Neto, Natal Vieira de, 1990-
2021 Autocura para Redes Definidas por Software [recurso eletrônico] / Natal Vieira de Souza Neto. - 2021.

Orientador: Pedro Frosi Rosa.
Coorientador: Luiz Antonio Pereira da Silva.
Coorientador: Flávio de Oliveira Silva.
Tese (Doutorado) - Universidade Federal de Uberlândia, Pós-graduação em Ciência da Computação.
Modo de acesso: Internet.
Disponível em: http://doi.org/10.14393/ufu.te.2021.246
Inclui bibliografia.
Inclui ilustrações.

1. Computação. I. Rosa, Pedro Frosi,1959-, (Orient.). II. Silva, Luiz Antonio Pereira da,1964-, (Coorient.). III. Silva, Flávio de Oliveira,1970-, (Coorient.). IV. Universidade Federal de Uberlândia. Pós-graduação em Ciência da Computação. V. Título.

CDU: 681.3

Bibliotecários responsáveis pela estrutura de acordo com o AACR2:

Gizele Cristine Nunes do Couto - CRB6/2091

UNIVERSIDADE FEDERAL DE UBERLÂNDIA
Coordenação do Programa de Pós-Graduação em Ciência da Computação
Av. João Naves de Ávila, nº 2121, Bloco 1A, Sala 243 - Bairro Santa Mônica, Uberlândia-MG, CEP 38400-902
Telefone: (34) 3239-4470 - www.ppgco.facom.ufu.br - cpgfacom@ufu.br

ATA DE DEFESA - PÓS-GRADUAÇÃO

Programa de Pós-Graduação em:	Ciência da Computação				
Defesa de:	Tese de Doutorado, 06/2021, PPGCO				
Data:	28 de abril de 2021	Hora de início:	10:00	Hora de encerramento:	13:50
Matrícula do Discente:	11713CCP005				
Nome do Discente:	Natal Vieira de Souza Neto				
Título do Trabalho:	Autocura para Redes Definidas por Software				
Área de concentração:	Ciência da Computação				
Linha de pesquisa:	Sistemas de Computação				
Projeto de Pesquisa de vinculação:	-				

Reuniu-se, por videoconferência, a Banca Examinadora, designada pelo Colegiado do Programa de Pós-graduação em Ciência da Computação, assim composta: Professores Doutores: Rafael Pasquini - FACOM/UFU, Rodrigo Sanches Miani - FACOM/UFU, Rui Luis Andrade Aguiar - Universidade de Aveiro, Eduardo Coelho Cerqueira - UFPA, Luiz A. da Silva - Virginia Tech (coorientador), Flávio de Oliveira Silva - FACOM/UFU (coorientador) e Pedro Frosi Rosa - FACOM/UFU orientador do candidato.

Os examinadores participaram desde as seguintes localidades: Rui Luis Andrade Aguiar - Aveiro/Portugal; Eduardo Coelho Cerqueira -Belém/PA; Luiz A. da Silva - Blacksburg, Virgínia, Estados Unidos; Rafael Pasquini, Rodrigo Sanches Miani, Flávio de Oliveira Silva e Pedro Frosi Rosa - Uberlândia/MG. O discente participou da cidade de Uberlândia/MG.

Iniciando os trabalhos o presidente da mesa, Prof. Dr. Pedro Frosi Rosa, apresentou a Comissão Examinadora e o candidato, agradeceu a presença do público, e concedeu ao Discente a palavra para a exposição do seu trabalho. A duração da apresentação do Discente e o tempo de arguição e resposta foram conforme as normas do Programa.

A seguir o senhor presidente concedeu a palavra, pela ordem sucessivamente, aos examinadores, que passaram a arguir o candidato. Ultimada a arguição, que se desenvolveu dentro dos termos regimentais, a Banca, em sessão secreta, atribuiu o resultado final, considerando o candidato:

Aprovado.

Esta defesa faz parte dos requisitos necessários à obtenção do título de Doutor.

Ressalta-se que os examinadores Rui Luis Andrade Aguiar e Luiz A. da Silva por serem estrangeiros, residentes em outro país e não possuírem CPF registrado no Brasil não podem assinar a ata de defesa.

O competente diploma será expedido após cumprimento dos demais requisitos, conforme as normas do Programa, a legislação pertinente e a regulamentação interna da UFU.

Nada mais havendo a tratar foram encerrados os trabalhos. Foi lavrada a presente ata que após lida e achada conforme foi assinada pela Banca Examinadora.

Documento assinado eletronicamente por **Rafael Pasquini**, **Professor(a) do Magistério Superior**, em 28/04/2021, às 15:07, conforme horário oficial de Brasília, com fundamento no art. 6º, § 1º, do Decreto nº 8.539, de 8 de outubro de 2015.

Documento assinado eletronicamente por **Rodrigo Sanches Miani**, **Professor(a) do Magistério Superior**, em 28/04/2021, às 16:02, conforme horário oficial de Brasília, com fundamento no art. 6º, § 1º, do Decreto nº 8.539, de 8 de outubro de 2015.

Documento assinado eletronicamente por **Pedro Frosi Rosa**, **Professor(a) do Magistério Superior**, em 28/04/2021, às 23:00, conforme horário oficial de Brasília, com fundamento no art. 6º, § 1º, do Decreto nº 8.539, de 8 de outubro de 2015.

Documento assinado eletronicamente por **Eduardo Coelho Cerqueira**, **Usuário Externo**, em 29/04/2021, às 16:00, conforme horário oficial de Brasília, com fundamento no art. 6º, § 1º, do Decreto nº 8.539, de 8 de outubro de 2015.

Documento assinado eletronicamente por **Flávio de Oliveira Silva**, **Professor(a) do Magistério Superior**, em 11/05/2021, às 14:24, conforme horário oficial de Brasília, com fundamento no art. 6º, § 1º, do Decreto nº 8.539, de 8 de outubro de 2015.

A autenticidade deste documento pode ser conferida no site https://www.sei.ufu.br/sei/controlador_externo.php?acao=documento_conferir&id_orgao_acesso_externo=0, informando o código verificador **2728851** e o código CRC **07E0BE2A**.

Este trabalho é dedicado aos cientistas brasileiros que, apesar de todos os obstáculos, trabalham incansavelmente para o desenvolvimento tecnológico do nosso país.

Agradecimentos

Não tenho certeza se é uma questão de sorte ou não mas posso afirmar que, ao longo da minha trajetória, acabei me deparando com pessoas extraordinárias... Algumas delas não posso deixar de mencionar e agradecer.

Primeiramente, agradeço à minha família, em especial minha mãe Luzia e minha avó Ilídia, pelo apoio incondicional em todos os caminhos que resolvi percorrer no âmbito pessoal, profissional e acadêmico. Aprendi que família e conhecimento são as maiores riquezas.

Agradeço à minha namorada Rayany pelo amor, companheirismo e incentivo em todas as minhas inciativas. Obrigado pela compreensão nos momentos de ausência em consequência do tempo dedicado a esta pesquisa.

Agradeço aos amigos Maurício, Daniel e Prof. Flávio, do grupo "SONAr Fellas", pelas importantes contribuições neste trabalho, e pelos momentos de diversão regados com bons chopes artesanais.

Agradeço à Algar Telecom pelo apoio para a participação em congressos e viagens acadêmicas, e aos colegas de trabalho pela motivação que sempre proporcionaram, em especial ao Umberto, um grande incentivador.

Agradeço à CAPES (Coordenação de Aperfeiçoamento de Pessoal de Nível Superior) pela bolsa de doutorado sanduíche concedida durante este trabalho.

Agradeço a todos os colegas do Connect Centre, em Dublin, Irlanda, por terem me recebido tão bem como pesquisador visitante durante o doutorado sanduíche. Em especial ao André, pela recepção em Dublin; ao João, colega de pesquisa e contribuidor imensurável deste trabalho; e, finalmente, ao Prof. DaSilva, pela oportunidade e pela supervisão desta pesquisa.

Por fim, gostaria de deixar um agradecimento especial ao meu amigo Prof. Pedro, orientador deste trabalho e que, desde o mestrado, vem me mostrando que um bom café entre amigos pode resultar em grandes ideias que não surgiriam nas reuniões em salas fechadas.

"O desejo profundo da humanidade pelo conhecimento é justificativa suficiente para nossa busca contínua."

(Stephen Hawking)

"A computação autônoma é uma jornada."

(A. G. Ganek e T. A. Corbi)

Resumo

As Redes Definidas por Software – *Software-defined Networking* (SDN) – e a Virtualização de Funções de Rede – *Network Functions Virtualisation* (NFV) – são abordagens que têm sido melhoradas nos últimos anos e possuem vantagens claras. Em SDN, novos elementos são introduzidos, como o controlador e aplicações, e requerem gerenciamento. Diversos trabalhos apontam alguns problemas relacionados a falhas em SDN, como inconsistências de regras de encaminhamento e erros em software e hardware. Em NFV, o esforço para virtualizar funções de rede acaba trazendo elementos complexos de serem gerenciados, como orquestradores de recursos de rede. Em ambas as abordagens, não houve estruturação arquitetural para tratar gerenciamento. Basicamente, os mecanismos para gerenciamento estão sendo incluídos posteriormente, conforme já era comum em outras propostas de redes de computadores e redes móveis. Considerando-se a não viabilidade de gerenciamento manual dessas estruturas complexas, uma nova arquitetura está sendo proposta: *Self-organising Networks Architecture* (SONAr). Nessa arquitetura, fundamentos de computação autônoma, amadurecidos na engenharia de software, e aspectos de Redes Auto-organizáveis – *Self-organising Networks* (SON) – amadurecidos em telecomunicações, são utilizados para autogerenciamento – *self-management*. A presente tese introduz a arquitetura SONAr e aprofunda no fundamento de autocura – *self-healing*. Realizou-se uma análise minuciosa de autocura para os planos de Dados, Controle e Gerenciamento e criou-se um *framework*, denominado *Network Operations Self-healing* (NOSH), com capacidades de autocura em SDN e NFV. Duas implementações distintas foram realizadas e os experimentos demonstram que o NOSH é capaz de autocurar, ou seja, curar o ambiente degradado, bem como curar a si mesmo. Além disso, os experimentos provam a eficácia da utilização de autocura considerando *Network Slice*s (NSs) com rigorosos requisitos de *Quality of Service* (QoS), como em *Ultra-Reliable and Low Latency Communications* (URLLC).

Palavras-chave: Autocura. Autogerenciamento. SDN. NFV. Fatiamento de Rede.

Abstract

Software-defined Networking (SDN) and *Network Functions Virtualisation* (NFV) approaches have been improved in the last years, and both have advantages, mainly because of the logically centralised control plane view. SDN introduces some new components, e.g. SDN controller and SDN applications, while in traditional networks, these elements did not exist. Some works point problems related to failures in SDN and its components, e.g. forward rules inconsistency and software or hardware failures. Concerning NFV, the efforts to virtualise classic network functions bring a considerable complexity to manage the architectural components, e.g. resource orchestration systems. In both approaches, there is no architectural structure prepared to deal with management. The network management field has been structured after the creation of the approaches. As manual network management in these complex architectures is not suitable, we suggest a new management architecture: *Self-organising Networks Architecture* (SONAr). Such new architecture bases on autonomic computing fundamentals from software engineer and *Self-organising Networks* (SON) concepts from telecommunications for employing network self-management. In this thesis, we present the SONAr and extend the self-healing fundamental for SDN and NFV by creating a SONAr-based framework, i.e. *Network Operations Self-healing* (NOSH). We deeply investigated the self-healing fundamental considering SDN and NFV particularities, including the fault tolerance peculiarities from Data, Control and Management layers. We developed two distinct implementations, and the evaluation proves that NOSH heals SDN and NFV layers from performance degradations and heals NOSH itself. Additionally, the evaluation demonstrates improvements in *Network Slice*s (NSs) with strict *Quality of Service* (QoS) requirements, e.g. in *Ultra-Reliable and Low Latency Communications* (URLLC), at run-time by using self-healing capabilities.

Keywords: Self-healing. Self-management. SDN. NFV. Network Slicing.

Lista de ilustrações

Lista de tabelas

Lista de siglas

ABM *Auto-boot Manager*

ACL *Autonomic Control Loop*

ACoE *Alarms Collector Entity*

AI *Artificial Intelligence*

AMQP *Advanced Message Queuing Protocol*

ANM *Autonomic Network Management*

API *Application Programming Interface*

BSLE *Behavior Self-learning Entity*

BSS *Business Support Systems*

CIM *Containerised Infrastructure Manager*

CN *Core Network*

CoE *Collector Entity*

CPI *Control Primitives Interceptor*

DCN *Data Centre Network*

DDoS *Distributed Denial-of-Service*

DHCP *Dynamic Host Configuration Protocol*

DPI *Deep Packet Inspection*

DSLE *Diagnostic Self-learning Entity*

E2E *End-to-End*

eMBB *Enhanced Mobile Broadband*

ETSI *European Telecommunications Standards Institute*

FACOM Faculdade de Computação

FCAPS *Fault, Configuration, Accouting, Performance, Security*

FFS *For Future Study*

HOEN *Hierarchical Orchestration of End-to-End Networks*

HTTP *Hypertext Transfer Protocol*

IE *Integration Entity*

IoT *Internet of Things*

LA *Local Agent*

LCoE *Logs Collector Entity*

LLDP *Link Layer Discovery Protocol*

MANO *Management and Network Orchestration*

MCC *Mission Critical Communication*

MCoE *Metrics Collector Entity*

MEHAR *Mondial Entities Horizontally Addressed by Requirements*

ML *Machine Learning*

MNO *Mobile Network Operator*

NAD *Network Administration Dashboard*

NBI *Northbound Interface*

NDB *Network Database*

NE *Network Element*

NEM *Network Event Manager*

NETCONF *Network Configuration Protocol*

NFV *Network Functions Virtualisation*

NFVI *NFV Infrastructure*

NM *Network Management*

NOSH *Network Operations Self-healing*

NS *Network Slice*

NSaaS *Network Slicing as a Service*

NSB *Network Service Broker*

NSLE *Natural Language Processing Self-learning Entity*

OAM *Operations, Administration, and Maintenance*

ODL *OpenDaylight*

ONF *Open Networking Foundation*

ONOS *Open Network Operating System*

OPEX *Operational Expenditure*

OSPF *Open Shortest Path First*

OSS *Operations Support Systems*

OVS *Open vSwitch*

OVSDB *Open vSwitch Database*

PSLE *Prediction Self-learning Entity*

QoE *Quality of Experience*

QoS *Quality of Service*

RAN *Radio Access Network*

RSLE *Rating Self-learning Entity*

RTT *Round Trip Time*

SBI *Southbound Interface*

SCE *Self-configuration Entity*

SCoE *Samples Collector Entity*

SDN *Software-defined Networking*

SDNC *SDN Controller*

SDR *Software-defined Radio*

SHE *Self-healing Entity*

SLA *Service Level Agreement*

SLE *Self-learning Entity*

SME *Self-management Entity*

SNMP *Simple Network Management Protocol*

SOE *Self-organising Entity*

SON *Self-organising Networks*

SONAr *Self-organising Networks Architecture*

SOPE *Self-optimisation Entity*

SP *Service Provider*

SPE *Self-protection Entity*

SPLE *Self-planning Entity*

SS7 *Signaling System number 7*

TAL *Tactical Autonomic Language*

TCoE *Topology Collector Entity*

TI Tecnologia da Informação

TN *Transport Network*

TSLE *Tuning Self-learning Entity*

UFU Universidade Federal de Uberlândia

URLLC *Ultra-Reliable and Low Latency Communications*

VIM *Virtualised Infrastructure Manager*

VM *Virtual Machine*

VNF *Virtual Network Function*

VNFM *VNF Manager*

VoIP *Voice over IP*

Sumário

Capítulo 1

Introdução

A evolução tecnológica nas últimas décadas resultou em novos requisitos de usuários/aplicações a serem considerados no núcleo das redes de computadores e em redes móveis. Para satisfazê-los, novas abordagens para controle e gerenciamento de rede foram propostas, sendo as Redes Definidas por Software – *Software-defined Networking* (SDN) – e a Virtualização de Funções de Rede – *Network Functions Virtualisation* (NFV) – as duas mais promissoras (RAMIREZ-PEREZ; RAMOS, 2016).

Em ambientes com SDN e NFV, o Gerenciamento de Rede – *Network Management* (NM) – possui alta complexidade, uma vez que diversos componentes novos são introduzidos, como por exemplo: controlador SDN, *VNF Manager* (VNFM) e Orquestradores de Infraestrutura. Além do gerenciamento dos componentes que já estavam presentes em redes tradicionais[1], os novos componentes introduzidos necessitam de gerenciamento e, muitas vezes, há certa complexidade para isso, uma vez que é primordial manter o funcionamento do plano de Controle.

O uso de SDN possui vantagens nítidas, como a topologia logicamente centralizada, serviços de rede programáveis e facilidade para experimentação de novos protocolos. No entanto, alguns problemas estão em aberto em SDN, como a saúde da rede (ABDELSALAM, 2018; COX et al., 2017). Nas redes tradicionais, protocolos distribuídos, sendo executados nos elementos de rede, gerenciam a saúde da rede e alteram rotas quando determinado *link* ou *switch* apresenta falhas. Em SDN, não há protocolos específicos para detecção de falhas, recuperação de elementos com falhas, alta disponibilidade de *links* etc (CHANDRASEKARAN; TSCHAEN; BENSON, 2016). SDN e NFV não foram desenvolvidas com funcionalidades de NM intrínsecas, ou seja, tais tecnologias necessitam de serviços suplementares, normalmente de terceiros, para gerenciamento de rede/recursos (FONSECA; MOTA, 2017).

Abdelsalam (2018) afirma que alguns desafios ainda existem no campo de SDN. Um desses desafios é lidar com inconsistências, visto que a separação dos planos de Controle

[1] Nesta tese, redes tradicionais significam arquiteturas anteriores à SDN/NFV, nas quais o controle e transmissão de dados eram feitos no mesmo plano (por exemplo a arquitetura TCP/IP da Internet).

e Dados faz com que informações coletadas pelo controlador possam estar desatualizadas. Informações não atualizadas podem implicar em problemas crônicos de consistência, provocando *loops* de roteamento/encaminhamento ou *black holes*. Com o surgimento de SDN, os desafios em aplicações[2] são consideráveis e precisam ser corretamente tratados, evitando problemas de desempenho, segurança ou escalabilidade (ABDELSALAM, 2018). Neste documento, considera-se que problemas vão eventualmente e inevitavelmente ocorrer na rede, e portanto a funcionalidade de autocura precisa existir em ambientes de SDN/NFV.

Alguns trabalhos apresentados na literatura buscam lidar com a manutenção da saúde[3] da rede em SDN e NFV. Para isso, aplicações SDN que rodam na *Northbound Interface* (NBI) são desenvolvidas para detectar e tratar falhas no plano de Dados. Contudo, erros no plano de Controle normalmente não são tratados. Ressalte-se que, para garantir saúde total de um ambiente SDN/NFV, é necessário que os planos de Dados, Controle e Gerenciamento (incluindo a camada de Aplicações de rede) estejam em perfeito funcionamento.

Redes de computadores e redes móveis contemporâneas tendem a utilizar SDN e NFV (LÓPEZ et al., 2017), como por exemplo, redes móveis de quinta geração (5G) e redes com suporte a Internet das Coisas – *Internet of Things* (IoT). Em ambientes 5G, diversos requisitos, tais como baixa latência, alto volume de dados, *small cells*, entre outros são esperados. Em IoT, bilhões de dispositivos com milhões de Elementos de Rede – *Network Element*s (NEs) (*switches* e roteadores) são introduzidos (LÓPEZ et al., 2017). Nota-se que é humanamente impossível lidar manualmente com o gerenciamento de bilhões de dispositivos/usuários/aplicações e requisitos novos (surgindo a todo instante).

Para tratar aspectos de gerenciamento nessas redes pode-se utilizar o conceito de Computação Autônoma – *Autonomic Computing*, introduzido por Ganek e Corbi (2003). Nesse conceito, a cura, otimização, configuração e proteção de um sistema são feitos autonomamente pelo próprio sistema. Para isso, Ganek e Corbi (2003) introduziram quatro fundamentos principais: autocura (*self-healing*), autoconfiguração (*self-configuration*), auto-otimização (*self-optimisation*) e autoproteção (*self-protection*).

Os conceitos de computação autônoma são utilizados há alguns anos nas redes móveis de telefonia (redes celulares), sendo intitulados como *Self-organising Networks* (SON). Pode-se considerar que a telefonia está avançada nesse quesito, visto que especificações (3GPP, 2018a; 3GPP, 2018c; 3GPP, 2018d) e protocolos para *self-**[4] já foram padronizados e implementados em equipamentos da rede de acesso. Contudo, o núcleo (*core*) das redes não apresenta nenhum padrão para gerenciamento.

[2] Nesta tese, o termo aplicação significa aplicações de programabilidade de rede.

[3] Considera-se que uma rede é saudável quando os nós e *links* da topologia de rede e os componentes de gerenciamento estão executando no estado normal (sem apresentar falhas).

[4] *Self-healing, self-configuration, self-protection, self-optimisation, self-orchestration, self-awareness, self-learning* etc (GANEK; CORBI, 2003; RAMIRO; HAMIED, 2011; VERMESAN et al., 2017; SADIGHI et al., 2018; STEIN et al., 2018).

Em SDN e NFV, como em diversas outras arquiteturas propostas ao longo dos anos, o gerenciamento não foi tratado nos primórdios, tendo sido incorporado posteriormente (FONSECA; MOTA, 2017). O grupo de pesquisa *Mondial Entities Horizontally Addressed by Requirements* (MEHAR), da Universidade Federal de Uberlândia (UFU), propõe uma arquitetura para autogerenciamento de redes de computadores e móveis (incluindo SDN e NFV), denominada *Self-organising Networks Architecture* (SONAr). Nessa arquitetura, procedimentos usados em SON e demais fundamentos de computação autônoma são utilizados para garantir o gerenciamento autônomo de redes de computadores corporativas e núcleo de redes móveis, com o mínimo de intervenção manual. Nesta tese, elaborou-se um *framework* baseado em NFV, denominado *Network Operations Self-healing* (NOSH), para *Operations, Administration, and Maintenance* (OAM) (MIZRAHI et al., 2014) a partir da SONAr.

A presente tese versa especificamente sobre o fundamento de autocura. Esse fundamento tem como objetivo detectar ou prever falhas em determinado sistema e recuperá-lo (voltar o sistema para o estado saudável). Em um sistema *standalone*, a autorrecuperação não é tão complexa quanto em redes de computadores/móveis, nas quais elementos de rede distribuídos e os componentes de controle/gerenciamento formam uma topologia de n elementos. É preciso monitorar (para detectar e prever falhas) os n elementos e executar procedimentos e algoritmos para recuperá-los de eventuais falhas.

Esta proposta modela uma rede de computadores em três planos nos quais o autogerenciamento deve ser capaz de executar procedimentos de autocura, sendo: (i) plano de Dados/Infraestrutura; (ii) plano de Controle; e (iii) plano de Gerenciamento. O plano de Dados usualmente entra no estado não-saudável quando elementos de rede (*switches* e roteadores) ou *links* falham. As falhas podem ser ocasionadas por erros de software ou hardware, ou até mesmo por intervenções manuais que causem problemas de inconsistência (como *loops* ou *black holes*). O plano de Controle apresenta falhas quando seus componentes (controladores e aplicações SDN) estão incomunicáveis ou apresentam mau funcionamento. Por fim, o plano de Gerenciamento também pode apresentar falhas. No caso da SONAr, os serviços da própria arquitetura podem estar indisponíveis.

Assume-se que capacidades de autocura devem ser incorporadas pelos três planos apresentados no parágrafo anterior. Para isso, a presente tese apresenta a incorporação do fundamento de autocura dentro da SONAr, buscando lidar com os principais problemas que podem levar a rede para um estado não-saudável (disponibilidade, capacidade e inconsistência). Ainda, para esta tese, duas implementações distintas para o *framework* NOSH foram realizadas, sendo a primeira com foco em autogerenciamento dos planos de controle e gerenciamento e a segunda com foco na manutenção autônoma de qualidade de serviço em tempo de execução. As implementações foram implantadas em diferentes *testbeds* para avaliações que serão discutidas mais a frente no presente documento.

1.1 Motivação

A principal motivação do presente trabalho é o crescimento de redes de computadores e a introdução de novos requisitos em aplicações novas. Assume-se que o gerenciamento manual de bilhões de dispositivos é algo impraticável. Além disso, alguns conceitos flexibilizam a otimização de redes para uso de aplicações com propósitos e intenções diferentes, como por exemplo Fatiamento de Rede – *Network Slicing*. Esses novos conceitos tornam complexo o gerenciamento de redes e introduzem a necessidade de funções automatizadas para lidar com eles.

Além disso, o presente trabalho é motivado por ambientes corporativos e de telecomunicações. Esses ambientes normalmente exigem requisitos *carrier-grade* (alta disponibilidade, escalabilidade, tolerância a falhas e latência baixa) para oferecer serviços de alta qualidade para clientes. Uma falha em um elemento de rede pode deixar diversos serviços indisponíveis e, no atual cenário de alto uso da Internet, qualquer indisponibilidade de alguns segundos significa perda financeira considerável.

O autogerenciamento de redes possibilita ainda a redução de custos com *Operational Expenditure* (OPEX) pelas organizações e minimiza a possibilidade de falhas humanas. Por exemplo, em uma Operadora de Redes Móveis – *Mobile Network Operator* (MNO), uma falha em determinado segmento de rede requer que administradores de rede atuem manualmente para lidar com falhas que ocasionalmente poderiam ser tratadas autonomamente. Ademais, equipes de analistas/técnicos de MNOs precisam estar alocadas continuadamente para monitoramento da rede/aplicações, o que pode ser realizado seguramente de forma autônoma.

1.2 Objetivos e Desafios da Pesquisa

O objetivo geral deste trabalho é a incorporação de aspectos de autocura na arquitetura SONAr, para garantir o funcionamento de redes SDN/NFV, permitindo que falhas nos planos de Controle, Dados e Gerenciamento possam ser resolvidas com a mínima intervenção humana. Para se alcançar o objetivo supracitado, propõe-se os seguintes objetivos específicos:

1. Especificação dos componentes e interoperabilidade dentro da SONAr necessários para rodar as funcionalidades de autocura;

2. Implementação dos serviços para monitoramento do plano de Dados considerando-se três abordagens: *Control loop*, Agentes locais e Interceptação de primitivas na *Southbound Interface* (SBI);

3. Implementação dos serviços para monitoramento do plano de Controle e plano de Gerenciamento;

4. Aplicação de ações de recuperação clássicas (praticadas em SON) para recuperar falhas nos planos de Dados – considerando *Network Slice*s (NSs) –, Controle e Gerenciamento;

5. Avaliação de técnicas de monitoramento por detecção *vs* predição e técnicas de recuperação *real-time* e *bucket-based*;

6. Demonstração da solução final, o *framework* NOSH, em um ambiente SONAr, executando todos os serviços e componentes em forma de funções de rede virtualizadas.

1.3 Hipótese

Assume-se a seguinte premissa: o fundamento de autocura aplicado às redes de comunicação é essencial para a existência de redes autonômicas.

Redes autonômicas normalmente separam o plano de Controle do plano de Dados e, também, há um direcionamento da *Open Networking Foundation* (ONF) para a utilização de um terceiro plano (plano de Gerenciamento). Portanto, espera-se que técnicas de autocura possam ser aplicadas nos três planos de redes autonômicas para prover qualidade de serviço e experiência de entidades comunicantes. Essa hipótese será detalhada e avaliada na presente tese.

1.4 Contribuições

Computação autônoma é realidade em diversas áreas da Engenharia de Software, bem como em redes de telecomunicações. Em redes de telecomunicações, no geral, e em redes móveis, em particular, todos os esforços de projeto, configuração, implantação e manutenção são focados na qualidade de um serviço específico: o serviço de voz. Pelo menos esta é a realidade da sinalização *Signaling System number 7* (SS7) de redes de telecomunicações, de protocolos *Voice over IP* (VoIP) e redes móveis até a geração 4G. É nosso entendimento que redes baseadas na Tecnologia 5G e pós-5G suscitarão novas expectativas em serviços disponibilizados através de NFV.

Considerando que novas tecnologias suscitam novas expectativas, pode-se antever que haverá maior rigor na qualidade de serviços oferecidos em geral e, particularmente, para aqueles classificados como *Ultra-Reliable and Low Latency Communications* (URLLC) das redes 5G. O presente trabalho propõe aplicar autocura, um dos principais fundamentos de computação autônoma, em abordagens contemporâneas de redes de computadores (como SDN e NFV). A primeira contribuição nítida é a entrega de uma solução para detecção e recuperação de falhas nessas redes, que pode ser evoluída futuramente para uso na indústria de telecomunicações, em provedores de acesso à Internet ou em redes corporativas de alta escala.

Incorporar capacidades de autocura à evolução da arquitetura SONAr é uma contribuição adicional que esta tese legará na forma de serviços de autocura, necessários à manutenção de elementos, que ela gerencia, em estado de perfeito funcionamento. Assim sendo, os demais pesquisadores do grupo MEHAR podem direcionar seus esforços para outros fundamentos de autogerenciamento, tais como autoconfiguração, auto-otimização ou autoproteção.

A presente tese deixa ainda contribuições acadêmicas, pois alguns trabalhos, discutidos no Capítulo 2, se preocupam em analisar problemas do plano de Dados de SDN. Não foram encontrados, até o momento em que o levantamento sistemático foi feito para esta tese, trabalhos dedicados à manutenção da saúde do plano de Controle. Destaque-se que, sem o plano de Controle, toda a rede fica inoperável. Deste modo, esta tese pretende definir como fundamentos de autocura podem ser aplicados ao plano de Controle e, por consequência, aplicados nos demais planos (Dados e Gerenciamento) da rede. Ademais, as funcionalidades de autocura idealizadas nesta tese são capazes de tratar falhas de *Quality of Service* (QoS) de NSs – nós acreditamos que se um determinado NS não está atingindo o *Service Level Agreement* (SLA) de QoS previamente estabelecido no aprovisionamento, esse NS está falhando (mesmo que a comunicação bruta pelo NS não tenha sido interrompida). Até onde nossa investigação pôde abordar, não foram encontradas soluções efetivas para manutenção de QoS de NSs durante a fase de *run-time* como estamos propondo.

Finalmente, destaca-se a contribuição para a indústria nacional. Normalmente, o Brasil importa soluções de monitoramento e gerenciamento de redes, aplicações e infraestrutura. É incomum encontrar grupos que desenvolvam soluções similares ao NOSH no Brasil. Geralmente, os grupos de pesquisa envidam esforços na utilização e no aprimoramento de soluções estrangeiras. A implementação de código, a partir da especificação elaborada nesta tese, é direcionada à construção de um produto, no qual serão utilizados *frameworks* e linguagens de programação padrões de mercado. O conhecimento adquirido neste processo fica como contribuição do trabalho, além do produto final que pode ser explorado futuramente na indústria ou, porventura, na comunidade *open-source*.

1.5 Organização da Tese

Esta tese está organizada em seis capítulos, sumarizados a seguir:

1. O Capítulo 1 contextualiza o trabalho proposto de forma sucinta, justificando a pesquisa proposta e sua relevância, e aponta onde o tema do trabalho se encaixa, bem como são apontados motivação, objetivos da pesquisa, hipótese e as contribuições ocasionadas desta tese;

2. O Capítulo 2 apresenta aspectos conceituais e arquiteturais de redes SDN/NFV e a abordagem SON utilizada nas redes móveis e que serve de base para o presente

trabalho. Por fim, se apresenta o estado da arte de pesquisas com foco em autogerenciamento e, particularmente, em autocura, elucidando as principais falhas que podem eventualmente ocorrer nos planos de Dados, Controle e Gerenciamento;

3. O Capítulo 3 apresenta uma visão geral da arquitetura SONAr, na qual o presente trabalho está inserido, descrevendo os principais serviços da arquitetura, a integração entre os componentes internos e externos (inter-camadas de Dados, Controle e Gerenciamento) e as duas implementações realizadas até o presente momento a partir da arquitetura;

4. O Capítulo 4 apresenta a proposta da presente tese, incluindo os novos conceitos de *Slices* de Gerenciamento e Catálogo de Gerenciamento, pontos fulcrais do trabalho. Detalha-se ainda o *framework* NOSH desenvolvido a partir da SONAr e sua interoperabilidade, a inicialização autônoma e programática dos recursos utilizados para autocura e aspectos de monitoramento e recuperação elaborados nesta tese, revelando-se detalhes de implementação;

5. O Capítulo 5 traz o método para a avaliação utilizado nos experimentos e os detalhes dos ambientes de experimentação utilizados, bem como as configuração previamente aplicadas. Em seguida, resultados são apresentados para comprovar a eficácia do *framework* e uma análise de resultados é descrita;

6. O Capítulo 6, por fim, aponta as principais contribuições do trabalho, as propostas de trabalhos futuros e as de contribuições em produção bibliográfica.

Capítulo 2

Fundamentação Teórica

Este capítulo apresenta os embasamentos filosóficos, conceituais e arquiteturais que formam a base referencial do presente trabalho. Inicialmente, as redes de computadores são contextualizadas considerando o cenário atual. Logo após, as redes móveis, motivadoras para o trabalho, são explanadas. Em seguida, alguns trabalhos de grupos de pesquisadores relacionados a autogerenciamento – *self-management* – são analisados. Por fim, o fundamento de autocura – *self-healing* – é esmiuçado no contexto de redes de computadores/móveis baseadas em SDN/NFV, considerando-se os planos de Controle, Gerenciamento e Dados.

As redes de computadores contemporâneas são constituídas de vários protocolos para operação, administração e manutenção. Além desses protocolos, novas abordagens foram propostas há alguns anos e têm sido aprimoradas pelas indústria e academia. Saliente-se que muitas propostas de tecnologias/protocolos se baseiam em redes de telecomunicações. Essas redes são robustas e, além de roteadores/*switches*, são constituídas de elementos para controle de sinalização, tarifação/bilhetagem, controle de políticas de navegação e tolerância a falhas.

Na área de computação é possível afirmar que as redes cresceram em duas subáreas distintas: (i) Telecomunicações; e (ii) Tecnologia da Informação (TI). MNOs e provedores Web – *Service Provider*s (SPs) – usualmente utilizam tecnologias e protocolos padronizados em (i), enquanto ambientes corporativos e empresas de *Cloud Computing* empregam tecnologias desenvolvidas em (ii). O termo "Redes Convergentes" não é novo e espera-se que, embora o Protocolo IP tenha sido largamente utilizado em telecomunicações, em um futuro próximo, as duas subáreas convirjam para uma única infraestrutura. Ademais, não seria ilógico dizer que as evoluções tecnológicas em ambas as subáreas tendem a utilizar as mesmas premissas, como por exemplo, o conceito de compartilhamento de infraestrutura utilizado para *multi-tenant* em *cloud computing* na subárea de TI e o conceito de compartilhamento de recursos utilizado para *Network Slicing* na subárea de Telecomunicações (Foukas et al., 2017).

A Seção 2.1 descreve duas abordagens distintas (porém complementares) que ambici-

onam se tornar o padrão de redes para os próximos anos.

2.1 Aspectos Conceituais de Redes de Computadores

A abordagem SDN (COX et al., 2017) surgiu para flexibilizar a operação e manutenção de redes de computadores convencionais. Nessa abordagem, o controle é feito fora da banda de dados (*out-of-band*), e um modelo aberto é adotado (em contrapartida ao modelo fechado dos grandes fornecedores de equipamentos de rede) e a possibilidade de testar novos protocolos de forma rápida torna-se viável (COX et al., 2017).

Concomitantemente à SDN, a área de telecomunicações trouxe o conceito de NFV (ETSI, 2014). Nesse tipo de abordagem, virtualização é a base para suportar elementos com funções de rede utilizadas amplamente em telecomunicações. Essas funções, conceitualmente denominadas *Virtual Network Function*s (VNFs), utilizam um modelo aberto no qual software se torna a base das aplicações (de controle). Assim sendo, funções do *core* de rede (como controle de sinalização, manutenção de sessão, *firewall* ou tarifação) são fornecidas em ambientes virtualizados, permitindo escalabilidade, manutenção menos complexa e recuperação de falhas.

Note-se que SDN e NFV possuem pontos em comum e, apesar de serem usados para propósitos diferentes, devem coexistir futuramente em redes de computadores e redes móveis (BLANCO et al., 2017). Para exemplificar, pode-se citar Zhang, Xie e Yang (2015), que apresentam uma arquitetura para redes 5G baseada em SDN e NFV. Zhang, Xie e Yang (2015) afirmam que separar o plano de Dados do plano de Controle é um recurso importante para 5G (redes 5G prometem capacidade para alto volume de tráfego, baixa latência, bilhões de dispositivos conectados, latência de 1 ms etc).

Por se considerar que SDN e NFV são abordagens maduras, amplamente conhecidas, então não temos de nos deter nos pontos positivos. Diferentemente do senso comum, desvantagens trazidas por SDN e NFV serão analisadas. A Subseção 2.1.1 apresentará sumariamente as duas abordagens e apontará os problemas que estão em aberto e foram apontados por outros trabalhos.

2.1.1 SDN e NFV: *Status Quo*

A Figura 1 apresenta a arquitetura geral de SDN, cujos elementos básicos são: o plano de Dados (*Data Plane*), constituído de diversos NEs; o Controlador SDN (*Controller*); e uma camada de Aplicações (SDN*Apps*). A comunicação do plano de Dados com o Controlador é feita através da interface SBI, que normalmente utiliza o protocolo OpenFlow, podendo-se dizer que a interoperabilidade é garantida por esse protocolo (COX et al., 2017).

Todavia, não se pode afirmar que OpenFlow é o protocolo padrão da SBI, uma vez que outros protocolos SBI são encontrados na literatura, como por exemplo ForCES (DORIA

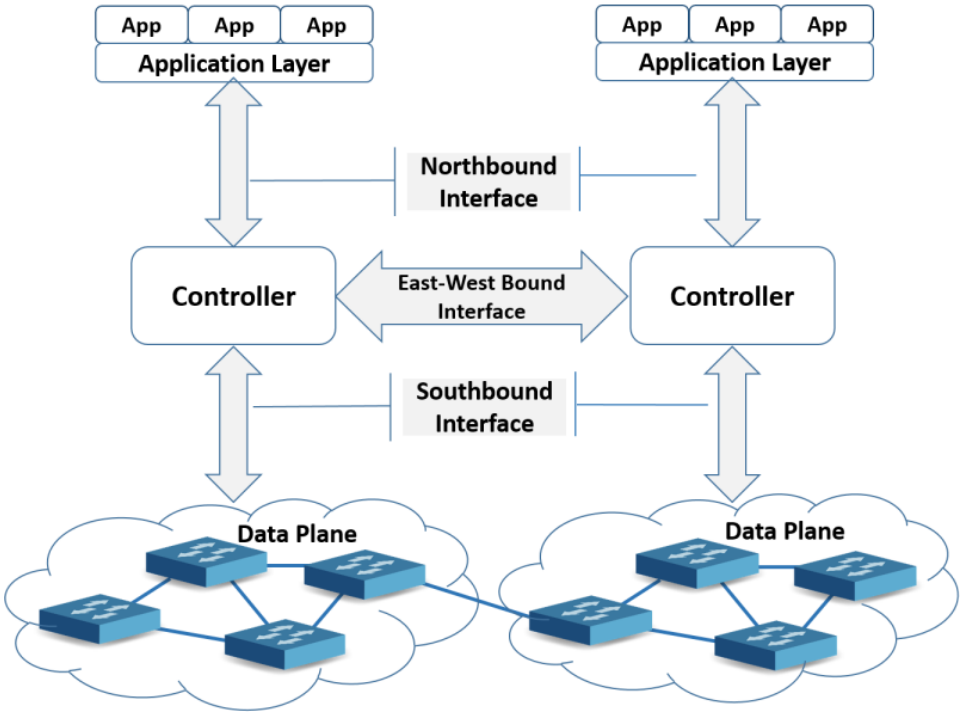


Figura 1 – Interfaces na Arquitetura SDN (COX et al., 2017).

et al., 2010) e NETCONF (ENNS et al., 2011). Porém, devido ao amplo uso na academia e indústria, pode-se inferir que OpenFlow é uma opção razoável para essa interface. Por outro lado, a NBI, que provê a comunicação entre *Apps* e *Controllers*, não conta com um protocolo padronizado ou amplamente utilizado.

Na Figura 1 há diversos aspectos a serem gerenciados, incluindo (mas não se limitando a): falhas de software ou hardware em qualquer elemento da figura, alta carga e inconsistência em tabelas de fluxo/roteamento. Nas redes tradicionais, as falhas em elementos de rede acontecem apenas no plano de Dados (onde o controle é executado *in-band*) (CHANDRASEKARAN; TSCHAEN; BENSON, 2016; FONSECA; MOTA, 2017; AZAB; FORTES, 2017).

Em SDN, além de NEs do plano de Dados, há elementos do plano de Controle que podem falhar. Falhas de software ou erros de configuração podem ser encontrados em Controladores SDN e em diversas *Apps* que podem estar em execução nesses controladores. Enfatize-se que a falha de um controlador pode afetar drasticamente a saúde da rede, uma vez que a centralização da topologia não permite que protocolos distribuídos, como o *Open Shortest Path First* (OSPF), efetuem a recuperação de falhas (como ocorre em redes tradicionais). Cox et al. (2017) afirmam que há diversas pesquisas em andamento visando aspectos tais como: minimização de *bugs*; consistência; recuperação rápida; algoritmos para serem aplicados em SDN e NFV; previsão de *deadlock*/*livelock*; algoritmos de *rollback*; entre outras.

Outros aspectos de gerenciamento em uma arquitetura SDN, como a ilustrada na

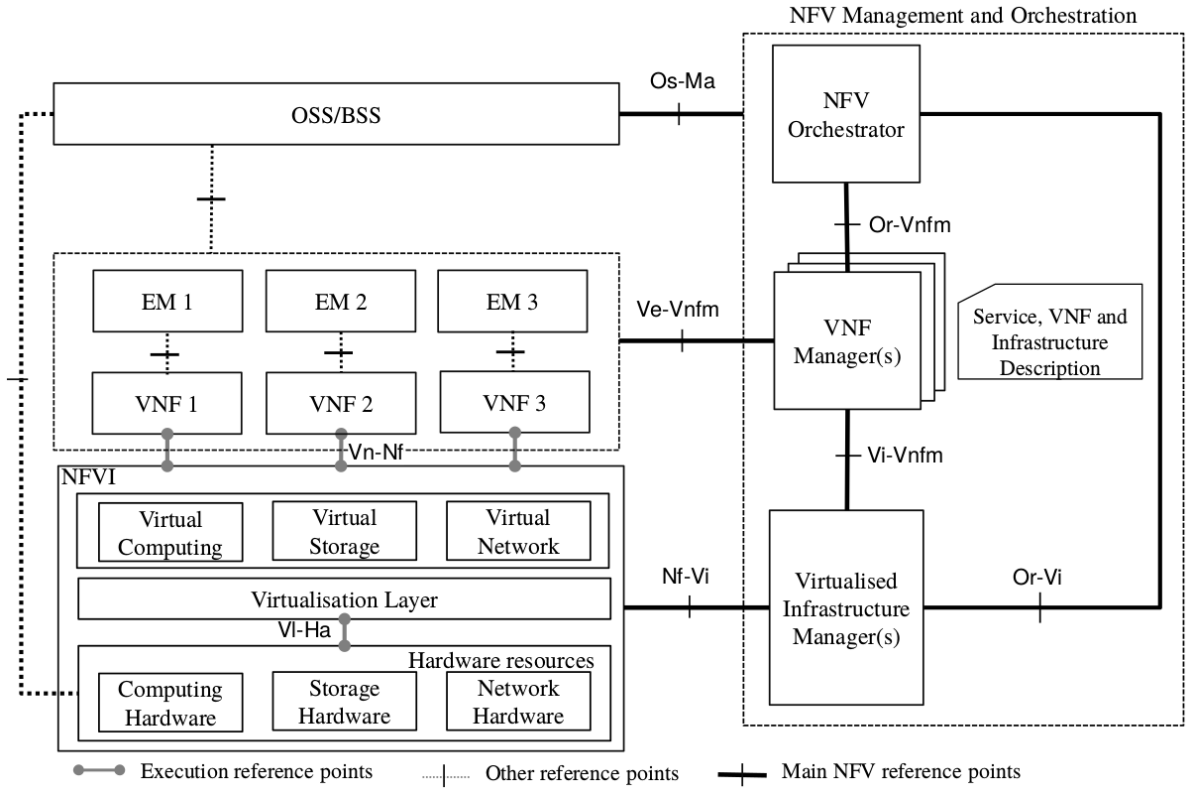


Figura 2 – Arquitetura NFV (ETSI, 2014).

Figura 1, são, entre outras, questões de monitoramento, congestionamento de *links* e auto-organização de fluxos em *switches*. O último ponto significa a realocação de determinadas regras de encaminhamento em *switches* devido a fatores externos, como segurança, distribuição de tráfego, falhas, busca por atender determinado QoS, entre outros.

Além das possibilidades de falhas citadas, destaque-se que todos os elementos (mesmo os de controle) podem ser conectados por meio do plano de Dados. A *East-West Bound Interface*, que realiza a comunicação entre domínios distintos, também pode ser prejudicada, caso *links* que interliguem domínios distintos apresentem falhas.

O presente trabalho objetiva tratar os problemas supramencionados em um ambiente SDN. Note-se que esses ambientes, para atender requisitos de operação e manutenção, inserem certa complexidade (com a inserção de novos elementos para controle). Além de SDN, ambientes NFV também trazem diversos elementos novos que podem aumentar a complexidade de gerenciamento.

A Figura 2 apresenta o *framework* arquitetural da abordagem NFV, padronizado pelo *European Telecommunications Standards Institute* (ETSI). É possível notar a separação entre as camadas Orquestração e Infraestrutura. Basicamente, os recursos computacionais (por exemplo *Computing*, *Storage* e *Network*) são fornecidos de modo virtualizado na camada de Infraestrutura, denominada *NFV Infrastructure* (NFVI). O *Virtualised In-*

frastructure Manager (VIM) é responsável por gerenciar recursos virtuais e de hardware, enquanto o VNF *Manager* gerencia as funções de rede virtualizadas propriamente ditas.

Nitidamente, uma rede SDN pode ser fornecida em uma arquitetura NFV através da NFVI. É importante destacar que qualquer camada (ou componente) representada na Figura 2 pode apresentar falhas. Além disso, em termos de redes de computadores, esses componentes podem estar distribuídos em diferentes NEs. Em Han et al. (2015) é possível encontrar uma discussão das oportunidades e dos desafios relacionados a NFV. Observe-se que a arquitetura NFV padrão (ETSI, 2014) possui diversos elementos novos, conforme ilustrado na Figura 2, que carecem de automação para gerenciamento.

As Funções Virtualizadas – *Virtual Network Function*s (VNFs), tais como roteadores, *gateways*, *proxies*, *firewalls*, e outros – precisam trabalhar ininterruptamente. Pode-se afirmar que mecanismos para gerenciamento de rede acoplados ou naturais em SDN e NFV precisam garantir que as VNFs estejam sempre disponíveis na infraestrutura, mantendo-as ininterruptamente mesmo na migração de uma delas (COX et al., 2017; LÓPEZ et al., 2017). Alguns protocolos/tecnologias são encontrados com o propósito de ajudar o monitoramento e gerenciamento nesse tipo de rede, como o OF-Config – baseado em NETCONF – ou o AdVisor (WICKBOLDT et al., 2015).

2.1.2 Análise dos Planos: Dados *vs* Controle *vs* Gerenciamento

O plano de Controle separado logicamente do plano de Dados é o ponto fulcral da filosofia SDN. Na telefonia, essa separação ocorreu anteriormente, sendo denominada Canal Comum (tráfego de sinalização) e tráfego de voz/dados. Considerando-se o sucesso dessa separação nas redes de telecomunicações, entende-se que em redes de computadores a tendência é que essa partição também seja adotada, quer com o uso de SDN, quer com outras propostas.

Normalmente, o gerenciamento da rede está diretamente ligado ao controle. Em SDN, pode-se afirmar que os elementos de gerenciamento estão no plano de Controle (COX et al., 2017), apesar de terem o propósito de gerenciar componentes de ambos os planos. Curiosamente, trabalhos de gerenciamento em SDN chegaram a propor um terceiro plano, denominado Plano de Gerenciamento – *Management Plane* – separado do plano de Controle. Wickboldt et al. (2015) listam os desafios de gerenciamento em redes SDN e propõem o terceiro plano mencionado. Os requisitos de gerenciamento comumente citados e mais desafiadores são: disponibilidade e resiliência; ferramentas para implantação (*deploy* e *rollback*); desempenho e escalabilidade; isolamento e segurança; planejamento de capacidade; e monitoramento e visualização (WICKBOLDT et al., 2015).

Outro trabalho que acredita em um Plano de Gerenciamento separado é Abdallah et al. (2018). O trabalho propõe um *framework* para *Fault, Configuration, Accouting, Performance, Security* (FCAPS) e afirma que *Fault* e *Security* são atribuições do controlador, porém *Configuration*, *Accouting* e *Performance* requerem análises mais prolongadas e,

portanto, podem ser atribuídas ao Plano de Gerenciamento. O presente trabalho entende que essa afirmação é parcialmente verdadeira, uma vez que o tratamento de falhas (*Fault*) pelo controlador normalmente se limita a falhas em NEs do plano de Dados. Todavia, o plano de Controle pode apresentar falhas, bem como o próprio Plano de Gerenciamento. A Seção 2.4 apresenta algumas topologias SDN e NFV em detalhes e discorre sobre o que seria autocura em cada uma das camadas nessas topologias – a saber camadas de Gerência (planos de Controle e Gerenciamento) e Infraestrutura (plano de Dados).

Nesta seção foi possível notar que diversos pontos para gerenciamento em SDN e NFV são problemas em aberto. Através do plano de Gerenciamento, a arquitetura SONAr, especificada no presente trabalho, pode ser concretizada. Essa arquitetura se inspira em conceitos da telefonia, que serão descritos na Seção 2.2, transportados para ambientes SDN e NFV.

2.2 Redes Móveis

Conforme descrito na Seção 1.3, os conceitos de SON são aplicados no presente trabalho com intenções de autocura. As SONs foram projetadas para telecomunicações com o intuito de permitir que redes de diferentes operadoras possam ser compartilhadas. Dessa forma, qualquer nó móvel (*smartphone*, *phone*, *tablet* etc) pode mudar de célula autonomamente, sem que a conexão seja perdida (3GPP, 2018b). Esta seção traz a fundamentação teórica de SON.

O conceito das Redes Auto-organizáveis – *Self-organising Networks* (SON) – constitui-se de três funcionalidades básicas: auto-estabelecimento (*self-establishment*), autocura (*self-healing*) e auto-otimização (*self-optimisation*).

O auto-estabelecimento é responsável por configurações feitas automaticamente antes que a rede entre no estado operacional, bem como as configurações necessárias quando um novo nó (por exemplo *eNodeB*) é implantado no ambiente (3GPP, 2018a). A especificação de *self-establishment*, 3GPP (2018a), é feita em alto nível de abstração e a maior parte dos itens estão marcados como estudo futuro – *For Future Study* (FFS). A especificação aborda aspectos, como por exemplo *download* de software, instalação e aplicação de *eNodeB*. Não há padronização para tratar a autoconfiguração da integração entre *eNodeB* com o *core* (gerência).

A auto-otimização é responsável por alterações que são feitas enquanto a rede está no estado operacional. Essas alterações costumam ser para *load balancing*, adição de novas células ou alteração de células vizinhas dinamicamente. Pode-se afirmar que após o início (*setup*) da rede, as alterações autônomas executadas são responsabilidade das funções de otimização conforme especificação de *self-optimisation* em 3GPP (2018b).

Por fim a autocura (foco deste trabalho) é responsável por identificar/prever falhas e solucioná-las/mitigá-las autonomamente. A especificação 3GPP (2018d) padroniza as

funções de *self-healing* nas redes de telecomunicações e foi usada como base para a proposta que será apresentada no Capítulo 4. Basicamente, a entidade de autocura monitora e gera alarmes que servem como gatilhos para recuperação autônoma. As ações de recuperação em falhas de *software* são essencialmente reinicialização do sistema, reabertura de backup de software, ativação de *fallback*, *download* de unidade de software ou reconfiguração de recursos. Já as ações de recuperação de hardware dependem de recursos existentes e do nível de redundância aplicada, mas basicamente essas ações são: isolar e remover recursos com falhas; remover recursos físicos; ou reiniciar um recurso (3GPP, 2018d).

Apesar de SON ser um conceito antigo e robusto (aplicado em redes móveis produtivas há anos), seus recursos ainda são motivo de investigação. SDN e *Software-defined Radio* (SDR) foram usados em SON para gerenciamento (RAMIREZ-PEREZ; RAMOS, 2016). Assim como SDN, a abordagem SDR incorpora elementos programáveis com flexibilidade e adaptabilidade. Ramirez-Perez e Ramos (2016) afirmam que já existem arquiteturas que permitem que SDR forneça funções para NFV (que possui suporte a SDN). O uso de SON nesses ambientes pretende adicionar capacidades e inteligência nos elementos de rede para que eles executem procedimentos de gerenciamento automaticamente, mantendo a rede performática sempre que estiver no estado operacional.

Observa-se que as especificações de SON padronizadas não descrevem os algoritmos utilizados para as funções *self-**. A Seção 2.3 vai descrever brevemente os conceitos de computação autônoma com foco em *self-healing* encontrados na engenharia de software em TI. Tais conceitos são similares àqueles das SONs das telecomunicações e também são usados como fundamentação teórica para o presente trabalho.

2.3 Gerenciamento de Rede

Esta seção apresenta os requisitos e as tecnologias chaves no contexto de redes de computadores com vistas ao Gerenciamento de Redes – *Network Management* (NM). Inicialmente, os preceitos de gerenciamento e operação de redes são explicados e, posteriormente, são descritos fundamentos de computação autônoma para se buscar autogerenciamento. Em seguida, a Subseção 2.3.2 contém os principais trabalhos relacionados ao tema da presente tese. Portanto, esta seção trata genericamente sobre autogerenciamento enquanto a Seção 2.4 apresenta a funcionalidade de autocura para os planos de Dados, Controle e Gerenciamento, destacando os desafios não resolvidos nesta matéria.

O conceito de Gerenciamento de Rede amplamente aceito afirma que se trata da operação de recursos de redes computacionais, passando por provisionamento, controle e monitoramento (DING, 2016; HEGERING, 1999; SUBRAMANIAN, 2010). Dessa forma, técnicas para OAM são utilizadas para administrar e manter os sistemas que compõem uma rede de computadores (CLEMM, 2006). Pode-se dividir o gerenciamento dos recursos

de rede em cinco etapas, sendo denominadas FCAPS, quais sejam: (i) gerenciamento de falhas; (ii) configuração e alocação de recursos; (iii) contabilização ou medição de recursos usados; (iv) avaliação de desempenho; e (v) aplicação de políticas de segurança (ITU, 1993; STEVENSON, 1995).

Redes futuras (como redes 5G) vão dar suporte a novos serviços (*e-health*, *e-commerce*, *streamming* etc) e serão concebidas através de SDN, NFV e SON (LÓPEZ et al., 2017; ZAIDI et al., 2018). Pode-se afirmar que essas redes focam em segurança, escalabilidade, robustez e rápida implantação de novas aplicações. López et al. (2017) afirmam que três campos devem ser melhorados para fornecer os requisitos de redes futuras: rádio (*radio*), operação e gerenciamento.

O gerenciamento de recursos de maneira menos complexa pode ser realizado através de SDN, enquanto NFV pode garantir gerenciamento e orquestração de funções de redes através de instâncias virtualizadas (BOURAS; KOLLIA; PAPAZOIS, 2017). Além disso, pode-se dizer que a modificação ou customização de fluxos em redes tradicionais requer configuração em cada dispositivo e implantação de novos serviços/protocolos levam muito tempo (o que pode ser minimizado por SDN). Essencialmente, o gerenciamento de redes lida com: falhas em *links*; congestionamento; controle de *Distributed Denial-of-Service* (DDoS); e latência (LÓPEZ et al., 2017).

Em relação a falhas, pode-se dizer que SDN facilita o gerenciamento, visto que a rede torna-se programável e com topologia centralizada. Ressalva-se que falhas ocorrem quando a rede não consegue entregar um serviço e, além disso, erros podem ocorrer quando a rede entra em algum estado incorreto. Nas redes tradicionais, o gerenciamento de falhas/erros é complexo pois o modelo não é aberto (fornecedores vendem equipamentos no modelo 'caixa fechada'). Todavia, mesmo que SDN possua um modelo aberto, pode-se afirmar que novos pontos de falha/erros são introduzidos (devido a novos elementos de controle terem surgido). Esclarece-se que as maiores falhas em *switches* e roteadores são *bugs* de software, falhas em hardware, protocolos mal configurados e fatores externos (FONSECA; MOTA, 2017).

Neste ponto, pode-se elucidar o conceito de Fatiamento de Rede – *Network Slicing*. Esse conceito é essencial para o desenvolvimento de redes futuras (como 5G e pós-5G) que devem suportar aplicações com requisitos diferentes, como latência, alta disponibilidade e alto volume de dados (BERGSTRA; BURGESS, 2011), rodando no mesmo ambiente (Afolabi et al., 2018). Com NSs, a alocação de recursos pode ser feita considerando níveis – SLAs – de QoS requeridos pelas aplicações. Ademais, MNOs podem desfrutar de novos modelos de negócio, como *Network Slicing as a Service* (NSaaS) (Zhou et al., 2016). Clarifica-se que as falhas nas redes tradicionais, citadas no parágrafo anterior, ocorrem quando um determinado problema ou erro acontece e a rede fica inoperável. Porém, considerando *Network Slicing*, podemos afirmar que uma falha não ocorre necessariamente quando um problema ou erro aconteceu, mas quando um NS não está conseguindo entregar

o QoS preliminarmente acordado. Por exemplo, em uma rede tradicional, uma rota apresenta falha quando um determinado link no caminho está indisponível (*down*). No *Network Slicing*, esse mesmo link pode estar disponível (*up*) mas com uma latência/*Round Trip Time* (RTT) maior que 1 ms; os NSs que requerem latência inferior a 1 ms vão estar irrefutavelmente falhando.

Nas redes tradicionais, pode-se afirmar que uma falha fica restrita, próxima ao elemento de rede no qual ela inicialmente ocorreu, visto que há caminhos alternativos para o tráfego. Em se tratando de SDN, uma falha precisa ser isolada pois, se atingir o controlador, pode culminar em *livelocks* ou *deadlocks* (FONSECA; MOTA, 2017). Os problemas de gerenciamento de falhas surgidos com SDN e que ainda não foram resolvidos são: local do controlador (em qual trecho da topologia ele deve estar instanciado); detecção de falhas da rede de controle; localização de falhas no controle; recuperação de falhas de controle; confiabilidade da comunicação entre *switches* e controlador; e violação da consistência da rede (FONSECA; MOTA, 2017). Alguns desses problemas são abordados no presente trabalho.

Fonseca e Mota (2017) cita os problemas em aberto no gerenciamento de falhas em SDN, sendo:

1. Plano de Dados: recuperação de falhas, padronização de recursos de tolerância e capacidade adicional do plano de Dados;
2. Plano de Controle: plataformas de programação tolerantes a falhas e recuperação em ambientes geo-distribuídos;
3. Aplicações SDN: tolerância a falhas;
4. Integração entre as camadas: coordenação entre camadas e interoperabilidade; e
5. Outros cenários: integração com NFV, QoS e inovação.

Pode-se citar como trabalhos contemporâneos em gerenciamento de redes os projetos 5G-NORMA, METIS II, COGNET e SELFNET (LÓPEZ et al., 2017). Esse último é abordado detalhadamente na Subseção 2.3.2 devido a similaridade com a SONAr. Afirma-se ainda que todos esses trabalhos vão na direção de autogerenciamento, bem como a arquitetura SONAr. Assim sendo, a Subseção 2.3.1 traz a fundamentação teórica base para autogerenciamento com foco no fundamento de autocura.

2.3.1 Fundamentos de Computação Autônoma e Autocura

A IBM provavelmente apresentou a primeira definição de computação autônoma (GANEK; CORBI, 2003). Os quatro fundamentos de computação autônoma apresentados por Ganek e Corbi (2003) são: autoconfiguração (responsável por adicionar novos componentes, serviços ou software em tempo de execução; autocura (responsável por detectar,

diagnosticar e reagir a problemas); auto-otimização (responsável por alocar recursos para lidar com mudanças e crescimento); e autoproteção (responsável por antecipar ataques e resolvê-los).

Alguns exemplos de comportamento autônomo em hardware e software são: gerenciamento de dados (análise de dados históricos); *dashboard* para percepção de serviços lentos; previsão de informações; gerenciamento de sistemas (noção fim a fim de uma transação); e servidores construídos para se protegerem e se recuperarem de um estado anormal (GANEK; CORBI, 2003).

Autocura, foco do presente trabalho, é definida como um fundamento de computação autônoma e os conceitos apresentados em Ganek e Corbi (2003) são pertinentes até os dias atuais. Basicamente, um sistema que aplica autocura deve ser capaz de diagnosticar um componente com falhas e corrigi-lo. Ao longo dos anos, o diagnóstico e recuperação de falhas através de autocura foram apresentados por diferentes pesquisadores, podendo-se mencionar Psaier e Dustdar (2011), Vilchez, Yahia e Crespi (2014), Thorat et al. (2015), 3GPP (2018d).

A maioria dos pesquisadores utiliza os conceitos da IBM para criar estados, políticas e algoritmos para sistemas autônomos. No caso de autocura, o princípio básico é permitir que o sistema retorne de qualquer estado anormal para o estado normal. Os três estados básicos são '*Normal*', '*Degraded*' e '*Broken*' (PSAIER; DUSTDAR, 2011; GHOSH et al., 2007), conforme ilustrado na Figura 3. Observa-se que um sistema não passa do estado '*Normal*' para '*Broken*' diretamente. O estado '*Degraded*' é atingido primeiramente, pois as falhas/erros normalmente não ocorrem de imediato (há um tempo de degradação denominado *fuzzy zone*). Não há medição exata para a *fuzzy zone*, pois cada sistema se degrada de modo diferente (PSAIER; DUSTDAR, 2011; GHOSH et al., 2007).

Al-oqily et al. (2012) fazem uma análise completa sobre arquiteturas e algoritmos de autocura para sistemas de propósito geral. Na área de redes de computadores, é complexo prover e gerenciar serviços que atendam requisitos de usuários, visto que esses requisitos requerem alto número de serviços na rede. Para entregar esses serviços de rede, a abordagem de computação autônoma é recomendável. Especificamente em redes de computadores, a autocura deve corrigir defeitos que ocorrem quando: nós entram/saem da rede; links ficam congestionados; ou ocorrem mudanças de informações de roteamento (AL-OQILY et al., 2012).

Essencialmente espera-se que um sistema com propriedade de autocura seja capaz de monitorar e reconhecer anomalias, localizar a fonte de uma falha, responder a condições variáveis e executar mecanismos para levar o sistema de volta ao estado operacional *Normal*. Os algoritmos de autocura propriamente ditos, basicamente, executam três serviços, sendo eles: *backup* distribuído; detecção de falhas; e recuperação de nó (restauração de *backup* ou adaptação da rede para operar sem o nó) (AL-OQILY et al., 2012).

Em SDN também são exploradas as propriedades *self-** de computação autônoma,

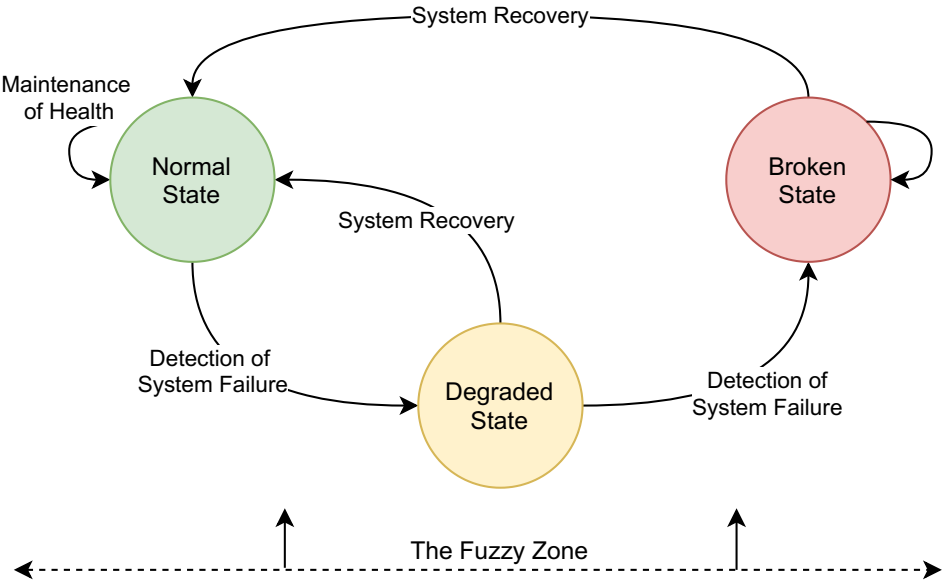


Figura 3 – Máquina de Estados de Autocura. Adaptado de Psaier e Dustdar (2011).

visto que, apesar das diversas vantagens, esse tipo de rede possui algumas questões em aberto, tais como: alto volume de tráfego; multimídia; segurança; tolerância a falhas; e padronização (COX et al., 2017). Ressalte-se que na literatura, fica nítido que os componentes responsáveis por tolerância a falhas focam no plano de Dados – convenientemente, controladores SDN amplamente utilizados, como o *Open Network Operating System* (ONOS) (ONOS, 2021) e *OpenDaylight* (ODL) (OpenDaylight, 2021), possuem funcionalidades para resolver falhas no plano de Dados. O presente trabalho tem, também, o intuito de resolver falhas nos planos de Controle e Gerenciamento. A Subseção 2.3.2 apresentará as propostas em direção a autocura para SDN, fazendo também uma breve discussão entre cada proposta e a SONAr.

2.3.2 Propostas visando Autogerenciamento para SDN

Algumas propostas que utilizam os conceitos apresentados na Subseção 2.3.1 são encontradas na literatura, o que mostra que autocura para SDN é um tema interessante do ponto de vista de investigação acadêmica. Fonseca e Mota (2017) mostram que tolerância a falhas e recuperação são problemas em aberto, considerando a falta de abordagens e padrões nessa área. Alguns trabalhos inclusive usam abordagens para autogerenciamento (WANG; YAN, 2016; CANINI et al., 2017; PADMA; YOGESH, 2015) em redes de computadores e outros até combinam esse conceito com SON (COX et al., 2017; RAMIREZ-PEREZ; RAMOS, 2016).

Tsagkaris et al. (2015) apresentam uma relação entre *Autonomic Network Management* (ANM) e SDN. Um sistema ANM baseia-se na computação autônoma proposta por

Ganek e Corbi (2003) e trabalha com o conceito de *Autonomic Control Loop* (ACL) para monitorar, analisar e executar procedimentos em uma rede. Tsagkaris et al. (2015) afirmam ainda que, apesar de certa similaridade, há um conflito entre ANM e SDN: aumentar a eficiência e confiabilidade da operação versus simplificar o gerenciamento e controle da rede. A abordagem ACL é uma das três usadas pela SONAr e será detalhada no Capítulo 4.

Outro trabalho relacionado a SDN e tolerância a falhas é o projeto LegoSDN (CHANDRASEKARAN; BENSON, 2014; CHANDRASEKARAN; TSCHAEN; BENSON, 2016), que usa uma técnica de isolamento de componentes com erros e basicamente opera na SBI. O isolamento de um controlador ou aplicação SDN com falhas permite que elas não cheguem ao plano de Dados. O LegoSDN propõe um novo *design* do controlador SDN para recuperar falhas, alegando que técnicas tradicionais (como *reboot* e *replay*) não podem ser aplicados em SDN diretamente. Chandrasekaran, Tschaen e Benson (2016) afirmam que os controladores são normalmente projetados de forma monolítica e falhas graves podem deixar o plano de Controle completamente inoperável.

O LegoSDN modifica o controlador SDN Floodlight e foca em dois tipos de falhas distintas: aplicações SDN que param inesperadamente; e aplicações SDN que instalam regras OpenFlow que causam algum erro (como *loop* ou *black hole*). As técnicas usadas para recuperar esses dois tipos de falhas são descritos em Chandrasekaran, Tschaen e Benson (2016). Essencialmente, o LegoSDN altera a sintaxe de primitivas OpenFlow sem alterar a semântica. Dessa forma, um *snapshot* é gerado anteriormente e, quando determinada primitiva causa problema, o *snapshot* é recuperado e a mensagem é reenviada de forma modificada (esperando-se que o formato modificado não cause o mesmo problema). Pode-se dizer que o LegoSDN difere da SONAr pois naquele é necessária modificação na arquitetura do controlador, enquanto na SONAr há independência de controlador. Apesar disso, pode-se dizer que os métodos de prevenção de falhas usados pelo LegoSDN servem de inspiração para técnicas na SONAr.

Computação autônoma também é encontrado em trabalhos relacionados com *Overlay Networks* (AL-OQILY et al., 2012) e *Wi-Fi* (DALAL, 2014), principalmente utilizando autocura. No caso de *Overlay Networks*, o problema reside em lidar com problemas que atrapalhem a QoS sem precisar de suporte da rede subjacente (*underlying*). No caso de redes *Wi-Fi*, o objetivo é mitigar falhas e corrigi-las automaticamente, uma vez que usuários domésticos (que utilizam *Wi-Fi*) não são especialistas no assunto (SANCHEZ et al., 2014).

Dalal (2014) propõe um *framework* para autocura em redes domésticas. Simplificadamente, o objetivo é que a rede seja capaz de reagir a patologias que danifiquem a *Quality of Experience* (QoE). O *framework* coleta informações na camada de Rede e algumas medições de dados são aplicadas para análise. As medições utilizadas envolvem ferramentas conhecidas, como *ping* ou largura de banda baseada nos dispositivos/aplicações rodando

em tempo real. Um exemplo é uma aplicação de *game*: caso o *framework* perceba perda de pacotes, ele tenta (autonomamente) alterar o tráfego para outro servidor (DALAL, 2014). Observa-se que redes *Wi-Fi* podem usar autocura ou outros fundamentos de SON, porém ressalva-se que há diferenças entre *Wi-Fi* e redes de telefonia (redes celulares). Gacanin e Ligata (2017) apresentam os desafios de computação autônoma na área de *Wi-Fi*.

Tratando-se de NFV, alguns trabalhos focam em criar *frameworks* para autocura com vistas a redes 5G. Sanchez et al. (2014) propõem um *framework* visando vulnerabilidades que podem ocorrer em ambientes SDN e NFV. Em SDN, as principais vulnerabilidades são escalabilidade, segurança e resiliência. Sanchez et al. (2014) focam em problemas de resiliência – principalmente do controlador – gerenciando autonomamente serviços, aplicações, controle (monitoramento dos links e *switches* controlados) e plano de Dados (estado dos *switches* e fluxos OpenFlow). Em NFV, o *framework* integra com o VNF *Manager* para propor migrações dinâmicas e criação de funções de rede (SANCHEZ et al., 2014). Ressalve-se que o *framework* trabalha de forma reativa (a detecção pró-ativa de vulnerabilidades é citada como trabalho futuro).

Ainda considerando SDN especificamente, Thorat et al. (2015) propõem um mecanismo de recuperação rápida utilizando autocura e um modelo analítico para calcular o tempo de recuperação de falhas. Nessa proposta, há módulos de recuperação de falhas no plano de Controle e de Dados. Basicamente, os módulos coletam informações periodicamente e calculam a melhor rota de backup em caso de anomalias. A recuperação muitas vezes se dá pelo *switch* e seus vizinhos para ocorrer o mais rápido possível (o controlador não é avisado).

O projeto SELFNET (NEVES et al., 2016) pode ser considerado o trabalho mais próximo do que está sendo proposto nesta tese. A ideia do projeto é uma arquitetura para SON, orquestração, administração e controle de acesso em SDN e NFV. Em essência, SDN, NFV, SON, *cloud computing* e *Artificial Intelligence* (AI) são integrados, criando-se assim um sistema de gerenciamento autônomo para a rede. O domínio de SDN e NFV é o foco do projeto, mas o mesmo pode ser compatível com sistemas não 3GPP futuramente.

No SELFNET, as questões autônomas são defrontadas através de subcamadas – *sublayers*. Basicamente a *Autonomic Management Sublayer* fornece capacidades de autocura, autoproteção e auto-otimização. Há também a definição da *Tactical Autonomic Language* (TAL), uma linguagem que especifica estratégias de resolução para fornecer ações autônomas. Na prática, sensores e atuadores são implantados em ambos os planos (Controle e Dados) e ficam responsáveis por ações de computação autônoma.

O foco da SONAr é em redes *core*, enquanto o SELFNET trabalha em redes 5G prioritariamente. Na SONAr, a camada de gerenciamento é única e todos os serviços (agrupados por entidades) são executados nesse plano, dissemelhante ao que ocorre no SELFNET. Apesar dessas divergências, pode-se dizer que há similaridade considerável

entre as duas abordagens. Dessa forma, uma futura integração entre SONAr e SELFNET não pode ser descartada.

Enquanto o SELFNET propõe claramente a implementação de características de *self-**, outros projetos possuem capacidade de autogerenciamento através de funções de rede, embora não mencionem claramente aspectos de *self-**. Destaca-se o COGNET (XU et al., 2016), que propõe a introdução de aprendizado de máquina para predição de demanda de recursos em NFV. O projeto prevê a coleta e análise de métricas de desempenho em NFV para prover melhor QoS a usuários finais. Apesar de NFV ser fortemente tratado pelo COGNET, a SDN não é mencionada, nem tampouco redes legadas.

Ainda no âmbito de NFV, pode-se destacar o projeto SUPERFLUIDITY (BIANCHI et al., 2016), que possui blocos de funções reutilizáveis similares a VNFs. O intuito do projeto é suportar auto-adaptação de redes SDN e NFV. Embora funções *self-** possam ser construídas nos blocos, não há menção a implementação dessas funções no projeto.

Conforme supramencionado, diversos trabalhos são encontrados abordando autocura para SDN e alguns para NFV. Os principais foram descritos brevemente nos parágrafos anteriores e utilizados como embasamento teórico para desenvolver aspectos de autocura na SONAr. A Tabela 1 compara as principais propostas. Ressalva-se que esses vários projetos estão sendo desenvolvidos paralelamente à SONAr, e nos próximos anos os resultados devem ser apresentados à comunidade. Para esta tese, é imprescindível despender alguns parágrafos para apontar a relação entre autocura e os diversos planos SDN/NFV. A Seção 2.4 descreve como um gerenciador, em uma rede SDN/NFV, deve saber distinguir, identificar e recuperar problemas dos planos de Dados, Controle e Gerenciamento.

Tabela 1 – Comparação dos Principais Projetos Correlatos.

Projeto	**SDN**	**NFV**	**Funções de Cura**	**Funções de Autocura**
LegoSDN (CHANDRASEKARAN; TSCHAEN; BENSON, 2016)	Sim	Não	Sim	Não
SELFNET (NEVES et al., 2016)	Sim	Sim	Sim	Parcial
COGNET (XU et al., 2016)	Não	Sim	Sim	Parcial
SUPERFLUIDITY (BIANCHI et al., 2016)	Não	Sim	Parcial	Parcial
SONAr / NOSH (Capítulos 3 e 4)	Sim	Sim	Sim	Sim

2.4 Autocura em Redes de Computadores e Móveis

Um sistema *standalone* é capaz de efetuar autocura de maneira menos complexa do que um sistema distribuído, uma vez que os componentes do sistema fazem parte de uma única aplicação, independentemente se esse sistema é modular ou monolítico. O sistema *standalone* normalmente reconhece uma falha e se recupera para voltar ao estado 'Normal'. Em sistemas distribuídos, como é o caso de redes de computadores e redes móveis, introduzir capacidade de autocura não é trivial, visto que elementos que compõem o sistema não estão centralizados em uma única aplicação e, por vezes, nem mesmo em um único servidor.

Uma rede de computador é um grafo (topologia de rede), com m nós (NEs), conectados entre si por n arestas (enlaces). Equipamentos diversos (servidores, computadores pessoais, equipamentos móveis, entre outros) também podem fazer parte do grafo como nós, se se observar a topologia completa (THORAT et al., 2015). Diversos problemas podem deixar a rede não saudável, sendo os mais comuns a abertura de determinados vértices do grafo devido a falhas nos links; outros problemas comuns são mau funcionamento de hardware/software, *bugs* em aplicações, sobrecarga da rede ou dos equipamentos nela contidos (FONSECA; MOTA, 2017). Neste trabalho, como já mencionado previamente, a rede também está não saudável caso requisitos de QoS não estejam sendo atendidos.

O grafo citado acima possui uma extensão quando se trata das tecnologias SDN e NFV. Com a introdução da separação lógica do plano de Controle, novos elementos surgem para gerenciamento da rede, como por exemplo o Controlador SDN, componentes de gerenciamento NFV (VIM, VNFM etc) e orquestradores de infraestrutura. Esses novos componentes são novos nós na topologia de rede e novos links para esses nós devem também ser considerados. Depreendendo, uma rede de computador ou rede móvel é um sistema constituído de várias partes (nós e arestas do grafo) distribuídas pelo ambiente. Quando se fala de inserir a capacidade de autocura nesse sistema, deve-se considerar os planos de Dados, Controle e Gerenciamento, com as particularidades de cada um. Isto é, quais falhas podem ocorrer em cada plano, como monitorar cada plano, além de como recuperar cada plano das falhas.

2.4.1 Autocura do Plano de Dados

O plano de Dados é a camada de infraestrutura física ou virtual na qual NEs, servidores ou *appliances* estão implantados. Nas redes tradicionais, como as primitivas de controle e dados são tratadas pelos mesmos NEs, o zelo necessário para manter a rede saudável consiste em monitorar os links/servidores e mantê-los em perfeito funcionamento. Quando um link está sobrecarregado, *down*, bloqueado ou rompido, os protocolos de rede, como o OSPF, reconfiguram as rotas para que os NEs evitem encaminhar tráfego para aqueles links. Pode-se afirmar que a cura de um problema através de roteamento é uma

reconfiguração de recursos. Tratando-se particularmente dos NEs, o cuidado de sistemas de gerenciamento está na análise de capacidade constante dos NEs, ou seja, avaliação da quantidade de memória utilizada, poder de processamento e *storage*.

A tecnologia SDN, apesar de aludir uma simplificação da operação de rede, traz alguns agravantes que não podem ser desprezados no contexto aqui explanado. Como o controle é totalmente apartado dos NEs que encaminham tráfego, protocolos como OSPF não resolvem problemas de topologia não saudável através de roteamento. De fato, atentando à arquitetura descrita na Subseção 2.1.1, o roteamento ou qualquer outra função de rede deve ser desenvolvida através de uma Aplicação SDN que vai rodar acima da NBI. Essa flexibilidade é uma vantagem de SDN, porém a falta de padronização e a não aceleração no desenvolvimento de aplicações SDN, conforme afirmado por Rehman, Aguiar e Barraca (2019), colocam a tecnologia em um estado imaturo.

Se por um lado a tecnologia SDN carece de aplicações para tolerância a falhas, por outro, um certo avanço na reconfiguração e recuperação de falhas foi realizado por algumas plataformas específicas. Pode-se afirmar que, convenientemente, controladores SDN como o ONOS (ONOS, 2021) e ODL (OpenDaylight, 2021) possuem capacidade de reconfiguração em caso de problemas no plano de Dados. A recuperação trata do roteamento aplicado pelas aplicações padrões dos controladores para evitar determinado link não saudável.

Na tecnologia NFV, a autocura trata da manutenção da saúde dos elementos virtuais que compõem a arquitetura. Essencialmente, os recursos computacionais (memória e processador), rede e *storage* devem ser monitorados e a realocação de recursos deve ser realizada para evitar falhas e para escalar o ambiente – *scale-in/out*. Considerando o plano de Dados, pode-se afirmar que algumas plataformas que integram arquiteturas NFV amplamente utilizadas possuem capacidades intrínsecas de realocação de recursos. Como exemplo, pode-se citar o Openstack (2021), um VIM que consegue realocar recursos computacionais em tempo de execução. O Kubernetes (2021), outro VIM amplamente utilizado, possui inclusive a funcionalidade (denominada na própria documentação) de autocura: se um *container* ou *pod*[1] apresentam falhas, a própria plataforma é capaz de recriar o *container/pod* em algum *working node*.

Os problemas de infraestrutura clássicos são relativamente previstos em SDN e NFV. Porém, problemas do ponto de vista de entrega de QoS não possuem tratativa prognosticada. No *Network Slicing* por exemplo, o gerenciamento da saúde da rede se dá pela manutenção dos NSs durante todo o seu ciclo de vida. O ciclo de vida de um NS possui as seguintes fases: (i) preparação; (ii) aprovisionamento; (iii) execução; e (iv) descomissionamento. Segundo a padronização vigente de *Network Slicing*, 3GPP (2018e), espera-se que os *frameworks* de OAM possuam capacidade de autocura para manter os NSs saudáveis durante todas as fases.

Na literatura, alguns *frameworks* de orquestração de redes Fim-a-Fim – *End-to-End*

[1] No Kubernetes, POD é uma unidade mínima de *deployment*, constituída de um ou mais *containers*.

(E2E) – são encontrados para aprovisionamento de NSs baseando-se nos recursos de infraestrutura disponíveis no ambiente – geralmente os requisitos de QoS são considerados no aprovisionamento. Todavia, em ambientes de produção, os problemas que podem levar a falhas na entrega de QoS devem ser considerados, como as inconsistências (*loops* e *black holes*), falhas em hardware/software, links congestionados e sobrecarga da rede. Na terceira fase do ciclo de vida de um NS, tempo de execução, *frameworks* OAM precisam reconfigurar recursos a todo instante para mitigar ou evitar falhas decorrentes dos problemas apontados (FONSECA; MOTA, 2017; Chen et al., 2018).

Nesta tese, a funcionalidade de autocura foca em manter os NSs saudáveis, isto é, o *framework* NOSH interage com orquestradores de redes E2E para obter informações acerca dos NSs e seus respectivos requisitos de QoS. Até onde as investigações realizadas para esta tese chegaram, há uma carência de *frameworks* OAM com esse nível de abrangência. As demais falhas no plano de Dados ficam a cargo de orquestradores e controladores NFV/SDN que operam no ambiente.

2.4.2 Autocura do Plano de Controle

Se por um lado problemas ocasionados por falhas no plano de Dados são bem abrangidos pela literatura, problemas do plano de Controle carecem de soluções efetivas. Primeiramente, a introdução do controlador SDN traz consigo algumas desafios para gerenciamento. O primeiro desafio está relacionado em manter o próprio controlador no estado 'Normal'. Ressalta-se que, se o controlador estiver indisponível por qualquer motivo, toda a rede estará inoperável – nesse caso, nem sequer as funções de rede implementadas nas Aplicações SDN conseguem aplicar suas ações. Outro desafio está na integração entre o plano de Dados e o plano de Controle: os NEs do plano de Dados precisam de comunicação ininterrupta com o controlador SDN, para que primitivas de controle sejam transmitidas corretamente pela SBI (Rehman; Aguiar; Barraca, 2019).

A Figura 4 mostra um controlador SDN – *SDN Controller* (SDNC) – controlando o plano de Dados – *Data plane* – composto de oito NEs. A comunicação entre o $SDNC_1$ e o NE_7 pode ser estabelecida através dos caminhos $SDNC_1$–NE_2–NE_6–NE_7, $SDNC_1$–NE_2–NE_6–NE_3–NE_7 ou $SDNC_1$–NE_2–NE_6–NE_3–NE_5–NE_8–NE_7. Caso as regras de encaminhamento nos NEs estejam configuradas para que a rota de NE_7 até $SDNC_1$ seja pelo caminho $SDNC_1$–NE_2–NE_6–NE_7, e caso ainda o link NE_6–NE_7 esteja congestionado ou rompido, primitivas de controle não chegarão em NE_7. Ainda na topologia da figura, se o link $SDNC_1$–NE_2 apresentar problemas, todo o domínio estará inoperável. Dessa forma, o emprego de autocura deve pressupor ações de tolerância a falhas, reconfigurando rotas de controle.

O $SDNC_1$ e suas aplicações – quais sejam aplicações SDN rodando na NBI – também devem ter aspectos de autocura para se recuperar de falhas de software e hardware que porventura aconteçam. Normalmente, essas falhas acontecem por causa de *crashes* do

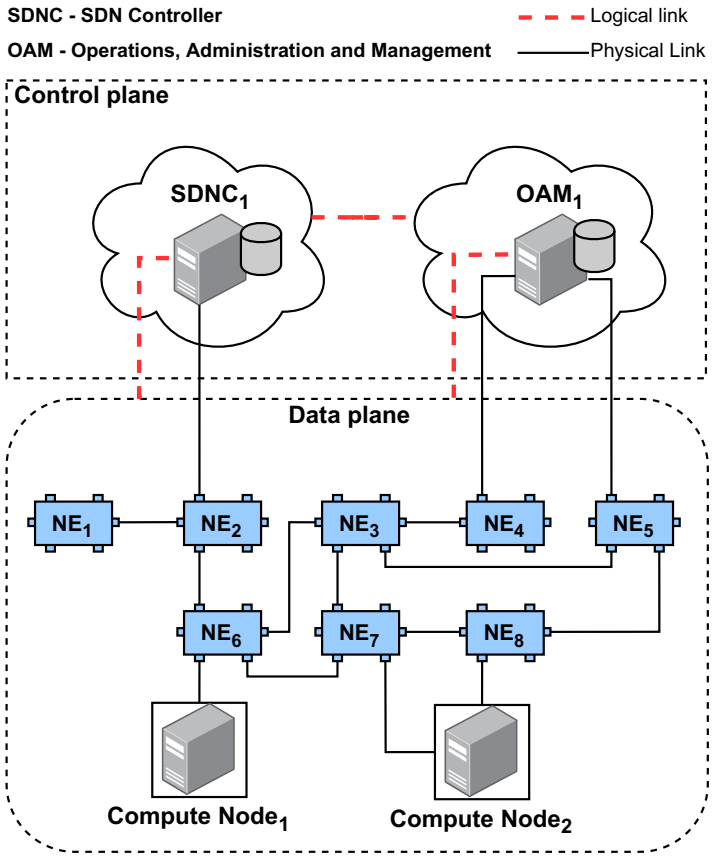


Figura 4 – Visão da Interoperabilidade Entre Planos de Dados e Controle.

sistema, *bugs* de software e também sobrecarga dos servidores nos quais o sistema executa (COX et al., 2017). No caso de servidores físicos, há uma limitação de recursos da infraestrutura. Porém, recomenda-se a utilização de servidores virtuais ou *containers* que possam ser autonomamente escalados para suportar aumento de carga. Ademais, as máquinas virtuais ou *containers* podem também ser recriados em servidores físicos que estejam em outra parte da infraestrutura.

2.4.3 Autocura do Plano de Gerenciamento

Na Figura 4, o OAM_1 representa qualquer elemento de gerência tradicional ou componentes de gerenciamento NFV. Por exemplo, o OAM_1 poderia estar representando um controlador VIM que orquestra recursos computacionais de *Compute Node*$_1$ e *Compute Node*$_2$. Notoriamente, OAM_1 possui conexões lógicas com o plano de Dados para comunicação com os nós computacionais. Contudo, na topologia representada, OAM_1 possui conexões físicas apenas com NE_4 e NE_5. Na eventual queda de NE_4 e NE_5 ou problemas entre os links OAM_1–NE_4 e OAM_1–NE_5, o controlador VIM não será capaz de operar recursos, pois não conseguirá enviar primitivas de gerenciamento para *Compute Node*$_1$ e *Compute Node*$_2$. Qualquer outro tipo de comunicação entre controladores/orquestradores

NFV e nós computacionais possui o mesmo nexo: um VNFM requer comunicação ininterrupta com suas VNFs gerenciadas, enquanto qualquer VIM requer comunicação ininterrupta com a NFVI.

Assim como o SDNC, *frameworks* OAM podem apresentar problemas computacionais de hardware e software. Orquestradores/controladores NFV comumente possuem capacidade de autocura de recursos gerenciados. Por exemplo, um VIM é capaz de reconfigurar recursos para que uma máquina virtual sob seu gerenciamento seja recuperada de uma queda. No entanto, não se encontram especificações de procedimentos de autocura para o próprio OAM. Como será explanado no Capítulo 3, diversos componentes e aplicações constituem a SONAr; é fundamental um mecanismo de autocura para esses componentes e aplicações.

Além do foco em autocura, para que NSs entreguem os níveis de QoS (requeridos na instanciação) durante o tempo de execução, esta tese foca em mecanismos para que os planos de Controle e Gerenciamento funcionem ininterruptamente. Para isso, elaborou-se um *framework* OAM baseado na arquitetura SONAr que utiliza fortes conceitos de NFV. No Capítulo 3 há o detalhamento da SONAr e seus principais serviços, enquanto o Capítulo 4 aprofunda no *framework* NOSH.

Capítulo 3

Uma Arquitetura para Redes Auto-organizáveis

No âmbito deste trabalho, criou-se um modelo de referência para autogerenciamento de redes de computadores baseando-se nos conceitos explanados no Capítulo 2. A partir deste modelo, projetou-se uma arquitetura auto-organizável denominada Arquitetura para Redes Auto-organizáveis – *Self-organising Networks Architecture* (SONAr) –, que será reportada neste capítulo. A SONAr pode ser descrita como uma solução para os problemas de gerenciamento em redes de computadores, telecomunicações e telefonia. Trata-se de uma arquitetura que congrega computação autônoma (desenvolvida na engenharia de software) e SON (desenvolvida nas telecomunicações). A ideia fundamental é permitir que redes de computadores utilizem SDN e NFV para permitir serviços autogerenciáveis.

Basicamente, o ambiente no qual a SONAr define um domínio pode ser dividido em três camadas, sendo elas: *Infrastructure Layer*; *Control Layer*; e *Application Layer*. A separação entre essas camadas pode ser considerada lógica ou abstrata. A SONAr propriamente dita faz parte do *Management Plane*, porém, o gerenciamento e a integração com as três camadas também faz parte do propósito da SONAr, conforme mostra a visão da Figura 5.

A *Client Layer*, na qual se encontram dispositivos diversos de usuários finais, como computadores e *smartphones*, na qual equipamentos de usuários e aplicações (*End-User*) são executadas, está fora do escopo desta tese. Portanto, a SONAr não é responsável pelo autogerenciamento de equipamentos dessa camada pois não é parte do domínio de Gerenciamento e Controle de uma rede de computador. Entendemos que aplicações e dispositivos do usuário final estão relacionados ao plano de Dados.

A *Infrasctructure Layer*, por sua vez, contém elementos da infraestrutura de rede (que podem ser físicos ou virtuais), interconectados por enlaces para compor a topologia da rede. Exemplos de elementos da *Infrastrucutre Layer* são *switches*, *switches* OpenFlow, NFVI, roteadores, VIM etc. Vale a pena ressaltar que esses elementos de rede podem ser físicos ou virtuais. Esses equipamentos são operados por controladores e orquestradores

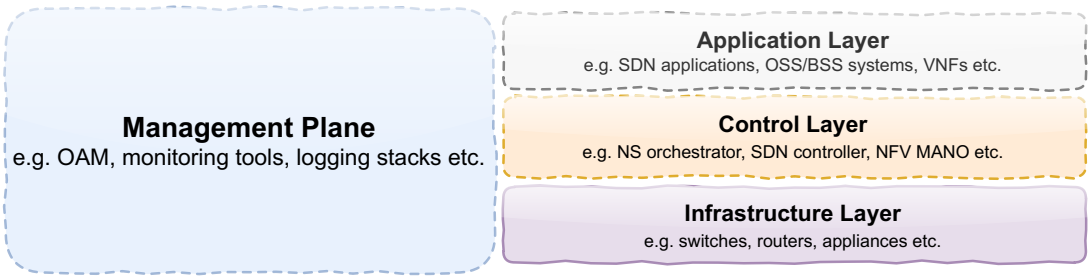


Figura 5 – Visão de Gerenciamento de Redes em Camadas.

SDN e NFV e, consequentemente, autogerenciados pela SONAr.

A *Control Layer* abriga todos os componentes de controle de uma rede. Tipicamente, esses componentes controlam *switches* SDN, através de protocolos como o OpenFlow, bem como elementos de rede legada, através de protocolos/tecnologias como SSH, telnet e *Simple Network Management Protocol* (SNMP). Componentes de NFV, com propósitos de controle e orquestração, também estão incluídos na *Control Layer*, como por exemplo o *Management and Network Orchestration* (MANO).

O *Management Plane* pode ser descrito como o plano central da SONAr, ou seja, o plano no qual os componentes da arquitetura estão estruturados e executam os serviços de autogerenciamento. Nota-se que a SONAr pode coexistir com outras arquiteturas e protocolos de gerenciamento, por exemplo o *Network Configuration Protocol* (NETCONF). O *Management Plane* agrupa todos os componentes de gerenciamento e possui integração nítida com as camadas de aplicação e controle, onde se encontram as ferramentas para administração da rede e as aplicações de funções de rede, por exemplo aplicações SDN que operam na NBI e sistemas de controle que operam na SBI.

Conforme mencionado, as camadas em torno da SONAr possuem diversos componentes que eventualmente se integram com a arquitetura. O foco deste capítulo é apresentar detalhadamente a SONAr. Dessa forma, será explorado o *Management Plane* e a Seção 3.1 define os componentes principais da SONAr.

3.1 Visão Geral da Arquitetura

A Figura 6 apresenta a visão arquitetural propriamente dita da SONAr, sendo que as entidades da arquitetura estão agrupadas em quatro grandes blocos, quais sejam: *Self-organising Entities* (SOEs), *Collector Entities* (CoEs), *Self-learning Entities* (SLEs) e *Integration Entities* (IEs). Fora desses quatro blocos, encontram-se o *Network Event Manager* (NEM) e o *Network Database* (NDB), que são estruturas compartilhadas por todos os blocos. Ainda, destacam-se os *Local Agents* (LAs) e *Control Primitives Interceptor* (CPI), que executam nas demais camadas para coletas de métricas de desempenho

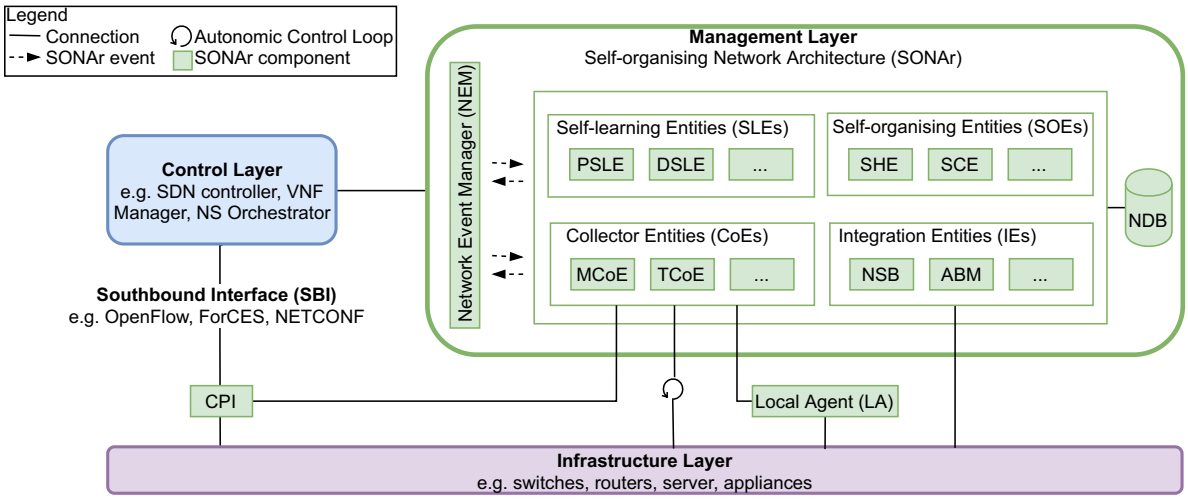


Figura 6 – Visão Geral dos Componentes da SONAr.

específicas. Entende-se que a Figura 6 é uma visão geral da SONAr, apresentando em um único desenho todos os componentes envolvidos na arquitetura. A estrutura da *Managemenet Layer* são, de fato, todos os itens da SONAr. Todos os componentes serão detalhadas adiante neste capítulo.

Os elementos apresentados na Figura 6 mostram a arquitetura de forma abstrata. Ressalta-se que há integração entre diversos desses elementos, usando o modelo *event-driven*. A SONAr foi criada utilizando-se fortemente os conceitos de microserviços e as ideias do Manifesto Reativo (BONÉR et al., 2014). O NEM é parte fundamental da plataforma, pois garante a comunicação de todos os componentes por meio de eventos.

Os eventos na SONAr podem ser criados ou recebidos por todos os componentes internos ou por componentes de outras camadas (por exemplo: um *switch* pode enviar um evento para o Controlador SDN), utilizando desta forma o modelo *Publisher/Subscriber*. Como exemplo, pode-se mencionar uma métrica de um *switch* OpenFlow. A *Metrics Collector Entity* (MCoE) pode coletar determinada métrica de uma porta no *switch* e publicá-la em forma de evento no NEM. Vários componentes, que estão subscritos para receber esse tipo de evento, vão recebê-lo e tratá-lo. Exemplo: esse tipo de métrica pode ser analisada tanto pela *Self-healing Entity* (SHE) quanto pela *Self-optimisation Entity* (SOPE) (essas entidades serão detalhadas na Seção 3.2).

Os componentes mostrados na Figura 6 são chamados de Entidades (*Entity*) pois são elementos que comportam diversos protocolos. Como exemplo pode-se citar a SHE (foco da presente tese), que engloba diversos serviços para permitir que os fundamentos de autocura sejam aplicados (como serviços de monitoramento, recuperação e análise de dados).

Os componentes da SONAr são desenvolvidos por meio do conceito de microserviços.

Isso significa que, na prática, cada componente desenhado na Figura 6 pode ser explodido em diversos microserviços. A comunicação entre esses vários microserviços e os componentes SONAr é feita através do NEM. Cada componente mostrado será descrito na Seção 3.2.

A integração entre os componentes da SONAr, utilizados para autocura, e a especificação da SHE será descrita minuciosamente no Capítulo 4. A forma concreta de se implementar a arquitetura (considerando suas características *event-driven* e os conceitos de microserviços) será detalhada na Seção 3.3.

3.2 Serviços para Autogerenciamento

Esta subseção faz um resumo dos componentes da SONAr representados na Figura 6. Trata-se de uma descrição breve dos serviços utilizados nesses componentes para autogerenciamento da rede. O Capítulo 4 vai aprofundar nos serviços relacionados à autocura, mas uma visão geral de toda a SONAr se faz necessária, pois eventualmente há integração entre os serviços de autocura e os demais.

3.2.1 *Self-Organising Entities*

As *Self-organising Entities* (SOEs) são responsáveis por aplicar os algoritmos de computação autônoma propriamente ditos. Pode-se considerar que são o núcleo principal da SONAr. Considera-se ainda que os algoritmos executados nessas entidades são *real-time*. Normalmente, as SOEs estão subscritas em eventos específicos de gerenciamento no NEM. Um evento é recebido, tratado imediatamente e novos eventos são gerados e publicados novamente no NEM. *Self-organising Entities* (SOEs) é composta por seis entidades principais, sendo:

1. *Self-healing Entity* (SHE): entidade responsável por aplicar algoritmos para predição ou detecção de falhas/erros, além de algoritmos para recuperação. A SHE deve detectar falhas na *Control Layer*, falhas na *Infrasctructure Layer* (plano de Dados), além de falhas nos próprios elementos da SONAr (*Management Plane*). Considerando a complexidade de ambientes de SDN e NFV, a SHE pode ser considerada como uma das principais entidades da SONAr, uma vez que é crucial manter a operação e o gerenciamento da rede. A SHE é o foco principal da presente tese e será detalhadamente descrita no Capítulo 4;

2. *Self-configuration Entity* (SCE): entidade que roda os serviços necessários para *bootstrapping* da rede e inserção/remoção de elementos de rede – *plug-in-play*. No *bootstrapping*, a SCE configura os fluxos necessários nos NEs para que a rede saia do estado inoperante para o estado 'em operação'. Nas infraestruturas atuais[1], um

[1] Informações coletadas em entrevistas com administradores de rede de uma MNO.

operador de rede precisa inserir manualmente um novo elemento de rede (exemplo: *switch*) na planta (criando toda a parte elétrica e de cabeamento de rede) e logo após configurar manualmente endereço IP, *hostname*, rotas, apontar o IP do controlador SDN (em caso de ambientes SDN) etc. O propósito da SCE é descobrir e aplicar essas configurações automaticamente. Além de automatizar o processo (poupando tempo e erros humanos), a SCE também garante a segurança da rede. Atualmente, um *switch* pode ser inserido na rede com propósitos maliciosos. Com a SCE, pode-se garantir que apenas *switches* certificados podem entrar em determinada rede;

3. *Self-optimisation Entity* (SOPE): entidade responsável pela execução de otimizações no ambiente. Basicamente, os serviços rodando na SOPE subscrevem-se a eventos contendo estatísticas e métricas da rede, sendo que a análise dessas métricas pode resultar em melhorias a serem aplicadas para otimização da rede. Exemplos de otimizações comuns em ambientes SDN/NFV são: reconfiguração de fluxos OpenFlow com o intuito de balancear melhor o tráfego na topologia; gerenciamento de NSs; e utilização de recursos computacionais de forma otimizada. Um outro exemplo de otimização da rede é evitar congestionamentos: a SOPE pode identificar que determinada porta ou *switch* vai ficar congestionada no futuro. Dessa forma, algoritmos de roteamento podem ser executados para alterar determinadas rotas e evitar esse congestionamento. Ressalta-se que a SOPE aplica algoritmos em tempo real, sendo que uma análise mais profunda das métricas não é responsabilidade dessa entidade (essa funcionalidade está descrita nas SLEs mais a frente neste texto);

4. *Self-protection Entity* (SPE): entidade que garante a segurança de todas as camadas da SONAr representadas na Figura 6. Eventos recebidos pela SPE são analisados e, caso seja identificado algum ataque ou evento malicioso, os serviços dessa entidade agem para isolar o(s) componente(s) infectado(s). Os serviços de isolamento são cruciais para garantir que não haja propagação de determinado ataque pelo restante da rede. Os componentes da *Control Layer* e do *Management Plane* recebem especial atenção da SPE. Isso se deve à importância dos componentes que rodam nessas camadas. Como exemplo, pode-se citar o controlador SDN. Um atacante que consiga acesso ao controlador passa a ter controle de toda a rede. Dessa forma, a SPE precisa garantir que esse elemento esteja seguro contra ataques a todo momento. Os elementos da própria SONAr devem ter essa mesma atenção;

5. *Self-planning Entity* (SPLE): esta entidade trabalha em conjunto com as demais SOEs para executar tarefas agendadas ou periódicas. Como exemplo pode-se citar determinada otimização descoberta pela SOPE. Essa otimização pode requerer ações semanais em determinado elemento de rede. A SOPE executa a ação, porém o agendamento e o gatilho para execução da ação são feitos pela SPLE. Portanto, ta-

refas repetitivas são controladas pela SPLE, além de ações que devam ser executadas no futuro;

6. *Self-management Entity* (SME): essa entidade é responsável por gerenciar o funcionamento das SOEs. Destaca-se que as SHE, SCE, SOPE, SPE e SPLE não podem executar seus serviços de forma isolada, pois inconsistências podem acontecer. Por exemplo, a SOPE pode tomar a decisão de passar determinado fluxo por uma porta (de determinado *switch*), que foi anteriormente bloqueada pela SPE (por questões de segurança). Nesse exemplo, a decisão da SOE vai sobrescrever a decisão prévia da SPE. A SME é responsável por organizar, enfileirar e priorizar decisões das demais SOEs. O conceito de transação também é implementado pela SME, ou seja, o *rollback* de determinada ação deve ser executado por essa entidade. Um exemplo de *rollback* é a configuração incompleta de determinada rota em uma rede SDN: caso a SCE ou qualquer outra entidade determine a configuração de uma rota em uma topologia de *n switches* OpenFlow, a SME precisa garantir que as configurações necessárias sejam executadas nos *n switches*. Caso contrário, deve-se efetuar a volta/retirada (*rollback*) das configurações que foram corretamente aplicadas.

3.2.2 *Collector Entities*

As *Collector Entities* (CoEs) são responsáveis por coletar informações da infraestrutura e dos componentes de controle e gerenciamento da própria SONAr. As informações coletadas são usadas para monitorar e avaliar, o estado e desempenho da rede, e fornecer garantia de QoS. Essas entidades são fundamentais para os serviços de autocura. Todo elemento de qualquer camada exibida na Figura 5 possui informações a serem coletadas, podendo ser estatísticas, métricas computacionais (processamento, memória e disco), *logs* de aplicações, informações de mudanças de fluxos/topologias, entre outras. As entidades CoEs são divididas em cinco componentes principais, sendo:

1. *Metrics Collector Entity* (MCoE): entidade responsável por coletar métricas de dispositivos de rede, tais como *links*, servidores, *containers* etc. Como exemplo pode-se citar informações de capacidade de portas de *switches*/roteadores, capacidade de servidores (como % de processador utilizado) ou informações de desempenho de aplicações. As métricas coletadas pela MCoE são indispensáveis para o funcionamento da SHE, pois a identificação de problemas nas camadas da SONAr é, por vezes, realizada através da análise dessas métricas;

2. *Topology Collector Entity* (TCoE): entidade responsável por aplicar algoritmos de descoberta de serviço/protocolo, monitorar mudanças na(s) topologia(s) de rede e monitorar alterações na topologia de controle. Um exemplo claro é o grafo de rede de determinada topologia: caso um novo *link* seja estabelecido manualmente por um

operador de rede, a TCoE deve receber este evento e realizar as alterações necessárias nas estruturas de dados armazenadas na SONAr, além de publicar eventos que sejam pertinentes caso outras entidades precisem saber da existência do novo *link*. Um exemplo é a SOPE: um novo *link* significa a possibilidade de otimizar determina rota; para a SHE, um evento de *link-down* é fulcral;

3. *Alarms Collector Entity* (ACoE): entidade responsável por coletar ou reagir a notificação de eventos estratégicos, como dados obtidos pelo SNMP ou limiares pré-configurados atingidos. Os alarmes detectados geram novos eventos que podem ser enviados ao *Network Administration Dashboard* (NAD) ou *Network Service Broker* (NSB) (descritos adiante neste capítulo) e relatados (automaticamente ou manualmente) a operadores/administradores de rede;

4. *Samples Collector Entity* (SCoE): entidade responsável por efetuar *Deep Packet Inspection* (DPI), *port mirroring* e *sniffing* na rede para inferir tipos de conteúdos de determinado tipo de comunicação para detectar anomalias na rede. Ao detectar uma anomalia, a SCoE lança um novo evento e as entidades responsáveis (SHE ou SPE por exemplo) podem tratá-lo. Um exemplo de anomalia é uma determinada comunicação de chamada telefônica: caso a latência necessária para que o áudio não sofra interrupções não seja respeitada, a SCoE pode identificar essa ocorrência;

5. *Logs Collector Entity* (LCoE): componente responsável por coletar qualquer tipo de *log*. Os elementos de rede, servidores na *Control Layer* (por exemplo aplicações SDN), entidades da própria SONAr, entre outras, geram *logs*. A LCoE coleta *logs*, enviando-os para estruturas centralizadas (banco de dados de pilhas de *logging*). Dessa forma, padrões de *logs* podem ser executados e eventos gerados. Um exemplo é algum serviço da SHE: caso haja um problema na própria aplicação SHE, um *log* de nível '*ERROR*' pode ser lançado, indicando uma anormalidade na aplicação.

As CoEs normalmente executam algoritmos periodicamente (*state driven*) para coletar dados de elementos de rede e demais componentes da SONAr. Isso significa que as entidades requisitam (tomam a ação) – a elementos de rede – métricas, *logs*, alterações de topologia etc. Essa estratégia, denominada ACL, pode ser insuficiente para autocura pois, dentre outros problemas, a rota de controle do servidor rodando SONAr até um NE pode estar indisponível. Dessa forma, Agentes Locais – *Local Agent*s (LAs) – inseridos em elementos de rede para coleta e envio de dados, além de interceptadores – *Control Primitives Interceptor*s (CPIs) – na SBI devem ser aplicados em conjunto com a estratégia de *control loop*. CPI e LAs serão detalhados ainda neste capítulo.

3.2.3 *Self-Learning Entities*

As *Self-learning Entities* (SLEs) fornecem estruturas para aprendizado de máquina e outros mecanismos, baseados em Inteligência Artificial – *Artificial Intelligence* (AI), para geração de conhecimento através de informações de rede. Esse conhecimento é utilizado pela SONAr para predição, *tuning* de desempenho (otimização), tradução de intenções, e assim por diante. Como exemplo, pode-se citar a função de predição para autocura. Imagine que periodicamente determinado *link* fique sobrecarregado (exemplo: o *link* que fornece um NS para *streaming* de um jogo de futebol pode ficar sobrecarregado todo domingo a tarde, quando há futebol ao vivo sendo transmitido). Algoritmos de AI rodando nas SLEs podem identificar esse padrão e gerar um novo evento (que será recebido pela SHE). A SHE pode executar, com o auxílio da SPLE, um balanceamento de tráfego todo domingo a tarde, de forma que o *link* em questão não seja sobrecarregado. Dessa forma, a predição de determinada ocorrência pode evitar que a rede entre em um estado não saudável.

Apesar da similaridade entre a as SOEs e as SLEs, essas últimas são separadas para não atrapalhar o desempenho daquelas. Algoritmos (de aprendizado de máquina principalmente) podem gastar horas consideráveis e alto processamento. Portanto a separação em entidades específicas é aconselhável. As SLEs são resumidamente descritas a seguir:

1. *Prediction Self-learning Entity* (PSLE): entidade responsável por aplicar algoritmos para classificar métricas, serviços, tempo (desempenho) e topologia, fornecendo uma representação discreta do cenário;

2. *Tuning Self-learning Entity* (TSLE): entidade responsável por fornecer otimização para todos os demais componentes da SONAr. A TSLE monitora a qualidade dos cenários, avalia métricas e níveis de serviço a fim de ajustar parâmetros para configuração de recursos em tempo de execução;

3. *Diagnostic Self-learning Entity* (DSLE): entidade responsável por implementar algoritmos para correlacionar causas raízes de problemas, garantia de serviço, detecção e diagnóstico de degradação na saúde ou segurança da rede. Essas ações são feitas comparando-se as métricas e limites pré-estabelecidos (pelos operadores de rede) ou pela análise da base de conhecimento gerado ao longo do funcionamento da rede;

4. *Rating Self-learning Entity* (RSLE): entidade que avalia os dispositivos de rede, *links*, serviços e aplicações, avaliando desempenho e histórico de cada um deles. Com a base histórica de conhecimento, a RSLE pode definir determinada estratégia de configuração para ser usada por outras entidades da SONAr;

5. *Natural Language Processing Self-learning Entity* (NSLE): entidade que implementa um processador de linguagem natural para inferir políticas, funções e filtros de

determinado elemento de rede. As intenções de comunicação das aplicações são inferidas através dessa entidade e são disponibilizadas para as demais entidades (como a SHE) para garantir o funcionamento correto de cada aplicação;

6. *Behavior Self-learning Entity* (BSLE): entidade que detecta comportamentos corriqueiros na rede. Os comportamentos são definidos através da identificação de padrões grupais que podem afetar a operação da rede de algum modo. Por exemplo, um determinado comportamento de uma aplicação SDN que causa inconsistência pode ser identificado como causa de um problema em consequência da ação de outra aplicação.

No presente momento, as SLEs estão em um estágio de especificação e podem receber novas entidades conforme novos pesquisadores atuem nesse campo na SONAr. Ressalta-se que os dados utilizados para predição podem ser coletados dos NEs rodando na Camada de Infraestrutura, controladores ou orquestradores rodando na Camada de Controle, bem como de aplicações SDN e VNFs rodando na Camada de Aplicações.

3.2.4 Componentes Auxiliares

É válido afirmar que as SOEs, SLEs e CoEs formam os componentes responsáveis pelo autogerenciamento e auto-organização da rede, em consequência de conterem os serviços essenciais para monitoramento e tomada de decisões. Além desses componentes, alguns outros podem ser notados na Figura 6, quais sejam: NEM, NDB, CPI, LAs e IEs. Esses últimos estão agrupados como entidades pelo fato de conterem serviços que publicam e recebem eventos, similarmente às SOEs, SLEs e CoEs. Os demais não se conectam ao NEM, e portanto são componentes auxiliares. Todos eles serão descritos nesta subseção.

3.2.4.1 *Control Primitives Interceptor*

O *Control Primitives Interceptor* (CPI) é um componente crucial para a SONAr e seu propósito é se estabelecer como um *proxy* entre o plano de Dados/Infraestrutura e o Controlador SDN, estando inserido na SBI. As primitivas que trafegam na SBI passam pelo CPI e são analisadas. Caso possuam semântica de gerenciamento, o CPI publica um evento no NEM com as primitivas. Caso contrário, elas são apenas enviadas ao controlador. Primitivas OpenFlow contendo métricas de determinado *switch* são exemplos de mensagens com semântica de gerenciamento. Conforme mostrado na Figura 6, o CPI fica precisamente localizado entre as camadas de infraestrutura e de controle.

Pode-se dizer que o CPI executa um *parsing* de mensagens OpenFlow inicialmente, e, logo após, analisa o conteúdo delas. Além desse serviço básico de *proxy*, o CPI pode fazer descarte de primitivas maliciosas antes que elas cheguem no controlador. Como exemplo,

a SHE utiliza o CPI para detecção rápida de eventuais problemas na rede. Qualquer outro protocolo da SBI, além do OpenFlow, pode ser suportado pelo CPI futuramente.

3.2.4.2 *Network Database*

O *Network Database* (NDB) armazena informações da rede (usualmente coletados pelas CoEs) e informações gerenciais da SONAr. As informações da rede podem ser métricas, dados de topologias, alarmes, *logs* etc. As informações gerenciais são coletadas por meio de eventos (recebidos ou gerados pelas entidades da SONAr), informações administrativas inseridas ou configuradas no NAD, informações de predição e análise gerados nas SLEs e histórico de ações tomadas pelas SOEs. Posto de outro modo, qualquer informação que eventualmente passe pela SONAr pode ser armazenada no NDB.

O armazenamento de dados é feito para fins de histórico ou até mesmo de processamento posterior (como naqueles processamentos executados pelas SLEs). O NDB é uma estrutura distribuída e, inicialmente, qualquer entidade da SONAr pode fazer consultas (*queries*) ou atualizações nesse banco.

3.2.4.3 *Network Event Manager*

O *Network Event Manager* (NEM) fornece a estrutura de *publisher/subscriber* necessária para manipular eventos da arquitetura. Lembremos que qualquer entidade na SONAr pode gerar eventos ou subscrever-se a determinados eventos (em forma de tópicos). O NEM é uma estrutura distribuída e tolerante a falhas, permitindo que eventos sejam trafegados entre as entidades.

A importância de um elemento desse tipo está no controle de tópicos e filas que ele proporciona, além da rápida distribuição e cópia de eventos que são trafegados entre as camadas da SONAr. Frisando que a comunicação entre as entidades da arquitetura são feitas no modelo *event-driven*, portanto, não há comunicação no modelo *request/response*. Caso uma entidade deseje utilizar um serviço de outra, a publicação de um evento através do NEM é obrigatória. Conforme mostrado na Figura 6, as intituladas 'Entidades' na SONAr trocam eventos com o NEM, isto é, entidades publicam eventos que são recebidos por outras entidades.

3.2.4.4 *Integration Entities*

As *Integration Entities* (IEs) contém entidades que recebem eventos e interoperam com outras camadas de rede ou sistemas de TI. Elas são necessárias pois implementam particularidades de determinado domínio de rede que não é responsabilidade das entidades que monitoram e tomam decisões (SOEs, SLEs ou CoEs). Por exemplo, a SHE pode tomar a decisão de solicitar ao controlador SDN do domínio uma alteração de rota. Não compete

à SHE implementar o protocolo de integração com o controlador SDN. Para isso, há uma IE específica. As IEs especificadas até o momento estão descritas a seguir.

Auto-Boot Manager

O *Auto-boot Manager* (ABM) é usado para iniciar o ambiente. Alocação de recursos, instanciação de VNFs, *deploy* de aplicações da própria SONAr, entre outras, são tarefas deste componente. O funcionamento do ABM se dá em conjunto com a SCE (responsável pela criação de fluxos de controle e gerenciamento nos NEs). Dessa forma, um administrador de rede fica incumbido de inicializar o serviço ABM[2]. Este, por sua vez, inicia todos os módulos da SONAr necessários no *bootstrapping*. Portanto, após o *start* do ABM, toda inicialização e reinicialização de componentes de gerenciamento e controle se dá autonomamente; toda operação necessária, quando um novo NE é inserido no domínio, se dá autonomamente (descoberta e configurações). A Seção 3.3 descreve em detalhes o funcionamento do ABM em uma implementação da arquitetura SONAr.

Network Service Broker

O *Network Service Broker* (NSB) é usado pela SONAr para integração com outros sistemas. Ele trabalha como um *broker*, fornecendo *Application Programming Interface*s (APIs) que são expostas para o ambiente externo e requisitando serviços de APIs dos demais sistemas (como controladores e orquestradores). Um controlador SDN de um determinado fornecedor – *vendor* – possui API em um formato diferente dos demais controladores de demais fornecedores. Ainda, versões diferentes de um controlador de um mesmo *vendor* podem ter formatos de APIs diferentes. O NSB possui os serviços implementados para integração com os controladores e orquestradores diversos.

Além da Camada de Controle, o NSB possui interoperabilidade com a Camada de Aplicação. Por exemplo, sistemas *Operations Support Systems* (OSS) e *Business Support Systems* (BSS) podem requisitar serviços da SONAr expostos no NSB. Ressalta-se que os serviços expostos possuem alto nível de abstração e compete à SONAr transformá-los em um nível de função de rede.

Network Administration Dashboard

O *Network Administration Dashboard* (NAD) fornece uma ferramenta administrativa no formato Web para interagir com as entidades da SONAr. Através dessa ferramenta, os operadores de rede podem visualizar graficamente métricas, alarmes, *logs* e outras informações, bem como gerenciar entidades (por exemplo inserindo limites para serem usados pelos algoritmos das SOEs). A intervenção humana para administração da rede

[2] Por razões óbvias, alguém precisa dar o primeiro comando de inicialização de forma manual.

é feita exclusivamente pelo NAD, facilitando o trabalho de operadores e administradores de redes.

3.2.4.5 *Local Agents*

Os LAs são aplicações inseridas localmente em NEs da Camada de Infraestrutura ou em servidores das camadas de Controle e Aplicações. Cada LA tem como objetivo principal a coleta de informações locais – tais como métricas de desempenho de servidores, links diretos e capacidade de recursos – e envio dessas informações para as entidades SONAr. Enfatiza-se que as informações coletadas poderiam ser devidamente coletadas pelas CoEs; todavia, em algumas circunstâncias, o envio de informações no sentido NE $\rightarrow$ SONAr pode ser necessário.

Um cenário no qual LAs podem ser benéficos é aquele de manutenção da saúde da rede, de forma imediata. Caso um problema ocorra, o LA pode identificá-lo localmente e enviá-lo de prontidão às entidades SONAr. Outro benefício dos LAs é evitar *flooding* na rede: o LA pode coletar informações localmente e sumarizá-las para enviar em períodos mais longos para a SONAr; por exemplo, o LA pode coletar localmente informações sobre memória de um NE e só enviar essas informações à SONAr caso o valor de memória usado esteja maior que determinado limite pré-configurado.

Para garantir a saúde de NSs durante tempo de execução, foi necessário o desenvolvimento de LAs, durante este trabalho, pois era a única forma de coletar informações de múltiplas rotas em uma topologia de rede OpenFlow; no Capítulo 4 justifica-se essa afirmação.

Esta seção descreveu todos os componentes SONAr especificados. Segundo nossa análise, os componentes reportados formam a arquitetura para autogerenciamento de uma rede auto-organizável. Contudo, do ponto de vista de implementação, não se faz necessário o desenvolvimento de todos os componentes. Para o intuito desta tese, duas implementações distintas foram utilizadas e serão descritas a seguir.

3.3 Implementações

Até o momento da escrita desta tese, realizou-se duas implementações da SONAr. A primeira está implantada na infraestrutura do Laboratório de Internet do Futuro da Faculdade de Computação (FACOM)/UFU e possui como foco a disponibilização de um ambiente SONAr para que outros pesquisadores possam aprofundar em entidades particulares da arquitetura; a segunda está implantada no Testbed IRIS (IRIS, 2021), e possui como foco a integração com um *framework* de orquestração de redes E2E denominado *Hierarchical Orchestration of End-to-End Networks* (HOEN). Ambas implementações são brevemente descritas nesta seção, pois foram utilizadas para avaliação do aspecto de autocura da SONAr.

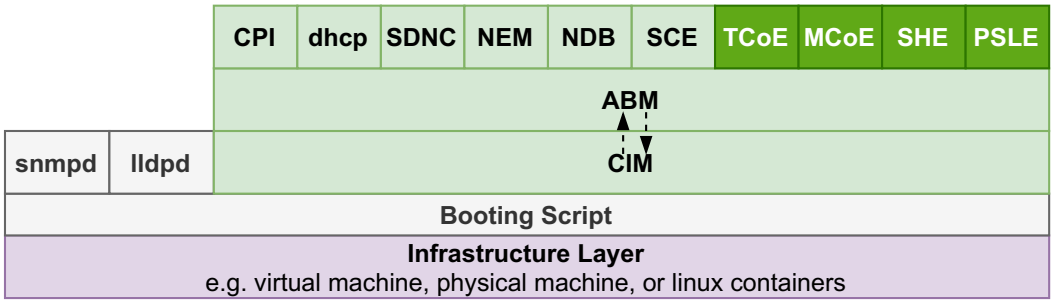


Figura 7 – Implementação SONAr.

3.3.1 *Framework* para OAM

Nesta tese, desenvolveu-se o *framework* NOSH baseado na arquitetura SONAr para análise da funcionalidade de autocura. Os servidores que hospedam o *framework* podem ser físicos ou virtuais e possuem basicamente a Implantação (*deployment*) do ABM e do *Containerised Infrastructure Manager* (CIM). O CIM é um sistema de orquestração de *containers* desenvolvido especialmente para a SONAr. Esse orquestrador instancia os *containers* necessários para funcionamento do *framework* – por exemplo, a instanciação do NDB e do NEM é feita através do CIM. Ainda, o CIM possui recursos para aplicar ações de recuperação (cura) do *framework* de certas anormalidades, ou seja, é um sistema fulcral para autocura – o Capítulo 4 detalha a funcionalidade de autocura do CIM.

Após o CIM executar, o ABM torna-se responsável pela orquestração das entidades SONAr. Portanto, a ordem de execução dos componentes, bem como as configurações de fluxos de encaminhamento nos NEs, são orquestrados pelo ABM. Para que o *framework* possa criar as regras de fluxos nos NEs, o CIM executa um controlador SDN no *bootstrapping* – nessa implementação, optou-se pelo controlador ONOS, por sua característica de *carrier-grade*, familiaridade dos desenvolvedores, APIs abertas e *benchmarkings* encontrados na literatura. Detalhes particulares do ABM não são tratados nesta tese, visto que está em outra vertente de pesquisa do grupo MEHAR.

A Figura 7 apresenta os módulos do *framework* SONAr implementado em uma visão de componentes. A *Infrastructure Layer* está presente pois o *framework* não deixa de ser uma aplicação rodando em uma infraestrutura – não há distinção para a infraestrutura necessária para se rodar o *framework*, podendo ser servidores físicos ou virtuais, *containers* LXC ou ainda *clouds* privadas ou públicas. Na infraestrutura, é adicionado um *script* que deve ser executado para que o *framework* se inicialize. Nessa implementação, utilizou-se um *shell script* que inicializa os seguintes componentes: (i) CIM desenvolvido em Java 8; (ii) um serviço Linux SNMP através do utilitário *net-snmp* (SNMP, 2019); e (iii) um serviço Linux *Link Layer Discovery Protocol* (LLDP) através do utilitário lldpd (LLDPD,

2019). A partir dessa inicialização, o CIM executa o ABM e este passa a manter a infraestrutura.

Para o *bootstrapping* autônomo, o ABM executa diversos *containers* Docker (MERKEL, 2014) contendo as entidades SONAr e o SDNC. Referindo brevemente, o ABM inicializa as aplicações listadas a seguir:

- Controlador SDN ONOS: utilizando uma imagem Docker pública, o controlador SDN – SDNC – é inicializado para controlar os NE OpenFlow da infraestrutura. Ressalta-se que para essa implementação optamos pelo *Open vSwitch* (OVS), porém outras implementações podem utilizar NEs diversos;
- *Dynamic Host Configuration Protocol* (DHCP): uma aplicação que permite que o *framework* entregue IP para cada NE encontrado na topologia;
- CPI: uma aplicação desenvolvida para ser o *proxy* entre os *switches* OpenFlow e o ONOS;
- NDB: utilizando uma imagem Docker do Apache Cassandra (Apache Software Foundation, 2021; CASSANDRA, 2014). O ABM inicializa o banco de dados e roda o *schema cql* necessário para criação das tabelas que são criadas inicialmente;
- NEM: utilizando uma imagem Docker do RabbitMQ (VMware, 2021a; BOSCHI; SANTOMAGGIO, 2013);
- SCE, TCoE, MCoE, SHE e PSLE: entidades fundamentais desta tese para inicialização do ambiente e aplicação de autocura.

As entidades SONAr listadas acima, que não foram criadas a partir de imagens Docker públicas, foram desenvolvidas em linguagem de programação Java 8 com o *framework* Spring Boot (VMware, 2021b). As escolhas das tecnologias utilizadas baseou-se na análise de padrões da indústria e familiaridade dos desenvolvedores. Como exemplo, o NEM poderia ter utilizado outro *message broker* – a SONAr foi concebida como uma arquitetura de referência e agnóstica a tecnologias. Após todas as aplicações listadas acima estarem em operação, a SONAr descobre o desenho da topologia, através do lldpd e TCoE.

Após as informações de topologia chegarem na SONAr, a SCE configura as rotas de comunicação entre os NEs e o ONOS – passando pelo CPI. Essencialmente, a SCE define as rotas de menor custo aplicando o algoritmo de Dijkstra; após essa etapa, a MCoE inicia a coleta de informações e a SHE e a PSLE aplicam suas funções de autogerenciamento conforme será detalhado no Capítulo 4.

A Figura 8 mostra o ambiente de *testbed* na FACOM/UFU no qual a implementação sugerida nesta seção pode ser implantada com o auxílio da ferramenta GNS3 (NEUMANN, 2014). Como exemplo, os componentes da Figura 7 são inseridos na máquina virtual

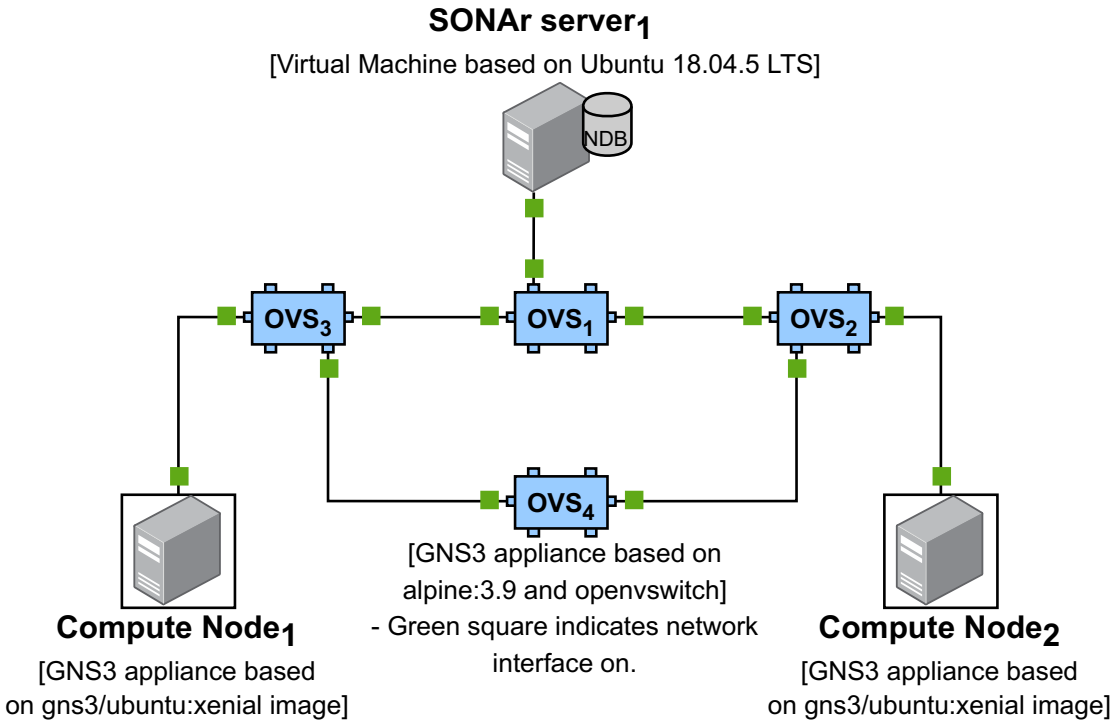


Figura 8 – Exemplo de Ambiente de Implantação SONAr na FACOM/UFU.

SONAr server$_1$ ou ainda entre várias máquinas. Note-se que, na Figura 8, o controlador ONOS também roda em SONAr server$_1$, porém poderia estar rodando em uma máquina separada. A configuração de rotas da SCE pode ser exemplificada através do OVS$_4$. Este NE possui duas formas de chegar até SONAr server$_1$: através do caminho OVS$_3$–OVS$_1$–SONAr server$_1$ ou através de OVS$_2$–OVS$_1$–SONAr server$_1$. A SCE define uma rota e aplica as regras de encaminhamento em cada OVS pelo caminho. Cabe, posteriormente, à SHE manter o canal de comunicação do plano de controle (entre OVS$_4$ e SONAr server$_1$) funcionando corretamente; caso necessário, a SHE deve reconfigurar as rotas.

Repara-se que a Figura 8 é uma topologia configurada momentaneamente e rodando no *testbed* FACOM/UFU com alguns OVS (Linux Foundation, 2021). Os procedimentos de operação e autocura mencionados no parágrafo anterior, bem como alguns adicionais, são propostos no Capítulo 4 e avaliados no Capítulo 5.

3.3.2 *Framework* para Manutenção de *Slices* em Tempo de Execução

Além da implantação da SONAr na FACOM/UFU, desenvolveu-se um *framework* para OAM em redes E2E, com vistas à orquestração de NSs em tempo de execução.

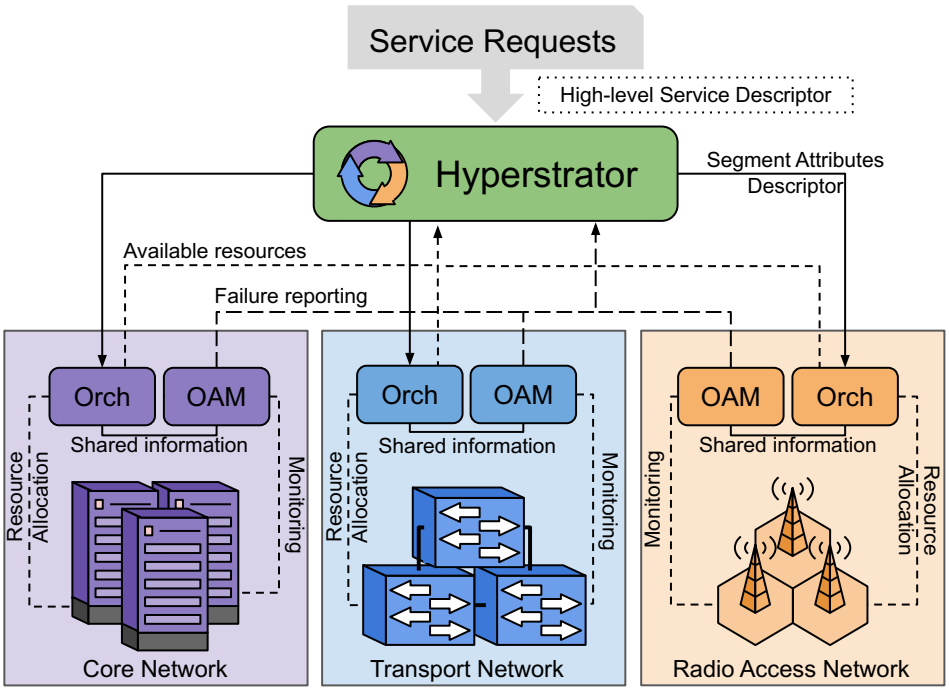


Figura 9 – Exemplo de Ambiente de Implantação SONAr em uma Rede E2E.

Esse *framework* foi integrado ao HOEN (SANTOS et al., 2018), o qual se trata de um *framework* para orquestração de NSs em redes E2E. Por simplificação de implementação e contemporaneidade dos dois projetos, decidiu-se pelo desenvolvimento das entidades SONAr na estrutura HOEN.

A Figura 9 mostra um exemplo do ambiente HOEN. A rede E2E é constituída de três diferentes segmentos, o *Core Network* (CN), *Transport Network* (TN) e *Radio Access Network* (RAN). O objetivo do HOEN é orquestrar NSs ao longo dos diversos segmentos; dessa forma, a criação de NSs é requisitada por Provedores de Serviço – SPs – a uma entidade denominada *Hyperstrator*. Essa entidade garante a consistência no aprovisionamento de NSs junto a orquestradores específicos de cada segmento, de modo que particularidades do segmento sejam tratadas para que os NSs sejam criados com os requisitos de QoS demandados pelos SPs. Na implementação em questão, optou-se pela implantação da SONAr como OAM no TN. Dessa forma, o orquestrador do segmento garante o aprovisionamento e descomissionamento dos NSs, enquanto o *framework* OAM assegura os requisitos de QoS durante a fase de execução dos NSs.

A implementação da SONAr junto ao HOEN foi feita através do desenvolvimento de um protótipo utilizando-se a linguagem de programação Python3. Para o NEM, utilizou-se o ZeroMQ (The ZeroMQ authors, 2021) por já ser um sistema *Publisher/Subscriber* utilizado no HOEN e atender os requisitos *event-driven architecture* do NEM. Para o NDB, utilizou-se uma estrutura em memória, justificável por ser um protótipo.

Ressalta-se que esperamos que a SONAr seja um modelo de referência para autoge-

Tabela 2 – Tecnologias Utilizadas nas Implementações SONAr.

Componente	**Implementação FACOM/UFU**	**Implementação HOEN**
Controlador SDN	ONOS	Ryu
NDB	Apache Cassandra	In-memory
NEM	RabbitMQ	ZeroMQ
SOEs, SLEs e CoEs	Spring Boot + Java 8	Python3 + ping + iperf3
LAs	-	Python3
NEs	Docker + OVS	LXC + OVS

renciamento de redes SDN/NFV. Todavia, as escolhas de implementação ficam a cargo do implementador, que deve levar em consideração: tempo de desenvolvimento, características do ambiente para o qual a arquitetura vai ser implantada, requisitos de negócio e engenharia, dentre outros. Por este motivo, apresentou-se duas implementações da SONAr já realizadas com tecnologias em sua totalidade *open-source*, mas implementações futuras não estão descartadas – a arquitetura SONAr é aberta e divulgada em trabalhos acadêmicos, mas implementações não possuem restrições, podendo-se construir produtos comerciais ou *open-source* a partir da arquitetura. A Tabela 2 mostra as diferentes tecnologias adotadas em cada implementação, demonstrando-se a característica agnóstica de tecnologia da SONAr. No Capítulo 4 justifica-se a utilização das duas implementações descritas nesta seção com vistas a autocura para os planos de Dados, Controle e Gerenciamento e as contribuições diretas desta tese nessas implementações.

Capítulo 4

Uma Proposta de um Sistema de Gerenciamento de Autocura

Para a especificação de um *framework* que aplique o fundamento de autocura foi considerado prioritariamente o aspecto filosófico por trás do conceito. Portanto, evocando a definição de Ganek e Corbi (2003), um sistema que aplique autocura deve ser capaz de identificar um problema no próprio sistema e se recuperar. Posto isto, pode-se frisar que, ao se construir um *framework* OAM para autocura, o próprio *framework* deve identificar mau funcionamento interno – de outra forma, poderia-se alegar que o *framework* possui capacidade de Cura (*healing*) e não de Autocura (*self-healing*).

O *framework* descrito neste capítulo, denominado *Network Operations Self-healing* (NOSH), considera que o sistema que está sendo autogerido é uma rede de computador ou uma rede móvel nas tecnologias SDN e NFV. O uso de NFV não é um requisito para utilização da SONAr, porém facilita a coleta de métricas e recuperação de falhas. Dessa forma, os planos de Dados, Controle e Gerenciamento constituem o sistema. Para aplicar o fundamento de autocura, o *framework* proposto é capaz de autogerir a rede e, em consequência, se curar, isto é, identificar e se recuperar de anormalidades em qualquer um dos três planos. Esse autogerenciamento não é simples, posto que por vezes, os planos mencionados possuem interseção (conforme exposto no Capítulo 3). Neste capítulo, os fatores para aplicar autocura nos três planos são descritos na medida em que se especifica o NOSH.

Uma maneira de manter o sistema em funcionamento em tempo de execução é através da criação de um canal lógico pelo qual o *framework* OAM monitora e aplica ações de recuperação. Esse canal lógico é uma contribuição do presente trabalho e será detalhado na Seção 4.1. O NOSH proposto é estruturado através da arquitetura SONAr, aplicando fortemente os conceitos preconizados no Capítulo 3. Uma vez que a interoperabilidade entre os componentes da arquitetura não foi apresentada, a Seção 4.2 detalha a interoperabilidade necessária para autocura da rede.

As fases para autocura aqui descritas podem ser sumarizadas como: (i) inicialização

autônoma do sistema e seus diversos módulos; (ii) monitoramento do desempenho do sistema; e (iii) aplicação de ações para evitar ou recuperar de falhas de desempenho do sistema. A fase (i) está especificada na Seção 4.3, levando em consideração os ambientes de experimentação disponíveis durante a elaboração desta tese – a saber: *testbeds* FACOM/UFU e IRIS. A fase (ii) está descrita na Seção 4.4, contando com a descrição das métricas a serem coletadas para análise e possível identificação de falhas de desempenho. E, finalmente, a fase (iii) está descrita na Seção 4.5 e conta com as ações para recuperação de falhas ou no mínimo suas mitigações.

4.1 *Slices* de Gerenciamento

Network Slicing é um conceito chave para redes móveis futuras, como 5G, pois permite que aplicações com requisitos de QoS diferentes compartilhem a mesma infraestrutura de rede. Este conceito preconiza que os serviços que demandam diferentes requisitos sejam alocados em redes virtuais isoladas, denominadas Fatias de Rede – *Network Slice*s (NSs). Dessa forma, cabe ao gerenciador da rede, nas camadas de Controle e Gerenciamento, operar o aprovisionamento e manutenção de um NS ao longo dos diferentes segmentos de rede que compõem a rede E2E. Curar NSs que passem por problemas de desempenho é um dos objetivos do *framework* aqui proposto.

O objetivo supracitado diz respeito à manutenção de um NS em tempo de execução, para que o mesmo consiga entregar QoS dentro do SLA acordado no aprovisionamento. Não obstante, utilizamos o conceito de *Network Slicing* para definir um novo conceito de gerenciamento – o qual denominamos '*Slices* de Gerenciamento'. Um *slice* para gerenciamento é uma porção/fatia da rede que será criada como um canal lógico de comunicação entre as entidades necessárias para manter a rede no estado 'Normal', isto é, saudável. Dessa forma, o gerenciador[1] possui um NS para envio de primitivas de controle/gerenciamento, e ainda para manter recursos computacionais isolados e garantidos. Fiéis ao aspecto filosófico de autocura, os *slices* de gerenciamento precisam autocurar, ou seja, o gerenciador precisa observar e recuperar imediatamente e prioritariamente *slices* de gerenciamento nos quais ele próprio está inserido.

4.1.1 Rede Lógica para Plano de Controle

A Figura 10(a) mostra um controlador SDN, SDNC_1, operando uma topologia multi-caminhos – *multi-path* – básica, com três NEs conectados topologicamente em anel. Cada NE comunica-se com SDNC_1 para troca de primitivas de controle através de fluxos de encaminhamento adicionados nas tabelas de fluxos dos NEs. A figura mostra um *slice* de gerenciamento no qual NE_3 utiliza para comunicação com o SDNC_1. De fato, a linha

[1] A partir deste ponto, 'gerenciador' será usado para caraterizar o *framework* NOSH em adição a controladores, orquestradores e aplicações SDN/NFV.

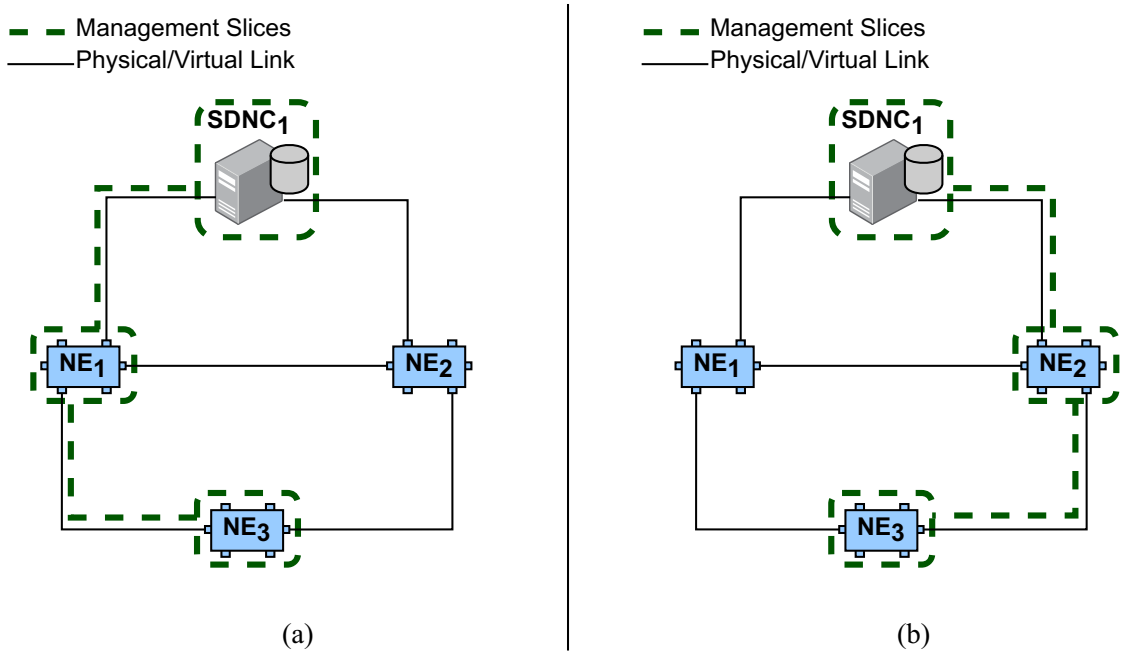


Figura 10 – *Slice* de Gerenciamento do Plano de Controle.

tracejada mostra quais partes das camadas fazem parte desse *slice*, a saber: (i) NE_3 e NE_1, que são utilizados no caminho da camada de controle; (ii) $SDNC_1$, que é o controlador propriamente dito; e (iii) os links NE_3–NE_1 e NE_1–$SDNC_1$, que fazem parte da rota de controle.

Posto isso, o NOSH deve cuidar da saúde de $SDNC_1$, NE_3 e NE_1, ou seja, garantir que possuam recursos computacionais suficientes e não se sobrecarreguem. Este cuidado faz parte do gerenciamento, porém na prática, os recursos utilizados fazem parte da camada de Infraestrutura, ou seja, devem garantir que os NEs e servidores do *slice* de gerenciamento possuam recursos para que não exista risco de primitivas de controle enviadas pelos NEs não chegarem ao controlador. Para recuperação de problemas, algumas ações do NOSH podem ser: migrar o controlador $SDNC_1$ para outra parte da infraestrutura, caso ele esteja sendo executado em uma máquina virtual ou *container*; reiniciar o controlador ou algum NE caso apresentem problemas computacionais; aumentar recursos de memória e processamento das máquinas; ou alterar o *slice* de gerenciamento através de balanceamento de tráfego ou reconfiguração de roteamento.

A Figura 10(a) possui o *slice* de gerenciamento tracejado assimetricamente apenas para representar sua flexibilidade. Quando um controlador SDN é migrado de local, pode-se alterar o *slice*, pois ele passa a englobar mais ou menos elementos de rede ou componentes de gerenciamento. De modo similar, caso uma reconfiguração de rotas ocorra, o *slice* de gerenciamento pode se alterar (abstratamente) de forma a considerar outros NEs. Esse roteamento pode ocorrer por diversas razões. Uma razão evidente seria um problema no

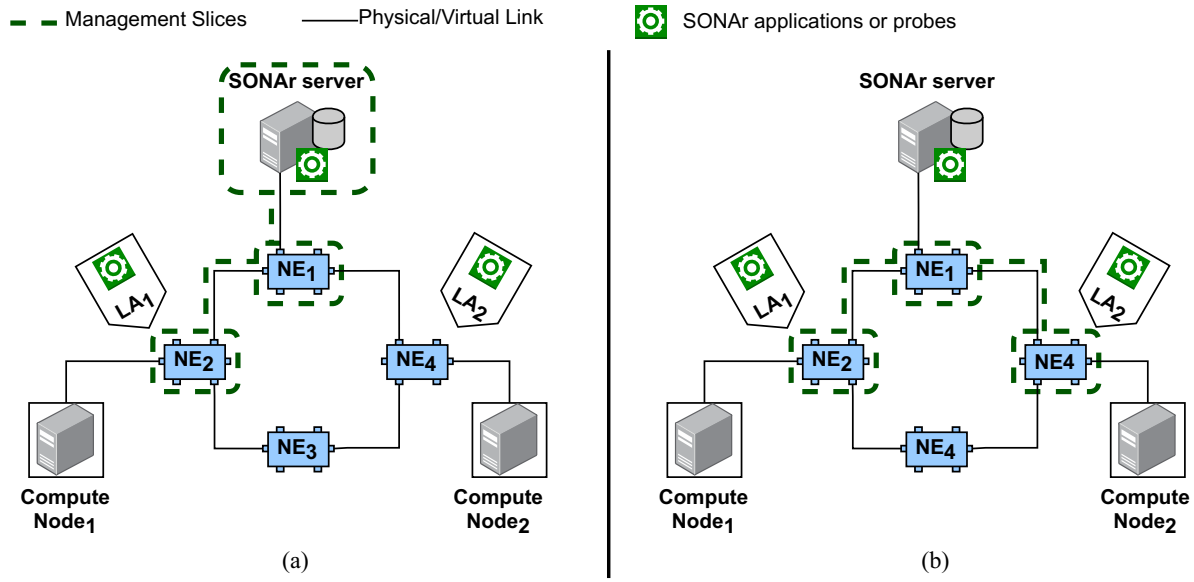


Figura 11 – *Slices* de Gerenciamento do Próprio Plano de Gerenciamento.

caminho da fatia do *slice* de gerenciamento – na Figura 10(a) o link NE_1–$SDNC_1$ pode apresentar algum problema (por exemplo *link down*, sobrecarga ou bloqueio) e o NOSH pode reconfigurar as rotas de NE_3 até $SDNC_1$ para utilizar o caminho NE_3–NE_2–$SDNC_1$. Nesse caso, o NOSH passa a cuidar preferivelmente do *slice* de gerenciamento apontado na Figura 10(b).

Ressalta-se que o *slice* de gerenciamento não considera apenas a camada de Infraestrutura, ou seja, ele é um canal lógico de comunicação (que compartilha recursos de infraestrutura do plano de dados) englobando as demais camadas. Dessa forma, o *slice* deve cuidar das aplicações SDN de $SDNC_1$. Reinicialização dessas aplicações, *scale-out*, aumento de recursos de memória/processamento fazem parte de ações de manutenção que o NOSH deve executar. Por fim, sobre as rotas aplicadas nos NEs, deve-se afirmar que rotas que fazem parte de *slices* de gerenciamento devem ser aplicadas com prioridade e, no caso de utilização de *queues* (como é possível no OpenFlow 1.3+), deve-se aplicar *minimum-rate*, *maximum-rate* e *priority* adequados; por esta razão, intitulamos essa rede lógica como *slice* de gerenciamento, visto que utiliza conceitos de *Network Slicing* como a criação de uma rede lógica sobre uma infraestrutura compartilhada.

Na Figura 10, o NOSH cria e gerencia três *slices* de gerenciamento, sendo um para cada NE. Ressalta-se que elementos de OAM, como o próprio NOSH, são englobados e se comunicam através de *slices* de gerenciamento na nossa proposta.

4.1.2 Rede Lógica para Plano de Gerenciamento

A Figura 11(a) mostra um servidor no qual o NOSH está implantado, SONAr *server*, e dois LAs implantados em NEs que possuem nós computacionais conectados. Este é outro caso de uso, no qual a comunicação entre os LAs e o SONAr *server* se dá por *slices* de gerenciamento. No exemplo, o LA_1 está implantado no NE_2 para coleta de informações locais, no mecanismo de *probing*. Após coletar as informações, LA_1 deve enviá-las à MCoE que executa no SONAr *server*; essa comunicação é efetivada através do caminho NE_2–NE_1–SONAr *server*. Como mostrado na Figura 11(a), um *slice* de gerenciamento é estabelecido para que a comunicação das primitivas de gerenciamento na direção LA_1 $\rightarrow$ SONAr *server* seja vigiada pelo NOSH (implantado/executando em SONAr *server*) o tempo todo. Ainda, o LA_1 é uma aplicação rodando no NE_2 e também faz parte do *slice* e deve ter tratativa prioritária bem como os NEs, rotas e SONAr *server*.

A Figura 11(b) apresenta um exemplo com um *slice* de gerenciamento estabelecido pelo NOSH para comunicação entre dois LAs. Uma coleta de informações executada por LA_1 é feita localmente, como informações de hardware e interfaces de rede. Porém, a SONAr prevê a interação de LAs para coletas de métricas de desempenho, como por exemplo a latência entre alguns caminhos na topologia. No exemplo da figura, LA_1 comunica-se com LA_2 para coletar informações da latência média em determinado período de tempo entre NE_2 e NE_4. O NOSH utiliza essas informações para autocura de NSs (da camada de dados) que passem por esse caminho e não estejam entregando o QoS adequado. Um *slice* de gerenciamento é estabelecido entre LA_1 e LA_2 e o NOSH mantém e cura esse *slice* de anormalidades, garantindo que os planos de gerenciamento e controle estejam executando da melhor forma possível.

O SONAr *server* propriamente dito faz parte do *slice* de gerenciamento, bem como suas entidades. O NOSH, operando no SONAr *server*, mantém monitoramento e recuperação (reconfiguração de recursos) das entidades do NOSH, assim como do servidor propriamente dito. Por fim, enfatiza-se que a configuração dos caminhos (rotas) utilizados por *slices* de gerenciamento é realizada de forma autônoma pelo NOSH e será detalhada na Seção 4.3.

4.1.3 Rede Lógica para Gerenciamento Inter-domínio

Com o estabelecimento de *slices* de gerenciamento, o NOSH consegue corrigir ou mitigar falhas nos planos de Gerenciamento e Controle, buscando manter a topologia de rede saudável, isto é, conectividade garantida entre componentes de controle/gerenciamento e elementos por eles controlados/gerenciados, bem como busca manter as aplicações de controle/gerenciamento em funcionamento. Ainda, um terceiro caso de uso para os *slices* de gerenciamento é a manutenção da saúde dos planos de Gerenciamento e Controle em domínios diferentes.

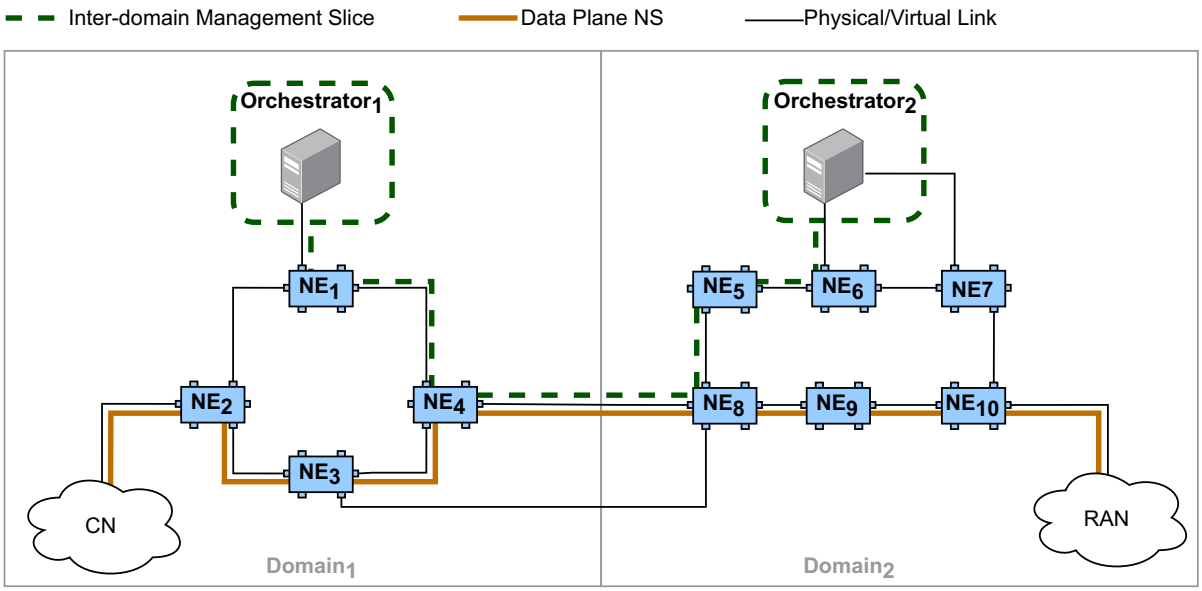


Figura 12 – *Slice* de Gerenciamento Inter-domínio.

O terceiro caso de uso diz respeito a diferentes MNOs que, eventualmente, vão precisar se comunicar para manter NSs do plano de dados em funcionamento. Um NS pode ser estabelecido através de duas ou mais MNOs. Como exemplo, é idealizado um NS que seja contratado por uma determinada indústria vertical para *Mission Critical Communication* (MCC) entre suas filiais. Por razões de uma única MNO não possuir rede (nesse caso infraestrutura de conectividade) suficiente para comunicação entre as filiais, o NS pode ser estabelecido passando pela rede de transporte – TN – de duas MNOs. O aprovisionamento não é um problema nesse caso, podendo ser realizado até mesmo manualmente. Todavia, a fase de manutenção do NS em tempo de execução é um problema, pois eventualmente uma MNO pode ter um problema em um link de interconexão que inviabilize a entrega do QoS daquele determinado NS. Uma primitiva de gerenciamento deve ser enviada imediatamente (assim que a falha for alarmada) de uma MNO para outra, para que autonomamente a segunda realize as reconfigurações necessárias.

Idealizamos a criação do *slice* de gerenciamento entre MNOs diferentes para garantir que primitivas de gerenciamento, essenciais em MCCs, sejam priorizadas. A Figura 12 apresenta um exemplo desse tipo de *slice*. Cabe ao NOSH (ou outro *framework* OAM similar) do Domain_1 manter os *slices* de gerenciamento em estado 'Normal' (saudável) a todo momento dentro de sua topologia, enquanto cabe ao Domain_2 manter no seu domínio. Na figura, dois orquestradores de NSs fazem parte da camada de Gerenciamento e precisam de comunicação ininterrupta entre si, mesmo estando em segmentos ou até mesmo domínios diferentes.

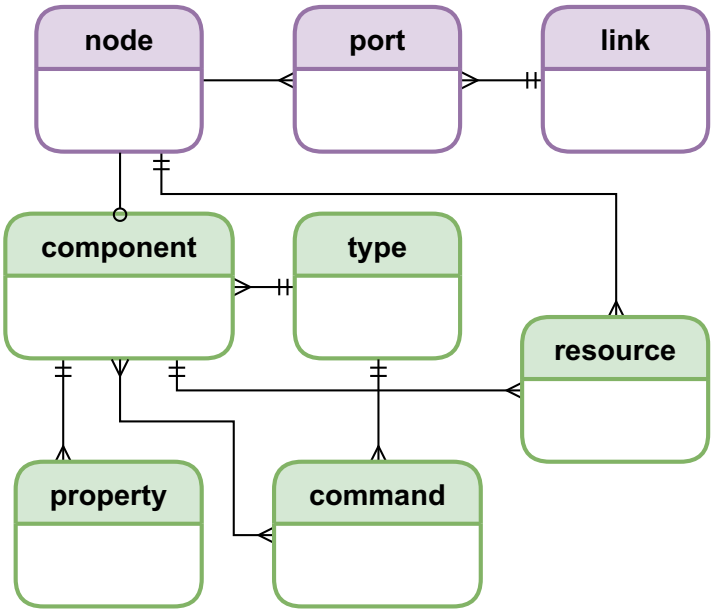


Figura 13 – Modelo Entidade-Relacionamento do Catálogo de Gerenciamento.

4.1.4 Catálogo de Gerenciamento

Os *slices* de gerenciamento precisam de meta-informações para distinguir quais elementos e caminhos na topologia devem ser operados, ou seja, quais são os NEs e componentes de gerenciamento que fazem parte do *slice* de gerenciamento. Essencialmente, o NOSH deve ter informações sobre cada recurso que está sendo gerenciado e qual sistema está gerenciando aquele recurso. Para isso, nós criamos o conceito de Catálogo de Gerenciamento, seguindo padrões de NFV – que possui outros catálogos para outros fins.

A Figura 13 apresenta uma proposta de catálogo para gerenciamento (os atributos das tabelas podem variar em cada implementação). As tabelas '*node*', '*port*' e '*link*' são previstas no NDB da SONAr e armazenam a topologia de rede descoberta pela TCoE – essas tabelas são essenciais para o funcionamento da SONAr, ao menos para autoconfiguração e autocura, independentemente do catálogo aqui proposto. As demais tabelas da figura representam o modelo de dados necessário para a operação autônoma dos *slices* de gerenciamento.

A tabela '*component*' é essencial e armazena cada elemento de controle/gerenciamento por onde passa um *slice* de gerenciamento. Portanto, há um registro nessa tabela para o NOSH, suas entidades, controlador SDN, VNFM etc. A tabela '*type*' armazena os registros necessários para distinguir diferentes tecnologias, por exemplo controlador SDN virtual/físico, OAM por *container* e VIM virtual. A tabela '*property*' possui propriedades básicas que cada componente de gerenciamento pode ter; por exemplo, as informações de *endpoint* de uma API REST de um controlador SDN são armazenadas nessa tabela. Complementarmente, a tabela '*command*' armazena comandos que podem ser executados

em um determinado componente. Nota-se uma relação entre '*type*' e '*command*', a qual é necessária pois um mesmo tipo de tecnologia pode ter comandos diferentes por ser de *vendors* diferentes ou, no caso de mesmo *vendor*, de versões diferentes. A tabela *command* pode ter comandos para monitorar um componente, bem como comandos para recuperar tal componente.

A tabela fulcral do Catálogo é *resource*, que representa uma tupla de $n = 2$. Cada componente de gerenciamento/controle pode gerenciar/controlar n recursos. Um controlador SDN, por exemplo, controla n NEs. Um *slice* de gerenciamento pode forçar a aparição de mais de uma linha na tabela '*resource*' para um mesmo componente de gerenciamento, pois um mesmo NE pode fazer parte de mais de um *slice* de gerenciamento.

O diagrama apresentado na Figura 13 mostra as principais tabelas do *framework*, necessárias para operação de *slices* de gerenciamento. Outras tabelas operacionais não estão representadas e podem ser criadas de acordo com a implementação. Por exemplo, a SHE pode possuir diversos microserviços, um para cada tipo de cura a ser executada. Portanto, o desenvolvedor baseia-se na SONAr, mas pode criar tabelas específicas para seus microserviços. A Seção 4.2 clareia a integração entre os diversos componentes do *framework* NOSH para funcionamento da arquitetura, incluindo funcionamento dos *slices* de gerenciamento.

4.2 Interoperabilidade de Entidades

A Figura 6, apresentada no Capítulo 3 para mostrar a disposição arquitetural dos componentes da SONAr, não transparece a interoperabilidade entre as entidades da arquitetura. Na prática, as entidades do *framework* NOSH são implantadas como *containers* rodando sob uma *engine* em um sistema operacional e, consequentemente, sob um servidor (virtual ou físico). A escolha do uso de *containers* se justifica pela portabilidade e padrão arquitetural da indústria. A portabilidade garante que as entidades possam ser migradas para outra parte da infraestrutura, bem como para *clouds* públicas ou privadas, sem muitas adaptações. O padrão arquitetural da utilização de *containers* parece ser uma boa prática na indústria, por flexibilizar interoperabilidade e ações para automatização, como inicialização, migração, e parada de aplicações.

A Figura 14 mostra a disposição de algumas entidades NOSH no processo de autocura. As camadas de Controle e Gerenciamento não estão representadas, pois na prática a conectividade do NOSH é provida pela Camada de Infraestrutura. Nessa figura, dois fluxos são apresentados. O formato da figura é para mostrar a interoperabilidade entre as entidades dentro do NOSH – as Seções 4.4 e 4.5 apresentam alguns diagramas de sequência de alguns serviços. O primeiro fluxo, representado em azul, é o fluxo de recebimento de um evento de alteração na topologia de rede. Esse fluxo, no NOSH, se inicia com o recebimento de uma mensagem na TCoE (a modificação de topologia é recebida pelo

serviço lldpd sendo executado conforme exposto na Seção 3.3.1).

Ao receber uma informação (passo 1), a TCoE a transforma em um evento SONAr e o publica em um tópico específico no NEM (passo 2). As entidades subscritas no tópico recebem o evento; neste exemplo, SHE recebe o evento (passo 3). A SHE busca informações relevantes (por exemplo topologia) no NDB (passo 4) e analisa a informação de modificação de topologia. Nesse caso, exemplifica-se um evento de *link down*. A SHE calcula uma rota alternativa para *slices* de gerenciamento que utilizem o link 'recém caído', gera comandos com ações a serem executadas e publica um evento no NEM com os comandos (passo 6). O NSB recebe o evento (passo 7) e requisita os comandos para o controlador SDN (passo 8).

Para clarear ainda mais a interoperabilidade das entidades, a Figura 14 mostra outro fluxo, em vermelho, indicando a coleta de métricas. Um determinado LA coletou (passo 1) métricas de desempenho de um NE e enviou-as para a MCoE (passo 2). A MCoE faz *parsing* das informações recebidas e as publicam, no formato de eventos, em um tópico no NEM (passo 3). O evento é recebido pela SHE (passo 4) que busca informações do NE no NDB (passo 5) e analisa se as métricas recebidas podem indicar alguma falha (passo 6). Em caso positivo, a SHE gera ações para recuperação da falha e monta os comandos. Os comandos são publicados como eventos no NEM (passo 7) e o NSB os recebe (passo 8). O NSB transforma os eventos em requisições para o controlador SDN e envia as requisições (passo 9). Recomenda-se a utilização do NSB para integração com controladores e orquestradores diversos. Porém, a SHE pode requisitar diretamente os controladores e/ou orquestradores em determinadas implementações.

O intuito desta seção é tornar mais clara a interoperabilidade das entidades SONAr entre si. Os dois exemplos supradescritos mostram entidades de diversos grupos (CoEs, SOEs e IE) interagindo. Qualquer outra comunicação dentro do SONAr é realizada de forma análoga: CoEs buscam ou recebem informações e as transformam em eventos; SOEs ou SLEs recebem os eventos e tomam ações baseadas no conteúdo. Ressalta-se que toda a comunicação entre os componentes internos se dá preferivelmente pelo protocolo *Advanced Message Queuing Protocol* (AMQP) (padrão da indústria para comunicação *publisher/subscriber*). A comunicação entre LAs e CoEs é particular de cada implementação. A comunicação entre IEs e os componentes de Controle, Infraestrutura e Dados fora do SONAr se dá preferivelmente pelo protocolo *Hypertext Transfer Protocol* (HTTP) (padrão da indústria para APIs restful ou *webservices*). A busca de informações em ACL iniciada pelas CoEs depende de cada ambiente, mas recomenda-se a utilização de protocolos já maduros na indústria, como o SNMP ou o *Open vSwitch Database* (OVSDB) (no caso de OpenFlow).

O monitoramento que parte dos LAs ou TCoE não é detalhado pois o será minuciosamente na Seção 4.4. As ações de recuperação criadas pela SHE serão detalhadas na Seção 4.5.

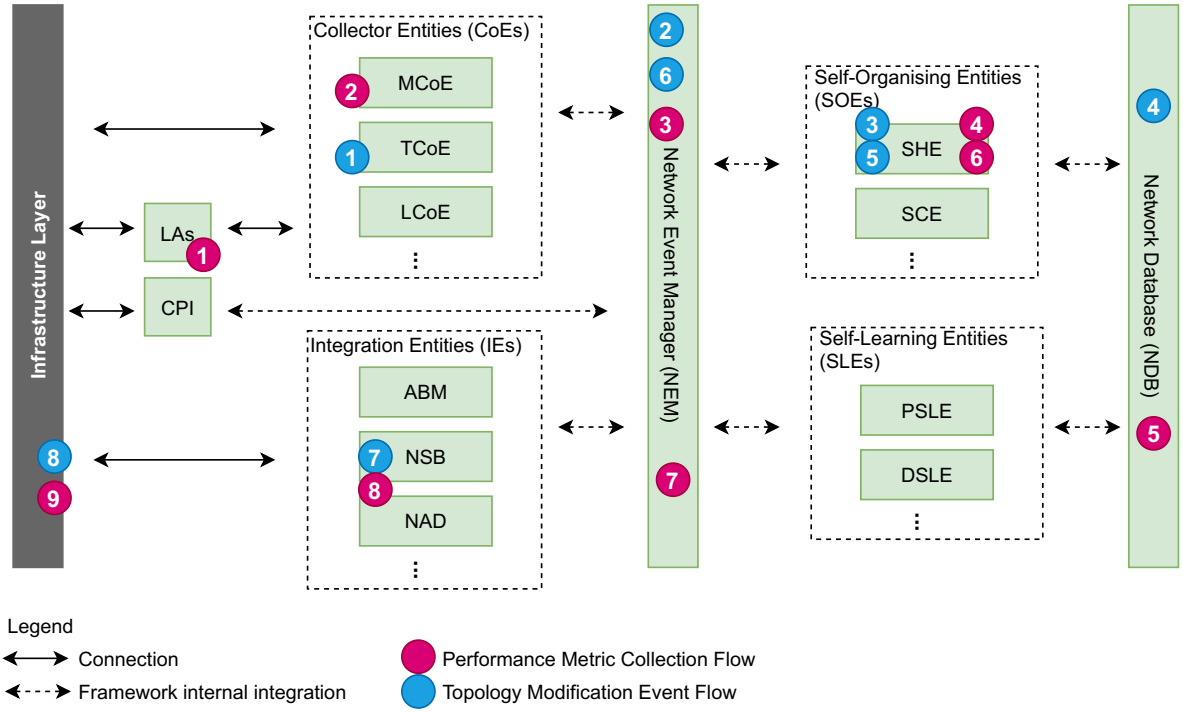


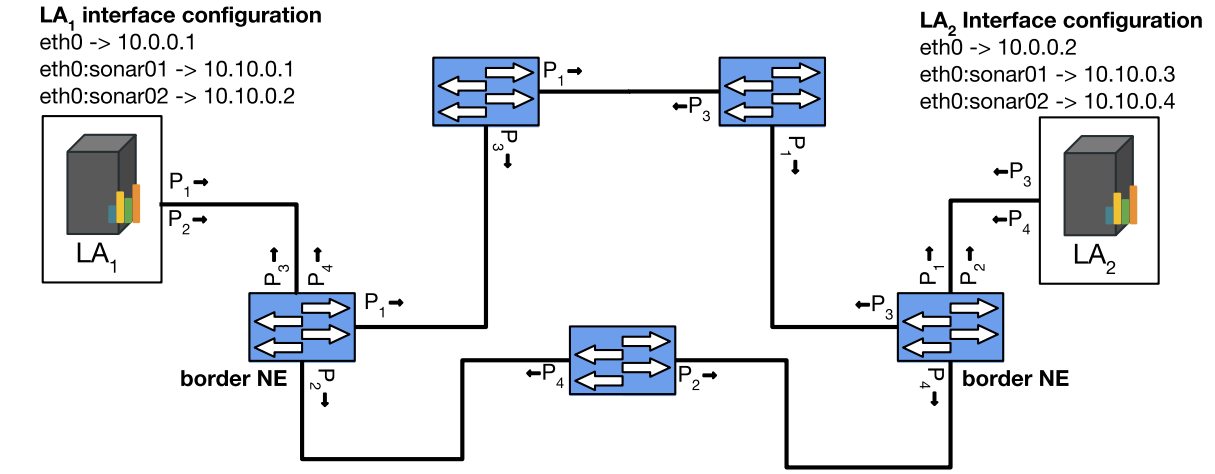
Figura 14 – Interoperabilidade dos Serviços das Entidades SONAr.

4.3 Inicialização Autônoma

O primeiro passo para estabelecimento dos *slices* de gerenciamento é a inicialização do *framework* NOSH. As entidades da plataforma são transformadas em imagens de *containers* que executarão em um servidor físico ou virtual. O *script* de *booting* é executado e o CIM trata de rodar – *run* – os *containers* e serviços necessários. Para a funcionalidade de autocura, faz-se necessário o estabelecimento dos canais de comunicação para as primitivas de gerenciamento e instanciação dos próprios serviços (que gera as meta-informações do Catálogo de Gerenciamento).

Quando o *script* de inicialização inicia os serviços (lldpd e snmpd) e o CIM (que executa os *containers* das entidades SONAr), o lldpd passa a enviar todos os eventos de topologia à TCoE. A TCoE envia as informações para a SCE, que salva a topologia recém descoberta no NDB. O CIM executa também o controlador SDN e o CPI, e a SCE calcula as melhores rotas entre cada NE da topologia e o CPI. O CPI e o controlador SDN podem ser executados no mesmo servidor ou em servidores diferentes. Após calcular as rotas, a SCE chama APIs do controlador SDN para criar fluxos de encaminhamento de cada NE até o controlador.

Logo após o estabelecimento das rotas de controle, a SHE (também inicializada pelo CIM) inicia os procedimentos de estabelecimento dos *slices* de gerenciamento e consequente inserção de informações no Catálogo de Gerenciamento no NDB. Todas as apli-



Management Traffic - Legend

Primitive	Source Address	Destination Address
P_1	10.10.0.1	10.0.0.2
P_2	10.10.0.2	10.0.0.2
P_3	10.10.0.3	10.0.0.1
P_4	10.10.0.4	10.0.0.1

Figura 15 – Coleta de Métricas de Desempenho por *Multi-paths*.

cações e serviços executados pelo CIM são inseridas no Catálogo pois necessitam de recuperação em caso de falhas. Dessa forma, armazena-se meta-informações acerca das entidades SONAr e serviços; por exemplo, armazena-se o endereço IP do servidor no qual a SCE está operando e comandos para reinicializar essa aplicação. Portanto, em caso de falhas na SCE, a SHE é capaz de reiniciá-la ou recriá-la. De forma análoga, todas as informações sobre as demais entidades e serviços são armazenadas no Catálogo. O monitoramento dessas aplicações e recuperação de falhas é descrito nas Seções 4.4 e 4.5.

O canal de comunicação dos *slices* de gerenciamento, ou seja, entre NEs e o servidor SONAr é executado de forma análoga às rotas de controles supramencionadas. Porém, os LAs possuem duas particularidades. A primeira é a criação de fluxos de encaminhamento entre cada LA e as CoEs. Dessa forma, os LAs são capazes de enviar informações coletadas. Para essa criação, o serviço dhcp elege um IP para cada LA e o CIM solicita à SCE a criação dos fluxos de encaminhamento. A SCE faz o *deploy* das aplicações que rodam nos agentes. O artefato de *deploy* já possui o IP das CoEs, de forma que cada LA saiba para qual entidade enviar informações coletadas. A SCE calcula a menor rota e cria os fluxos de encaminhamento nos NEs. A partir daí, a SHE é capaz de monitorar e manter as rotas entre LAs e CoEs saudáveis.

A segunda particularidade dos LAs é a criação das regras de encaminhamento entre eles próprios. Como a SHE monitora desempenho em topologias *multi-path*, cada LA se comunica com outros LAs através de diversas rotas. Como exemplo, a Figura 15 mostra

uma topologia *multi-path* básica (anel). Dois LAs estão implantados nos NEs de borda, quais sejam LA_1 e LA_2. O LA_1 se comunica com LA_2 através de duas rotas diferentes e, portanto, os NEs precisam de regras de encaminhamentos que considerem as duas rotas. Para tal, o *framework* NOSH define interfaces virtuais para que primitivas possam ser enviadas por rotas diferentes.

Para exemplificar, o LA_1 da Figura 15 possui uma interface 'eth0', para a qual o dhcp atribuiu IP 10.0.0.1. Para que os NEs saibam como trafegar primitivas de gerenciamento de LA_1 até LA_2, o NOSH calcula todas as possíveis rotas entre os dois LAs. No exemplo, há apenas duas rotas possíveis de $LA_1 \rightarrow LA_2$. Portanto, o NOSH define duas interfaces virtuais que vão ser criadas no LA_1 para que dois endereços IP diferentes sejam atribuídos. Assim sendo, o NOSH envia ao LA_1 as informações para criação de 'eth0:sonar01' e 'eth0:sonar02', com os respectivos IPs 10.10.0.1 e 10.10.0.2. Após isso, o NOSH define regras de encaminhamento e solicita a criação delas nos NEs. Na prática, as interfaces virtuais são criadas através do próprio sistema operacional e LA_1 pode se comunicar com LA_2 enviando, nas primitivas, o *IPv4 source address* 10.10.0.1 ou 10.10.0.2.

A Figura 15 mostra um exemplo de comunicação entre $LA_1 \rightarrow LA_2$ através dos dois caminhos possíveis, bem como na direção contrária $LA_2 \rightarrow LA_1$. O NOSH define regras de encaminhamento na direção contrária pois, por motivos de administração, o desempenho em uma direção pode ser diferente na outra. Por exemplo, por aplicação de políticas, a latência entre $LA_1 \rightarrow LA_2$ pode ser diferente da latência entre $LA_2 \rightarrow LA_1$. Essa configuração autônoma de LAs e regras de encaminhamento é análoga para qualquer topologia com qualquer quantidade de LAs. Na Seção 4.4, descreve-se como a coleta de informações ocorre após a inicialização autônoma.

A Figura 16 mostra um diagrama de sequência para ilustrar o funcionamento da inicialização autônoma dos LAs. Ao ser iniciado, o LA envia uma primitiva de gerenciamento `config_req` para a MCoE. Esta por sua vez requisita informações da topologia e demais LAs configurados na topologia ao NDB ou a orquestradores que porventura estejam executando no ambiente. A MCoE cacula as possíveis rotas entre o novo LA (que está se iniciando) e os demais. A partir daí, o controlador SDN é requisitado para criação das regras de encaminhamento nos NEs. Uma vez que as regras foram configuradas, a MCoE responde ao LA com uma primitiva `config_resp`, contendo as informações de interfaces virtuais que devem ser criadas. O LA cria as interfaces localmente e começa um ACL, enviando métricas de desempenho periodicamente para a MCoE. Esse tipo de monitoramento e os outros previstos para autocura são detalhados na Seção 4.4.

4.4 Monitoramento e Identificação de Falhas

As ações clássicas de autocura, como aquelas previstas nas SONs, são realizadas através de duas fases básicas: monitoramento e recuperação. A fase de monitoramento, por

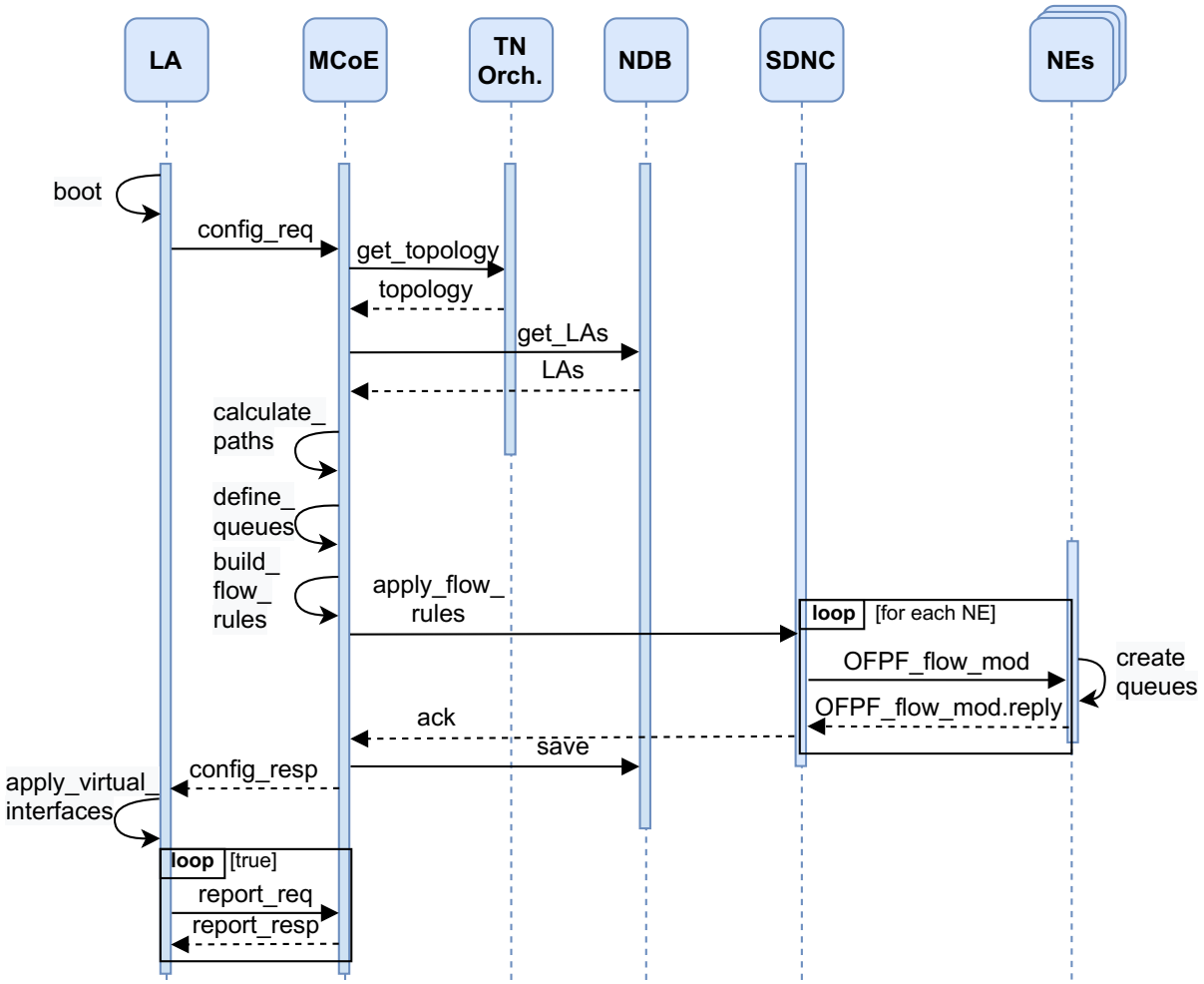


Figura 16 – Inicialização Autônoma de Agentes.

sua vez, pode ser dividida em duas sub-fases, que são: (i) coleta de métricas; e (ii) identificação de problemas que podem levar a falhas de desempenho. A coleta de métricas de desempenho do NOSH é feita através de três técnicas distintas e complementares, quais sejam:

- ❏ *Autonomic Control Loop*s (ACLs): Cada CoE aplica uma função de coleta de dados periodicamente nos elementos das camadas de Infraestrutura, Controle e Gerenciamento. Por exemplo, a MCoE coleta informações relativas a métricas enquanto a LCoE coleta arquivos de *logs* de aplicações. Esse *loop* de coletas se inicia após o provisionamento das informações de topologia e elementos de gerenciamento no Catálogo de Gerenciamento. Cada implementação pode ter diversos microserviços de coleta de informações que sejam pertinentes ao ambiente no qual o *framework* está sendo inserido;

- ❏ *Local Agent*s (LAs): Por algumas particularidades, a coleta em ACLs realizada pelas

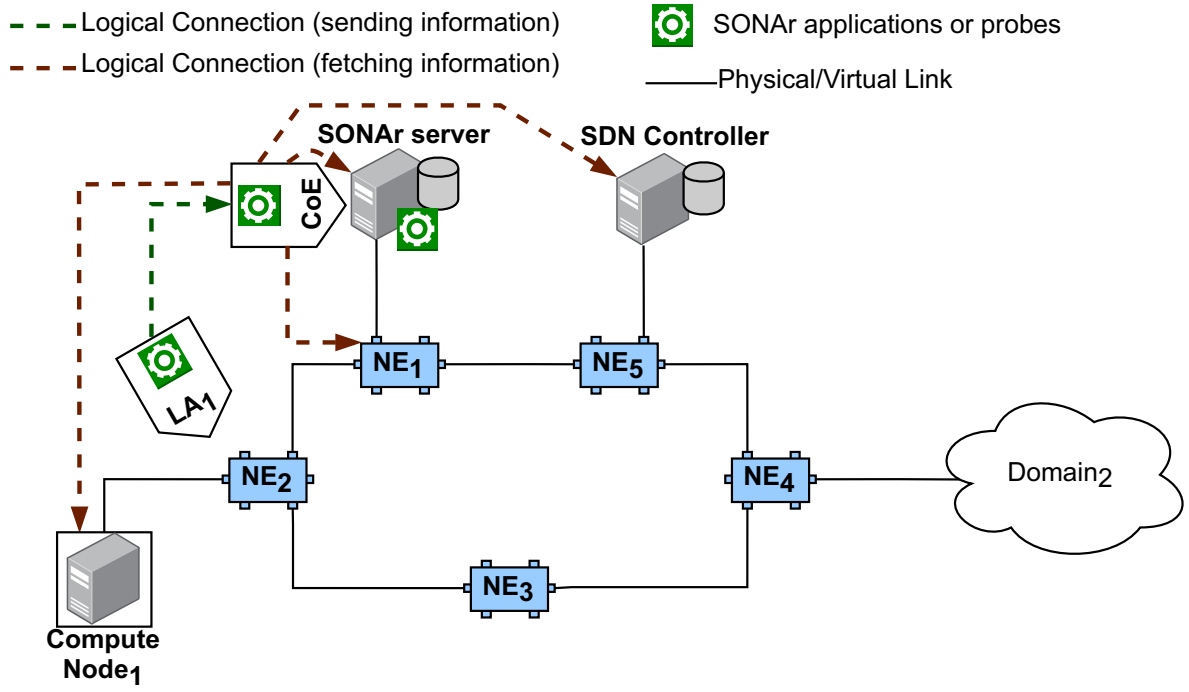


Figura 17 – Coleta de Métricas em Duas Direções.

CoEs pode não ser suficiente. Um exemplo é a coleta de informações de latência entre diversas partes da rede, conforme exemplificado na Figura 15. Nesse caso, faz-se necessária a implantação de agentes (aplicações) locais em elementos dentro da própria Camada de Infraestrutura. Os LAs processam localmente e coletam informações que posteriormente são sumarizadas e enviadas às CoEs;

- Interceptação de Primitivas na SBI: O CPI é implantado como um *proxy* entre os NEs e o controlador SDN do domínio. Essa interceptação permite que o NOSH receba primitivas de controle e encontre problemas que o controlador SDN não encontraria. Por exemplo, uma primitiva indicando um *link-down* pode não ter nenhuma tratativa pelo controlador e aplicações SDN, mas o NOSH pode decidir que esse *link-down* vai ocasionar uma falha nas rotas de controle ou nos *slices* de gerenciamento.

A utilização de ACL na direção CoEs → NEs possui vantagens pela simplicidade de implementação: como as CoEs rodam em conjunto no NOSH, e este por sua vez possui informações da topologia, é trivial o desenvolvimento das funções de coleta desde que se utilize protocolos padrões na indústria, como o SNMP ou o OVSDB. A desvantagem dessa técnica está no *flooding* de primitivas de controle/gerenciamento que são introduzidas na rede e quando há um elemento não atingível, ou seja, não há rotas disponíveis para a CoE chegar no elemento em questão. A utilização de LAs implica no envio de informações

na direção LA → CoEs. A principal vantagem é que um pré-processamento pode ser executado localmente e o LA não precisa enviar informações o tempo todo. Por exemplo, se o LA detecta que um NE possui entre 60% e 70% de utilização de memória RAM, essa informação não precisa ser enviada periodicamente a uma CoE, apenas quando houver uma variação grande de memória.

A Figura 17 exemplifica a coleta de informações em duas direções. As linhas tracejadas em verde indicam um LA_1 que está implantado no NE_2 enviando informações para uma CoE. Essas informações são enviadas através das rotas do *slice* de gerenciamento criado na inicialização do LA. As linhas tracejadas em marrom indicam a CoE, executando em um servidor denominado SONAr server, tomando a ação de coletar métricas. Nota-se que a CoE pode coletar informações de todos os elementos nas camadas de Infraestrutura, Controle, Gerenciamento e Aplicações. Por isso, a CoE coleta informações de um *Compute Node* na estrutura, bem como informações dos NEs (na figura está representado apenas a coleta em NE_1), e ainda informações do controlador SDN. As informações coletadas contém: recursos computacionais e capacidade, por exemplo quantidade de memória RAM, processador e disco sendo utilizados; estado de aplicações diversas, por exemplo status (*up/down*) de aplicações SDN no controlador; métricas de desempenho da rede, por exemplo RTT entre dois NEs; dentre outros. Todas as informações coletadas são utilizadas pelas entidades SONAr para avaliação e ressalta-se que não descrevemos todas aqui pois cada implementação de microserviços diferentes na SHE pode requisitar novos tipos de métricas que precisam ser adicionadas no NDB.

Uma vez que métricas estão sendo coletadas, a segunda fase do monitoramento é realizada, ou seja, o NOSH avalia as métricas para identificação de gargalos – *bottlenecks* – de desempenho. As SOEs avaliam as métricas em tempo real, isto é, imediatamente quando as recebem, para identificar gargalos que necessitam de ações imediatas. As SLEs podem avaliar as métricas em tempo real ou ao longo do tempo, em um pós-processamento, visando a predição de possíveis problemas futuros. Nesta tese, considera-se que as métricas avaliadas no plano de Dados podem indicar falhas em NSs e por isso o NOSH possui como foco as operações OAM em conjunto com um orquestrador de NSs. As métricas avaliadas no plano de Controle e Gerenciamento visam manter os *slices* de gerenciamento saudáveis, incluindo as rotas de controle/gerenciamento usadas pelo controlador SDN e pelo NOSH, aplicações SDN e entidades SONAr, além da infraestrutura (servidores, *containers*, *appliances*) no domínio que as entidades estão inseridas.

Em todos os casos supracitados (nos planos de Dados, Controle e Gerenciamento) a SHE possui implementados os microserviços para avaliação de métricas e integração com orquestradores de NSs e com controladores SDN. Com o Catálogo de Gerenciamento, a SHE é capaz de identificar quais rotas e quais aplicações serão afetadas por uma falha. Com a integração com orquestradores e controladores, a SHE é capaz de identificar quais elementos de controle ou NSs serão afetados pela falha. O processo de detecção das falhas

possui os seguintes passos: (i) um LA coleta informações locais e as envia para as CoEs ou qualquer CoE requisita informações de algum elemento da topologia (via APIs HTTP, telnet ou SNMP por exemplo); (ii) as métricas são transformadas em eventos, que são publicados em um tópico `metric` no NEM; (iii) a SHE, através de microserviços subscritos no tópico `metric`, recebe os eventos e inicia a análise; e, por fim, (iv) a SHE decide se as informações recebidas indicam alguma falha, requisitando informações adicionais dos controladores SDN, orquestradores NFV e NDB.

O mecanismo de predição da SONAr funciona de forma similar, porém as informações são recebidas por SLEs, como a PSLE. Os microserviços essenciais executam análises aritméticas considerando as métricas coletadas. As funções aritméticas resultam em possíveis problemas considerando o histórico de métricas previamente coletadas e armazenadas no NDB. Por exemplo, uma métrica indicando uma latência em uma determinada rota é analisada pela PSLE e, caso se determine um crescimento linear no valor da latência, é possível determinar quando um problema vai ocorrer. Valores de tráfego enviado/recebido por interfaces de rede também podem indicar deterioração de links, ou seja, pode-se prever quando um determinado link vai ficar congestionado e evitar esse congestionamento ou, ao menos, assegurar que os *slices* de gerenciamento não sejam afetados pelo problema.

Além de análises aritméticas, as SLEs podem conter algoritmos de Aprendizado de Máquina – *Machine Learning* (ML) com objetivo de prever ou definir situações na rede. Alguns algoritmos podem processar informações coletadas durante meses e identificar problemas de latência que ocorrem em determinados períodos. Como exemplo, um evento que ocorra todo domingo a tarde, como o *streaming* de uma partida de futebol, pode congestionar um link. Com essa predição, a SHE pode criar alguma ação de reconfiguração de recursos para que NSs não sejam afetados pelo congestionamento.

Além da rede propriamente dita, sobrecarga nos elementos de controle e na infraestrutura (alto processamento, crescimento de uso de memória ou aumento de uso de disco) podem ser previstas pelas SLEs. Ressalta-se que o intuito da presente tese é mostrar as funcionalidades que um *framework* como o NOSH pode oferecer. As funções de monitoramento propriamente ditas são inúmeras e podem ser implementadas posteriormente como novos microserviços. Todavia, entendemos que essa arquitetura (com esse mecanismo de monitoramento) permite que o NOSH aplique autocura em todas as camadas. A cura propriamente dita é feita através de ações para mitigação ou recuperação de problemas; tais ações são descritas na Seção 4.5.

4.5 Mitigação e Recuperação de Falhas

Após monitoramento e identificação de problemas que culminem em falhas nos planos de rede, o *framework* NOSH executa ações para mitigar ou recuperar de fato dessas falhas. A SHE é a entidade responsável pelas ações de recuperação propriamente ditas,

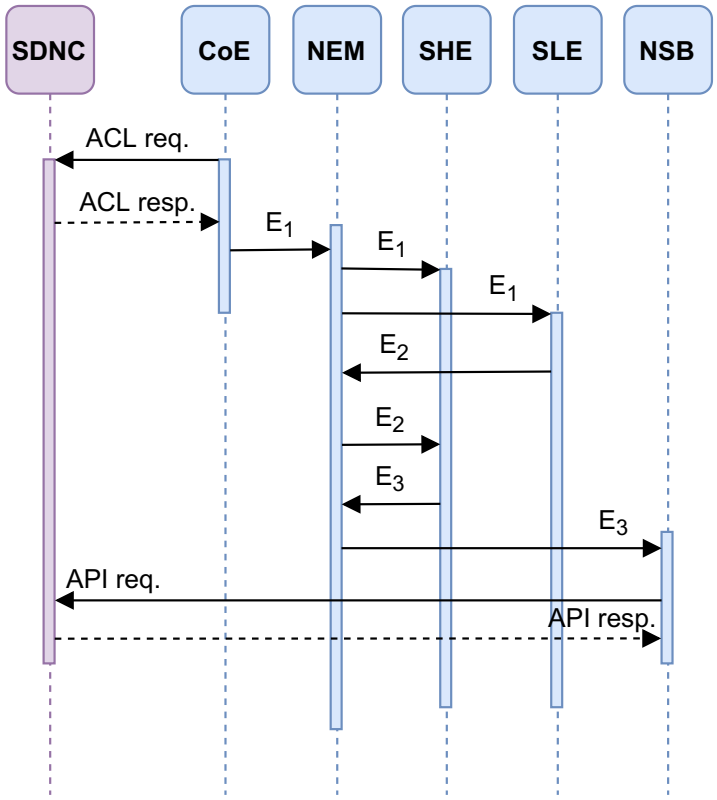


Figura 18 – Interoperabilidade Orientada a Eventos para Monitoramento e Recuperação.

podendo conter diversos microserviços para resoluções de diferentes problemas. Nesta seção, apresenta-se a interoperabilidade nas fases de monitoramento e recuperação, os mecanismos de recuperação da SONAr e alguns casos de uso de microserviços que podem estar contidos na SHE.

A Figura 18 mostra um exemplo da interoperabilidade na transição das fases de monitoramento e recuperação. Inicialmente, as CoEs requisitam informações a um elemento da rede, seja de qualquer plano. No exemplo, uma CoE requisitou informações a um controlador SDN. Ao receber a resposta da requisição, a CoE publica um evento E_1 no NEM. E_1 é recebido por duas entidades subscritas para aquele tipo de evento, no caso SHE e SLE. No exemplo, esse evento foi analisado pela SHE e nenhuma ação em tempo real foi necessária. Porém, a SLE identificou um possível problema futuro ao avaliar E_1 e decidiu que uma alteração na rede deve ser executada para evitar o problema. A SLE publica um evento E_2 de recuperação, terminando a etapa de monitoramento e identificação de falhas.

A fase de recuperação é realizada por SOEs em qualquer caso; isto é, mesmo recuperação de falhas identificadas pelas SLEs são executadas pelas SOEs, em especial a SHE. Na Figura 18, o evento de recuperação E_2 é enviado da SLE para o NEM. A SHE recebe o evento de recuperação E_2 e define as ações necessárias para aquele tipo de recuperação. Após definir uma ou mais ações – note-se que uma sequência de tentativas de recuperação

pode ser necessária –, a SHE publica uma ação de recuperação através de um evento E_3 no NEM. Esse evento E_3 é recebido pelo NSB, que gera os comandos necessários em um ou mais elementos (NEs, controlador, orquestrador ou servidor da infraestrutura) e requisita a execução desses comandos. No diagrama de exemplo da Figura 18, o NSB optou por requisitar um serviço do controlador SDN, requisitando uma API do mesmo.

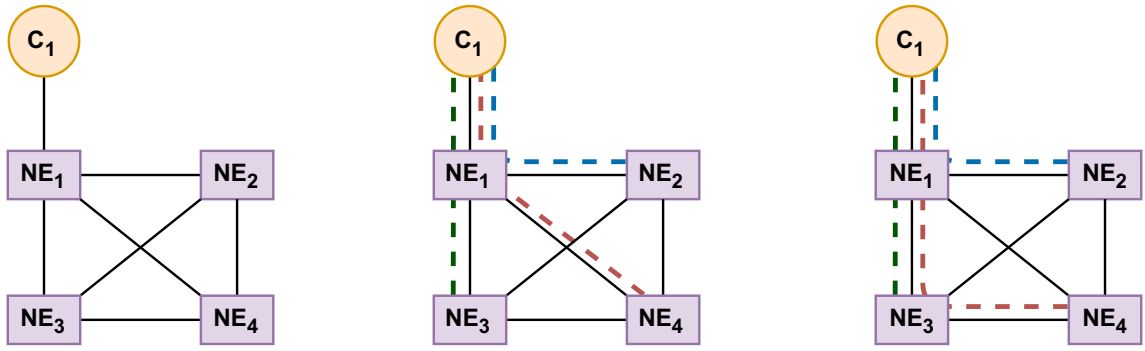
Quatro tipo de ações de recuperação podem ser definidas pelos microserviços da SHE, quais sejam:

1. Reinicialização – *Reboot*: é responsável por executar ações básicas para reiniciar componentes. Diversos componentes podem ser reiniciados, como por exemplo aplicações, controlador SDN, NE ou *containers* da SONAr (onde os componentes da arquitetura estão rodando). Os serviços de reinicialização devem se integrar com todos esses componentes e ser capazes de executar as atividades para reinicialização. Alguns componentes possuem API para reiniciar em modo *graceful*, como aplicações de controladores ONOS e ODL por exemplo, enquanto outros precisam receber comandos por *script* (normalmente *shell script* ou *script* de ferramentas específicas para gerenciamento de configuração, como *Ansible*);

2. Reconfiguração de recursos: é responsável por realizar configurações necessárias para voltar para o estado '*Normal*'. No plano de Dados, as configurações podem ser alterações em fluxos OpenFlow (alteração de rotas), reconfiguração de *slices*, alterações em VLAN etc. No plano de Controle, mensagens trafegando na SBI podem ter sua sintaxe alterada para evitar falhas nas aplicações ou no controlador SDN. Em alguns casos, a SHE pode realizar ações necessárias para as reconfigurações (chamando API nos controladores ou através de OVSDB/SNMP) ou publicar um evento para que a SCE faça as reconfigurações. Outra reconfiguração de recursos é o aumento de memória e processamento em nós computacionais. Por exemplo, roda-se a reconfiguração de memória de uma máquina virtual que esteja hospedando um controlador SDN;

3. Redefinição – *Reset*: quando ações de reinicialização ou reconfiguração não resolvem os problemas, a SHE pode fazer uma redefinição completa de determinado componente. Esse *reset* significa restaurar o hardware ou software e, eventualmente, aplicar um *restore* de determinado *backup*. Aplicações em *containers* podem ter *reset* através da recriação do *container* a partir da imagem inicial do mesmo (como é o caso do *reset* das entidades da própria SONAr). O ABM pode ser requisitado para efetuar o reinício de determinada topologia completamente;

4. Migração: alguma vezes parte da infraestrutura está com problema e é necessário migrar o local no qual determinado componente está implantado. Por exemplo, pode-se migrar o controlador SDN para uma parte saudável da topologia; migrar



(a) SDN topology. (b) Initial paths. (c) New path applied.

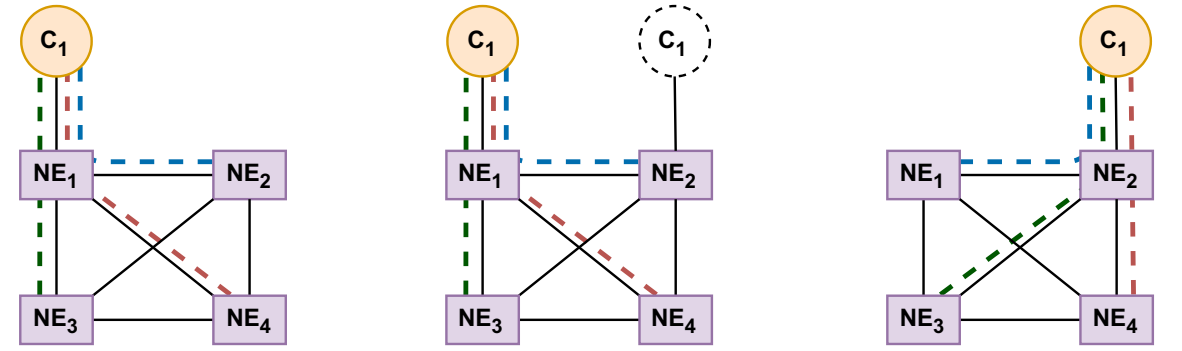
Figura 19 – Recuperação de Rotas do Plano de Controle.

switches virtuais para outro nó computacional que não passe por falhas; ou até mesmo migrar um *container* de uma *Virtual Machine* (VM) para outra. Considerando ambientes virtualizados, são inúmeras as ações de migrações que os microserviços podem executar, limitando-se à capacidade de recursos físicos do domínio (*cloud* em que estão sendo executados ou infraestrutura *bare metal*).

Ressalta-se que a sequência de ações – *workflow* – de recuperação executadas pela SHE pode incluir qualquer combinação dos itens acima mencionados. Por exemplo, se é identificado um problema em uma aplicação SDN que roda em um *container*, pode-se executar *reboot* do *container* (que é a ação mais rápida). Persistindo o sintoma de falha, pode-se executar um *reset* do *container* a partir da imagem de origem. Se ainda assim a aplicação não voltar ao estado 'Normal', uma migração pode ser executada (por vezes o problema está entre aplicação e infraestrutura). Dessa forma, várias ações de recuperação podem ser executadas sequencialmente na tentativa de eliminar o sintoma (falha) que está deixando o componente em estado não saudável.

Para discorrer sobre alguns cenários reais que a recuperação é executada pela SHE, apresenta-se nesta seção quatro casos de uso. Ressalta-se que diversos outros são possíveis, mas pela nossa experiência a arquitetura SONAr possui os componentes essenciais para recuperação de qualquer cenário em plano de Dados, Controle e Gerenciamento. Portanto, os quatro casos de uso aqui exemplificados servem para exercitar as capacidades de um *framework* para autocura como o NOSH.

O primeiro caso de uso é no canal de controle, afetado por um evento de desligamento de um link (previsto ou não). A primitiva OpenFlow de desligamento do link pode ser capturada pelo CPI. No caso do OpenFlow (versão 1.3+), normalmente a queda de um *link* é informada para o controlador pela primitiva OFPPS_LINK_DOWN. Ao



(a) Initial SDNC placement. (b) New SDNC instantiation. (c) Control paths migration.

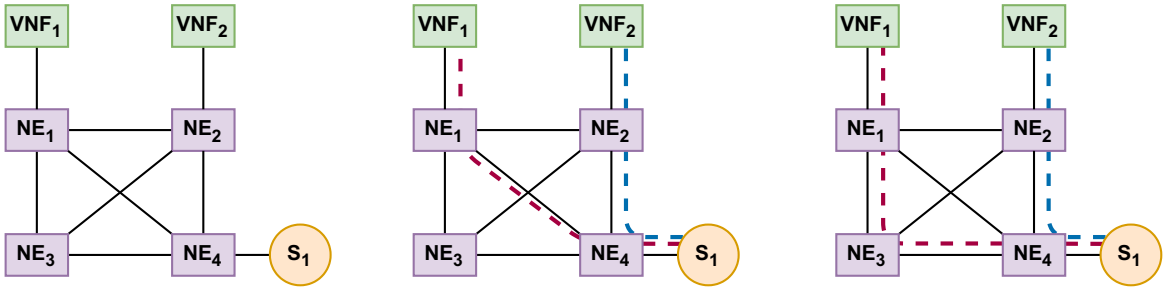
Figura 20 – Recuperação com Migração de Controlador.

interceptar essa primitiva, o CPI a duplica e envia a cópia (na forma de evento) para o NEM. A primitiva original é enviada para o controlador SDN. Outra forma de identificar esse evento é a TCoE receber um evento lldp indicando queda do link na camada 2. Usualmente a SHE e SOPE recebem o evento (estão subscritas nesse tipo de tópico). Após receber o evento, a SHE executa os algoritmos necessários para analisar o problema.

A Figura 19(a) mostra a topologia SDN do caso de uso. Na Figura 19(b) são mostradas as rotas entre cada NE e o controlador SDN C_1. Ao receber um evento de OFPPS_LINK_DOWN do link NE_1–NE_4, a SHE busca as informações de topologia no NDB e infere que o NE_4 vai ficar sem comunicação de controle com o controlador C_1. A SHE recalcula uma nova rota e gera ações de reconfiguração de fluxos de encaminhamento nos NEs para que um novo caminho entre NE_4 e C_1 seja estabelecido, conforme mostra a Figura 19(c).

O segundo caso de uso diz respeito a uma falha que pode ocorrer na infraestrutura e culmina na migração de um controlador SDN. Esse caso de uso é exemplificado na Figura 20. Ao identificar uma falha no link NE_1–C_1, no NE_1 propriamente dito, ou no nó computacional sob o qual C_1 está implantado, a SHE consulta o NDB e se depara com a topologia e rotas exemplificadas na Figura 20(a). Dessa forma, a alternativa de recuperação que SHE possui é a migração de C_1 para outra parte da infraestrutura, como por exemplo para um nó computacional conectado em NE_2. Para a migração, a SHE instancia o controlador C_1 no novo nó computacional, exporta os dados de C_1 anteriores e importa no novo, conforme mostra a Figura 20(b). Após isso, as rotas de comunicação do plano de controle são recalculadas e reaplicadas conforme mostra a Figura 20(c).

O terceiro caso de uso demonstra a atuação da SHE no Plano de Gerenciamento. Como exemplo, a Figura 21(a) ilustra um servidor S_1 conectado a um *switch* NE_4 e duas VNFs, VNF_1 e VNF_2, conectadas respectivamente a NE_1 e NE_2. O canal de comunicação



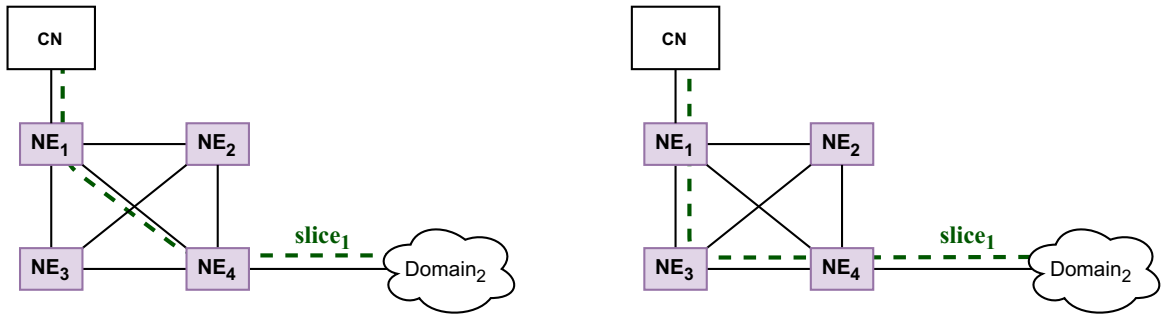
(a) NFV topology. (b) Calculated paths. (c) Alternative path.

Figura 21 – Recuperação de Rotas do Plano de Gerenciamento.

– *slice* de gerenciamento – está estabelecido através dos caminhos VNF_1–NE_1–NE_4–S_1 e VNF_2–NE_2–NE_4–S_1 (rotas previamente calculadas pela SONAr), conforme ilustrado na Figura 21(b). Na eventual falha, derrubada ou congestionamento do link NE_1–NE_4, a SHE deve recuperar o canal de comunicação entre VNF_1 e S_1; para isso, a SHE recalcula um novo caminho e aplica as regras de encaminhamento da nova rota, por exemplo VNF_1–NE_1–NE_3–NE_4–S_1, conforme apresentado na Figura 21(c). Ressalta-se que a recuperação é a reconfiguração das rotas – reconfiguração de recursos – e a SHE faz isso através da publicação de um evento SONAr e, por consequência, interoperando na direção NEM → NSB → controlador SDN → NEs.

Não obstante, nesta altura pode-se afirmar que as entidades SONAr são aplicações que exercem funções de rede, infraestrutura ou orquestração. Portanto, não seria inverídico afirmar que VNF_1 e VNF_2 poderiam ser entidades SONAr ou LAs. Por exemplo, na primeira suposição, S_1 poderia ser um servidor comportando CoEs, NEMs e NDB, enquanto VNF_1 poderia ser a SHE e VNF_2 poderia ser a PSLE. Na segunda suposição, S_1 poderia ser o *framework* NOSH enquanto VNF_1 e VNF_2 poderiam ser dois LAs aplicando *probing* nas interfaces de NE_1 e NE_2. Apesar de em nenhum grau haver impedimento de que as entidades SONAr sejam distribuídas, é recomendável que rodem como um único serviço para garantir que os eventos internos cheguem aos componentes o mais rapidamente possível. Ainda considerando a Figura 21, S_1 pode ser qualquer componente de orquestração ou controle de NFV, por exemplo VNFM, enquanto VNF_1 e VNF_2 podem ser qualquer recurso gerenciado pelo componente S_1.

O derradeiro caso de uso aqui demonstrado está ilustrado na Figura 22(a). Nele, demonstra-se a recuperação de um NS com sintomas que podem levar a falhas. O domínio em questão possui uma topologia multi-caminhos básica com quatro NEs conectados entre si, uma conexão intra-domínio com uma *Core Network* (CN) e uma interconexão com outro domínio – $Domain_2$. A figura exemplifica um NS fim-a-fim – $slice_1$ – que passa



(a) Network slice path. (b) Alternative path.

Figura 22 – Recuperação de Rotas de *Slices* de Rede.

virtualmente nos dois domínios. A topologia de rede apresentada mostra o transporte entre o CN e o outro domínio, que foi estabelecido através do caminho CN–NE_1–NE_4–$Domain_2$.

O NOSH, ao identificar uma falha no link NE_1–NE_4, deve recuperar o caminho entre as aplicações que utilizam $slice_1$ para comunicação e operação. Neste exemplo, a SHE deve observar se a falha pode afetar o $slice_1$ e, para isso, ela integra com algum orquestrador de NSs rodando no domínio. Se a falha, por exemplo latência alta, compromete a saúde de $slice_1$, a SHE reconfigura recursos (latência alta pode comprometer NSs que possuem requisitos de latência baixa). A SHE requisita informações dos NSs ao orquestrador e, ao notar que $slice_1$ vai ser afetado, recupera-o através da reconfiguração de rotas conforme mostra a Figura 22(b).

Os casos de uso apresentados nesta seção amparam a demonstração de recuperação que o NOSH é capaz de executar. Reforça-se que incontáveis casos de uso são possíveis através do *framework*, uma vez que a arquitetura está preparada para agrupar inúmeros microserviços nas SOEs (como na SHE). Há uma particularidade em relação a recursos compartilhados. Quando uma degradação no desempenho de um recurso compartilhado é identificada pela SHE ou SLEs, a SHE pode recuperar componentes que já apresentam falhas ou, como ação preventiva, pode reconfigurar recursos de componentes que ainda não apresentem falhas. Um exemplo claro de recurso compartilhado é uma rota na topologia de rede. Uma rota que esteja com alguma degradação de desempenho pode afetar diversas aplicações (NSs, clientes do plano de dados ou componentes do plano de controle e gerenciamento). A SHE age para recuperar alguma aplicação quando esta aplicação nitidamente vai ser afetada pela rota problemática: a isso denominamos Técnica de Recuperação em Tempo Real. Porém, há outro tipo de ação de recuperação que uma arquitetura como a SONAr pode efetuar: a Técnica de Recuperação Baseada em *Buckets*.

A recuperação em tempo real age para curar (reconfigurar, redefinir, reinicializar ou

migrar) recursos que já estão sendo afetados. Por exemplo, no caso de uso de recuperação de rotas de NSs, a SONAr recupera, ou seja, reconfigura rotas dos NSs que estão sendo afetados pela degradação da rota. Se alguns NSs requerem menos de 5 ms de latência mas a rota não está entregando essa latência, a SONAr reconfigura rotas para esses NSs. Todavia, se a degradação continua aumentando, dentro em breve a latência da rota atingiria 6 ms, 8 ms, 10 ms e assim por diante. Portanto, os NSs que requerem 10 ms de latência seriam afetados no futuro próximo. Para antecipar a recuperação de NSs que presumivelmente vão ser afetados pela mesma degradação, a SONAr pode aplicar a técnica de recuperação baseada em *buckets*.

O termo '*bucket*', utilizado para outros tipos de trabalhos na área, significa intervalos de valores pré-definidos. Determinada aplicação é inserida em um *bucket* dependendo do intervalo que ela se encontra. Por exemplo, uma aplicação de URLLC que requer menos de 1 ms de latência se encaixa no *bucket* de 0 a 1 ms, enquanto uma que requer menos de 5 ms se encaixa no *bucket* de 1 a 10 ms. Para latências utilizadas nos experimentos deste trabalho, implementou-se os seguintes intervalos: { 0, 1, 10, 30, 50, 70, 100, ∞ }, que foram escolhidos baseando-se nos requisitos de latência de aplicações futuras que podem ser encontrados em Parvez et al. (2018). Para explicar os *buckets* utilizaremos o caso de uso da Figura 22. Imagina-se que vários NSs estão rodando no domínio e compartilham a rota NE_1–NE_4. Ao identificar um aumento de latência nessa rota, a SONAr requisita ao orquestrador de *slices* quais são os NSs que utilizam a rota NE_1–NE_4 e seus respectivos requisitos de latência. Ao identificar uma latência de 5 ms, a SONAr não recupera (reconfigura as rotas) apenas os NSs que requerem menos de 5 ms. De outro modo, a SONAr reconfigura todos os NSs com latências no intervalo de 5 ms. Na implementação desta tese, SONAr recupera todos os NSs no intervalo de 1 a 10 ms. A essa técnica demos o nome de Recuperação Baseada em *Buckets*.

O objetivo do uso dos *buckets* é antecipar a recuperação de recursos que certamente serão afetados em breve. Com essa antecipação, a recuperação se inicia antes que o recurso seja afetado. Em grandes topologias com milhares de NSs por exemplo, a reconfiguração de rotas pode não ser tão rápida. Portanto a técnica dos *buckets* é eficaz e complementar à técnica da recuperação em tempo real. Ressalta-se que a técnica de *buckets* foi explicada com um exemplo prático. Porém, a técnica serve para qualquer outro tipo de recurso e não somente NSs. Os valores dos intervalos variam em cada caso de uso e devem ser decididos em tempo de implementação.

Para avaliar as duas técnicas de recuperação e os demais conceitos introduzidos neste capítulo, realizou-se as implementações descritas na Seção 3.3 e executou-se experimentos através delas para examinar várias funcionalidades aqui descritas, como inicialização autônoma, recuperação de falhas de rede e autocura do NOSH. No Capítulo 5, detalha-se os ambientes SDN e NFV nos quais o NOSH foi implantado e se discute os resultados obtidos nos experimentos para validar o *framework* e arquitetura SONAr, bem como o

fundamento de autocura em um sistema de redes de computadores atendendo planos de Dados, Controle e Gerenciamento.

Capítulo 5

Experimentos e Análise dos Resultados

Este capítulo descreve os cenários de experimentação utilizados para avaliar o fundamento Autocura em ambientes SDN e NFV, por meio do *framework* NOSH proposto no Capítulo 4, e apresenta os resultados obtidos. A realização dos experimentos demandou a implementação das funções de Autocura propriamente ditas na SONAr, bem como de partes dos componentes que não haviam sido desenvolvidas até o momento de concepção desta tese. Assim sendo, implementou-se as rotinas para estabelecimento dos *Slices* de Gerenciamento, e consequente criação da base de dados do Catálogo, a inicialização autônoma-programática dos LAs, as funções de monitoramento da saúde da rede e as funções de auto-recuperação das camadas de rede e do próprio NOSH.

A incorporação dos aspectos de autocura no NOSH está estritamente relacionada a decisões de implementação. Devido ao farto número de casos de uso que a arquitetura pode comportar, na fase de implementação, pode-se optar por qual o plano de rede alvo. Por exemplo, se o intuito da equipe de desenvolvimento for manter o plano de Dados saudável, aspectos de gerenciamento e orquestração de NSs e seus requisitos de QoS devem ser levados em consideração. Sob outra perspectiva, o time de desenvolvimento que pretenda implantar o NOSH, para cuidar da saúde do plano de Controle, deve se preocupar com o canal de comunicação entre NEs e Controlador. Para demonstrar o atingimento dos objetivos descritos na Seção 1.2, neste capítulo optou-se por experimentos baseados em casos de uso para cada um dos principais planos (Dados, Controle e Gerenciamento), frisando-se que no nosso entendimento, a Camada de Aplicações de rede está contida no Gerenciamento.

Inicialmente, descreve-se o método utilizado para validar a hipótese apresentada na Seção 1.3 e para demonstrar o funcionamento do NOSH. Na Seção 5.2, explica-se os casos de uso selecionados com suas respectivas camadas e os experimentos propriamente ditos, detalhando-se minuciosamente o ambiente SDN/NFV criado, componentes utilizados, topologias de rede (grafo de rede), hardware e simuladores, entre outros. Finalmente, na Seção 5.3, discute-se os resultados obtidos, os ganhos esperados com a utilização desse tipo de *framework* e limitações da nossa proposição.

5.1 Método para a Avaliação

Para avaliação do Plano de Controle, foi definido um ambiente com a plataforma ONOS, incluindo seu controlador OpenFlow padrão e *switches* OpenFlow como NEs. A preferência pelo ONOS se deu por suas características *carrier-grade*, sua adequada reputação em *benchmarking* e familiaridade dos pesquisadores. A seleção do OpenFlow como protocolo da SBI se deu por ser o protocolo mais utilizado em ambientes SDN.

O caso de uso selecionado foi o reestabelecimento do Controlador ONOS quando o mesmo sofre uma queda. Para tal experimento, foi forçada a queda do *container* no qual o ABM executou o ONOS e posteriormente a destruição completa do *container*. Nesses cenários, pretende-se mostrar ações possibilitadas com o NOSH, tais como a reinicialização de aplicações defeituosas em tempo de execução e também a redefinição de um controlador SDN.

Outro caso de uso escolhido tem como intuito avaliar o Plano de Gerenciamento. Para tanto, as aplicações comportando as entidades da própria SONAr são forçadas a entrarem em um estado não saudável. Como elas próprias fazem parte do *slice* de gerenciamento estabelecido no *bootstrapping*, o NOSH recupera essas aplicações aplicando as ações descritas na Seção 4.5, como reinicialização ou redefinição. O objetivo desse tipo de experimento é mostrar a capacidade de autocura propriamente dita, a qual prega que o próprio sistema – nesse caso o NOSH – é capaz de se recuperar de eventuais problemas. Para demonstrar a eficácia do NOSH neste quesito, forçou-se inclusive que uma própria aplicação SHE entrasse em estado não saudável. Com isso pretende-se validar que o NOSH pode curar e autocurar.

Em adição ao caso de uso supramencionado, uma função de rede DHCP também foi estabelecida e posteriormente submetida a um estado não saudável. O intuito é demonstrar que funções de rede como aquelas de NFV também possuem sua saúde gerenciada pelo NOSH. Os experimentos avaliam o tempo gasto para inicialização autônoma das entidades SONAr utilizadas e das funções de rede, bem como o tempo gasto para reestabelecimento das aplicações em questão.

Para o Plano de Dados, optou-se por avaliar aspectos de autocura em tempo de execução de NSs que possuem rigorosos SLAs de QoS. Através desses NSs, aplicações que requerem QoS específicos se comunicam e necessitam que esse QoS seja atendido durante toda a comunicação. Com este tipo de experimento, é possível demonstrar a interoperabilidade da SONAr com outros componentes do Plano de Gerenciamento – neste caso com orquestradores de NSs – e sua capacidade de cura em tempo real na fase de execução. A opção por NSs para avaliação do Plano de Dados se dá por dois fatores: (i) pelas nossas investigações, NSs possivelmente farão parte de ambientes SDN/NFV em redes futuras (5G e pós-5G); e (ii) pelo nosso entendimento, os controladores SDN padrão da indústria convenientemente já possuem capacidades de tolerância a falhas no Plano de Dados, não sendo necessário avaliação na presente tese.

Aproveitando o ambiente de orquestração de NSs, foi possível avaliar outros conceitos propostos no Capítulo 4 através dos seguintes métodos de avaliação: (i) avaliação do tempo gasto para inicialização autônoma de LAs e estabelecimento dos respectivos *slices* de gerenciamento; (ii) comparação da identificação de falhas por detecção versus a identificação de falhas por predição; (iii) comparação da recuperação em tempo real versus a recuperação com o conceito dos *buckets*; e, por fim, (iv) testes de desempenho para demonstrar o tempo gasto em monitoramento e o tempo gasto em recuperação.

A Seção 5.2 descreve os experimentos. Para melhor entendimento, são apresentadas imagens do ambiente SDN/NFV utilizado em cada caso de uso, com explanações sobre a localização de cada componente em sua respectiva camada, e imagens com as topologias de rede empregadas. Gráficos de tipos diferentes foram escolhidos para visualização adequada das comparações propostas nesta seção.

5.2 Experimentos

Nesta seção estão listados os experimentos realizados para avaliação do *framework* NOSH como ferramenta OAM para garantia de resiliência de um ambiente SDN e NFV. A resiliência deve garantir a saúde dos planos de Dados, Controle e Gerenciamento; no plano de Dados, deve-se considerar os SLAs de QoS solicitados por aplicações. Dois *testbeds* foram utilizados para experimentação, visando mostrar o aspecto agnóstico de tecnologia da SONAr, ou seja, a arquitetura é um modelo de referência que permite diversas implementações em diversos ambientes/infraestruturas.

5.2.1 Autocura em Redes Autoadaptáveis

Os experimentos desta seção procuram avaliar as técnicas idealizadas no Capítulo 4, sobretudo aquelas envolvendo o Catálogo de Gerenciamento e *Slices* de Gerenciamento. Para isso, desenvolveu-se o *framework* OAM NOSH e foi configurado um ambiente para executá-lo. Não é intuito desta seção avaliar exaustivamente as técnicas e procedimentos do NOSH, mas sim apresentar a interoperabilidade do *framework* com suas diversas entidades e discutir as funções na SHE e a inicialização – *bootstrapping* – da plataforma que executa o NOSH. A parte da avaliação da SHE é óbvia, mas a avaliação e discussão sobre a inicialização do *framework* pode causar certa estranheza. Acontece que, para manter a rede saudável, são necessários procedimentos adicionais na inicialização, o quais serão discutidos ao longo deste capítulo.

A implementação utilizada nos experimentos a seguir se baseia naquela descrita na Seção 3.3.1, com adição dos novos procedimentos de inicialização e as funções de Autocura (através de microserviços) propriamente ditas. A Figura 23 recapitula o ambiente FACOM/UFU. Observa-se quatro NEs que, na nossa implementação são *containers* rodando na *engine* Docker (MERKEL, 2014) com a instalação das bibliotecas do *Open*

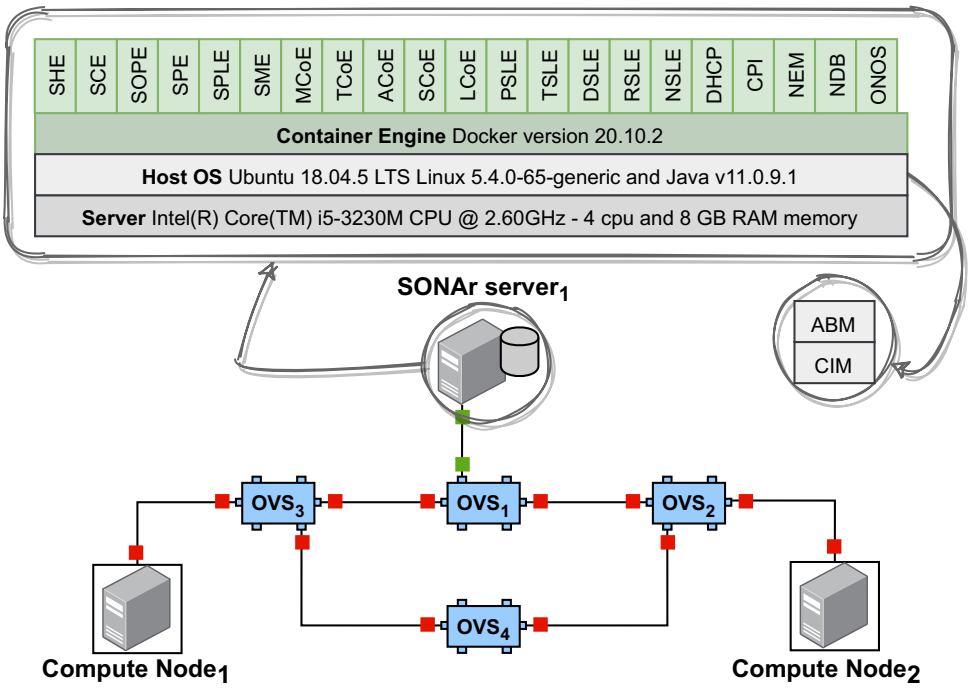


Figura 23 – Visão de Servidores e *Containers* para Experimentos de Autocura.

vSwitch (OVS) (Linux Foundation, 2021). Os experimentos não entram em detalhes sobre a SBI, pois a SONAr é agnóstica de tecnologia. Portanto, avaliou-se nesta seção procedimentos inerentes à arquitetura. O servidor SONAr da Figura 23 é o ponto fulcral da instalação, pois roda a plataforma de autogerenciamento como um todo, incluindo controlador SDN, aplicações de rede (SDN e NFV) e aplicações SONAr (entidades e componentes da plataforma).

O servidor SONAr – SONAr server$_1$ – possui uma espécie de lupa para detalhar tudo o que está rodando adentro. Foi utilizado um servidor Intel(R) Core(TM) i5-3230M CPU @ 2.60GHz com 4 núcleos e 8 GB de memória RAM – para todos os experimentos desta seção. Nesse servidor, instalou-se o sistema operacional Ubuntu 18.04.5 LTS com núcleo Linux 5.4.0-65-generic e Java v11.0.9.1. No nível do sistema operacional, contando com a JVM, implantou-se o ABM e CIM para, respectivamente, inicialização da rede e orquestração dos *containers* da SONAr. Portanto, a execução das aplicações ABM e CIM são as únicas ações manuais necessárias no NOSH. Ainda no sistema operacional, instalou-se a *engine* Docker versão 20.10.2 que abstrai as bibliotecas necessárias para execução dos *containers*. Acima da *engine*, todas as aplicações foram implantadas como *containers* – este é um ponto essencial para manutenção e autocura das aplicações de rede. Recomenda-se a utilização de *containers* para ações de cura (recuperação) e pela facilidade de migração entre servidores (caso necessário).

Para análise do Plano de Controle e Gerenciamento, criou-se *containers* para executar o controlador ONOS, uma função de rede DHCP e o CPI que, apesar de ser um

componente SONAr, possui função de *proxy* de rede na SBI. Para avaliação da SONAr propriamente dita, implantou-se algumas SOEs (SHE, SCE, SOPE, SPE, SPLE e SHE), algumas CoEs (MCoE, TCoE, ACoE, SCoE e LCoE), algumas SLEs (PSLE, TSLE, DSLE, RSLE, RSLE e NSLE), um *container* NEM e um NDB. Todas as imagens de *containers* SONAr utilizam o *namespace* 'meharsonar' e foram construídas basicamente a partir da compilação do Spring Boot e *build* da imagem localmente, exceto para o NEM, que utiliza uma imagem de RabbitMQ e o NDB, que utiliza uma imagem de Apache Cassandra. O ONOS também possui uma imagem própria, enquanto o DHCP é uma versão construída com Spring Boot que utiliza uma biblioteca de servidor dhcpd *open-source*.

As particularidades para autogerenciamento do Catálogo, necessário para a execução das funções de autocura propriamente ditas, são descritas nesta seção junto aos experimentos. O primeiro passo é a execução do CIM, que vai se estabelecer e conectar à *engine* de *containers* disponível no servidor. Em seguida, inicia-se o ABM, que vai inicializar todos os elementos de rede. Essa inicialização trata de executar todas as funções de rede, controladores, orquestradores, elementos de rede (como *switches*) e aplicações SONAr disponíveis no ambiente. Uma vez que o ABM encerra sua execução, todas as entidades SONAr são capazes de executar de forma individual e tratar do ambiente. Por exemplo, a SHE é capaz de identificar degradações nos planos de Controle, Gerenciamento e Dados e recuperar aplicações dessas degradações. A SHE é capaz de autocurar a si mesma – o pleonasmo foi utilizado nesta frase de forma proposital para deixar claro que a SHE deve possuir funções de autocura e não somente cura. Alguns dos experimentos elaborados na SONAr, com o ambiente em questão, estão descritos nesta seção. Ao leitor interessado, alguns detalhes de implementação podem ser encontrados no Apêndice A.

5.2.1.1 Análise da Inicialização das Entidades SONAr

Como mencionado previamente, a inicialização autônoma do SONAr permite que n entidades sejam executadas na plataforma em questão, sendo que tais entidades podem ser da SONAr ou VNFs que fazem parte da rede em questão. Neste experimento, executou-se o NOSH através da inicialização do ABM e CIM e coletou-se os tempos despendidos para cada aplicação. Nota-se que as aplicações DHCP e o controlador ONOS também foram executados da mesma forma, ou seja, o ABM tratou de inicializá-los. Cada aplicação foi executada através de um *container* na *engine* Docker. Isso é uma característica da implementação utilizada para o presente experimento, porém outras formas de inicialização podem ser implementadas em cenários particulares.

A Figura 24 apresenta todas as aplicações inicializadas pelo ABM. A abscissa lista a ordem (a partir da origem) na qual o ABM iniciou cada aplicação. A representação gráfica apresenta na ordenada, o tempo total cumulativo que cada aplicação gastou para ser iniciada. Ressalta-se que os tempos representados significam o tempo desde que o ABM iniciou cada inicialização até que o *container* daquela aplicação estivesse no estado

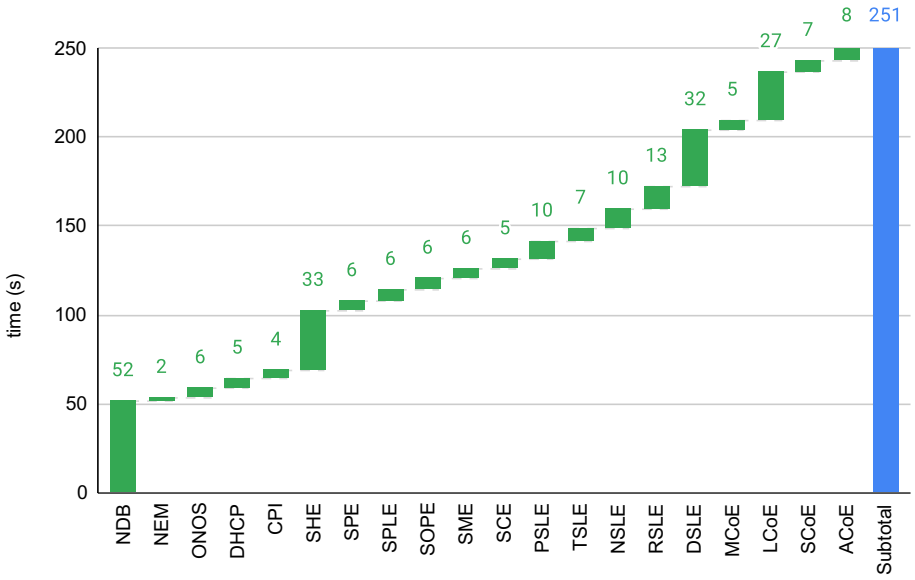


Figura 24 – Tempo de Inicialização do *Framework*.

`RUNNING`. Isso significa que, para cada aplicação, o ABM lê um arquivo de configuração contendo qual a imagem de *container* a ser utilizada para a aplicação, monta as variáveis de ambiente necessárias para a aplicação e executa o *container*. Executar o *container* significa que o ABM fez uma requisição na API do CIM. Após a execução do *container*, considera-se que a aplicação está sendo executada e o tempo final é medido. O ponto aqui é que cada aplicação pode demorar mais tempo para ser executada dependendo da quantidade de serviços que serão inicializados. Para o ABM, isso não faz diferença, visto que seu papel é inicializar a aplicação. Portanto, no *snapshot* mostrado na figura, a aplicação SPE gastou 5,608 segundos, arredondados para 6 segundos, para ser iniciada; porém, após sua inicialização, os serviços Java da SPE começam a ser executados e esse tempo não é relevante para esta análise, com exceção do NDB e da SHE.

Como pode ser observado nos resultados, o NDB gastou um tempo relativamente maior em comparação às demais entidades. Isso aconteceu pois o ABM precisa esperar que o NDB tenha seus serviços inicializados, ao contrário da maioria das aplicações, devido a caber ao ABM rodar o *schema* do banco de dados. Portanto o ABM inicia o NDB e aguarda que o serviço esteja completamente em execução para executar as criações de *keyspace* e tabelas. Para as demais aplicações isso não é relevante, por exemplo, o ABM precisa garantir que o NEM esteja sendo executado, mas não precisa aguardar que os serviços estejam em execução (os eventos não serão trocados internamente até que o NEM esteja executado).

A outra exceção do experimento é a SHE. Nota-se que o tempo de inicialização dela é maior que o das demais aplicações. Isso acontece devido à particularidades desta entidade. A SHE, ao ser executada, precisa de informações de todas as aplicações (componentes)

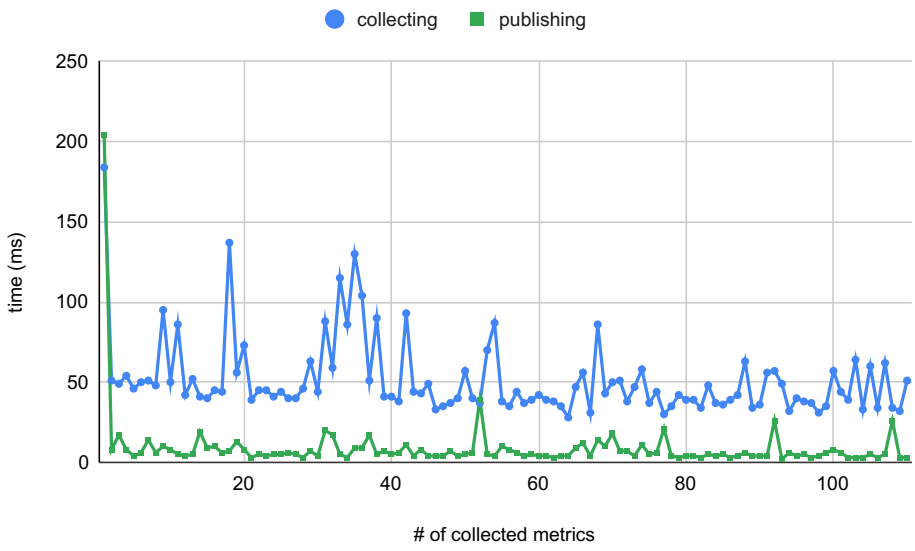


Figura 25 – Análise de Coleta de Métricas.

que o ABM executou, para serem armazenadas no Catálogo de Gerenciamento. Sem essas informações, a SHE não saberia quais aplicações fazem parte da Camada de Controle e Gerenciamento, ou seja, aplicações que a SHE deve manter saudáveis. O gráfico mostra o cumulativo de tempos e a SHE gastou 33 segundos para ser inicializada. O ABM inicializa a SHE e, quando esta está no estado `RUNNING`, ele segue a inicialização das demais aplicações. Com isso, o ABM finaliza a inicialização de todas as demais aplicações e posteriormente aguarda que ao menos uma aplicação SHE lhe envie um evento indicando que a inicialização dos serviços (da própria SHE) foi concluída. O ABM recebe o evento e envia um outro evento para um tópico específico contendo todas as aplicações que foram inicializadas. A SHE recebe o evento, monta a estrutura de dados e armazena no Catálogo de Gerenciamento. Com isso, a SHE passa a gerenciar todas as aplicações de Controle e Gerenciamento inicializadas pelo ABM, as quais são as aplicações de rede que compõem o ambiente.

Os resultados apresentados mostram uma comparação no tempo de inicialização das diversas aplicações de rede contidas no ambiente de experimentação. Esses valores servem de parâmetro de comparação para a recuperação ou cura que será apresentada ainda nesta subseção.

5.2.1.2 Avaliação de Coleta de Métricas

Os experimentos apontados na Seção 5.2.1.1 garantem a execução autônoma de uma aplicação para cada CoE. Após a inicialização, cada CoE aplica as técnicas de coleta propriamente desenvolvidas. Neste experimento, desenvolveu-se na MCoE serviços de ACL para coleta de informações da infraestrutura, ou seja, métricas de desempenho da

Camada de Infraestrutura, mas que contenham informações de Controle e Gerenciamento. O serviço de coleta básico nesse cenário é a busca por métricas na *engine* de *containers*, ou seja, coletar informações que indiquem o estado de todos os *containers* sendo executados. A MCoE não possui inteligência para avaliar se os *containers* são aplicações de Controle ou Gerenciamento, ou demais aplicações rodando na mesma infraestrutura – isso cabe à SHE.

Neste experimento, como o único servidor executando *containers* é o SONAr server_1, a MCoE requisita informações dos *containers* ao CIM que está em execução neste servidor. Outras instâncias de CIM poderiam estar executando, mas recomenda-se que o CIM cuide de buscar informações em todas as *engines*, de todos os servidores em execução, visto que ele é um orquestrador de *containers*. Para coletar informações, a MCoE faz requisições HTTP na API do CIM periodicamente. O CIM se conecta como um cliente Docker na *engine* em questão e busca informações de todos os *containers* em execução.

O papel da MCoE é coletar informações através do ACL, transformar as informações em eventos SONAr e publicá-los no NEM. O gráfico da Figura 25 mostra o tempo gasto para que a MCoE execute as etapas. A fase *collecting* é o tempo gasto para que a MCoE requisite informações e receba a resposta, enquanto a fase *publishing* é o tempo de transformação da métrica em evento e consequente publicação do mesmo no NEM. Nota-se que não há variação de tempo de coleta, visto que a MCoE busca as informações através da API do CIM. O CIM gasta alguns milissegundos para se conectar à *engine* de *containers* e buscar todos os *containers* criados e/ou em execução e retornar no formato JSON para a MCoE. Nota-se ainda que o tempo de transformação da métrica coletada em evento e publicação no NEM também não varia em cada ACL; este é um comportamento esperado, visto que as CoEs não possuem regras complexas e não fazem nenhuma análise das métricas coletadas (apenas as coletam e as enviam ao NEM). Por fim, ressalta-se que, previstamente, na primeira métrica coletada os tempos de *collecting* e *publishing* atingiram quase 200 ms – isto se deve à montagem e instanciação do cliente REST utilizado pela MCoE.

5.2.1.3 Comparando Técnicas de Recuperação

Os eventos de métricas coletados pela MCoE nos experimentos são recebidos por entidades que se subscreveram aos tópicos daquele tipo de evento. A SHE é uma dessas entidades e portanto recebe as métricas coletadas em forma de eventos SONAr. Ao receber um evento, a SHE é capaz de executar várias técnicas/funções idealizadas no Capítulo 4. A fase de monitoramento é executada para cada evento, enquanto a fase de recuperação é executada nos casos em que um evento identifique algum problema.

Para a manutenção das aplicações de rede, a SHE avalia constantemente se um evento não danifica alguma aplicação que está armazenada no Catálogo de Gerenciamento. Eventos contendo métricas são analisados pela SHE para identificação de possível degradação

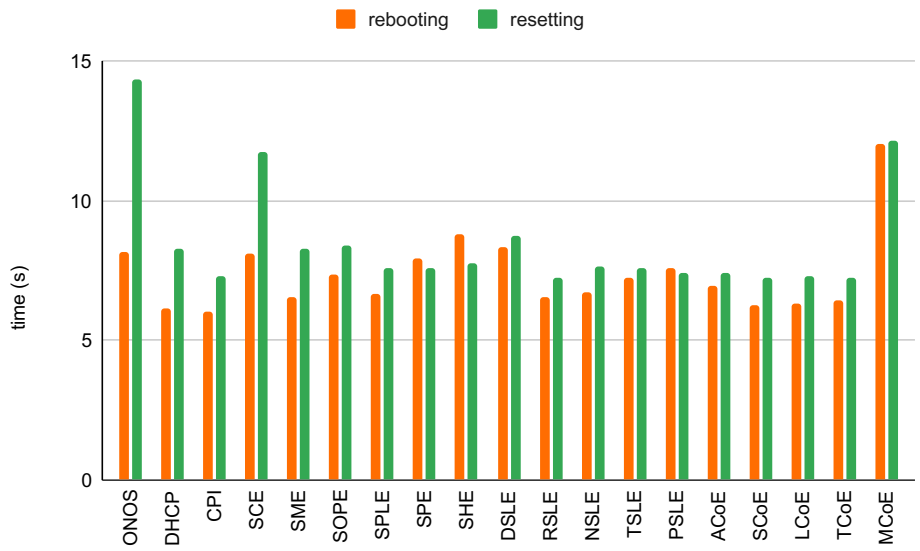


Figura 26 – Comparação de Técnicas de Reinicialização e Redefinição.

em alguma aplicação. A SHE recebeu centenas de eventos com métricas da saúde dos *containers* e analisou os eventos; não houve nenhuma ação de recuperação, visto que os *containers* estavam no estado `RUNNING`. Em seguida, iniciou-se a *fuzzy zone*, ou seja, um momento em que se forçou a degradação de *containers*.

Neste experimento, duas ações de degradação/quebra foram efetuadas, quais sejam: (i) forçou-se a parada de *containers* contendo as aplicações de rede inicializadas pelo ABM – o *container* não foi destruído, mas somente parado ou pausado em algumas situações; e (ii) os *containers* foram destruídos da *engine*.

A análise aplicada pela SHE significa buscar todas as aplicações de rede no Catálogo de Gerenciamento e, para cada uma, verificar se o identificador do *container* está contido no evento recebido. Em caso positivo, a SHE verifica a informação de *status* do *container* e, caso o *status* não seja `RUNNING`, a SHE identifica uma degradação; em caso negativo, a SHE identifica um problema, visto que se as informações do *container* não foram recuperadas, ele deixou de existir.

A ação de parada/pausa de um *container* faz com que a SHE utilize a técnica de recuperação por reinicialização – *reboot*. A ação de destruição de um *container* faz com que a SHE utilize a técnica de recuperação por redefinição – *reset*. Essas duas técnicas foram escolhidas para demonstrar a SHE aplicando duas diferentes funções de recuperação. Em ambos os casos, a SHE monta as requisições de recuperação baseando-se nas informações de *command* contidas no Catálogo e requisita ao CIM.

O experimento foi repetido diversas vezes, mas os resultados não tiveram variação significativa. A Figura 26 contém os resultados de um *snapshot* e permite comparar as duas técnicas de recuperação aplicadas. Nota-se que, conforme esperado, na maioria dos

casos a técnica de reinicialização é mais rápida do que a técnica de redefinição. Isto se dá pois a reinicialização apenas solicita a volta de um *container*, que já estava previamente criado, através de sua inicialização, enquanto que a redefinição precisa buscar uma imagem, instanciar variáveis de ambiente, criar o *container* e logo após iniciá-lo.

Duas aplicações na Figura 26 chamam a atenção. São elas, a SHE e a MCoE. A atenção está no fato de que o leitor pode se perguntar 'como a SHE se cura?', visto que ela está em degradação, ou como o evento informando a degradação chega no NOSH, visto que a MCoE está degradada.

A recuperação da SHE é feita através de outra instância SHE. Portanto, neste experimento, foram executadas duas aplicações SHE, através do ABM, ou seja, dois *containers* distintos foram iniciados com a imagem Docker da SHE. A primeira aplicação SHE teve seu *container* parado e depois destruído, enquanto a segunda aplicação SHE recuperou o *container*. Para que não haja conflito nas ações de recuperação, é importante que a SHE bloqueie ações em um componente específico do Catálogo e que o CIM seja capaz de lidar com requisições sobrepostas de reinicialização/redefinição. O Apêndice A contém detalhes de como a SHE e CIM foram desenvolvidos nesta tese.

A recuperação da MCoE só é possível, pois no experimento, o ABM executou duas aplicações MCoE. Dessa forma, as duas coletam métricas concomitantemente e publicam os eventos dessas métricas. Não há detalhes de implementação para isso, visto que não há impacto relevante para SOEs, como a SHE, receber eventos com o mesmo tipo de informação. De fato, isso acontece, pois a MCoE pode estar aplicando ACL com uma periodicidade curta, por exemplo a cada 1 segundo, enquanto a SHE pode levar mais de 1 segundo pra recuperar algo. Dessa forma, o bloqueio de componente de Catálogo é basilar em implementações da SHE.

Recomenda-se que as implementações de CIM sejam capazes de executar mais de uma instância para as aplicações acuradas, como SHE e MCoE. Isto justifica a implementação de um CIM da própria SONAr, em detrimento da utilização de orquestradores terceiros. Apesar de diversos desses orquestradores possuírem funções de autocura, como é o caso do Kubernetes, é importante que o orquestrador saiba quais aplicações são capazes de possuir mais de uma instância. Em outras implementações, pode-se permitir que o ABM e CIM de fato só criem 1 instância para a aplicação, desde que a SHE armazene comandos para recriar a instância em questão; por exemplo, a SHE pode possuir uma função para recriar MCoE caso não receba nenhuma métrica de MCoE durante algum tempo.

5.2.1.4 Comparando Monitoramento e Recuperação

Conforme exposto no Capítulo 4, a cura se dá através das fases complementares de monitoramento e recuperação. Todavia, enfatiza-se que essas fases não são sequenciais como pode dar a entender, visto que é monitorada a rede primeiramente, para em seguida se tomar alguma ação de recuperação. A fase de monitoramento é feita o tempo todo

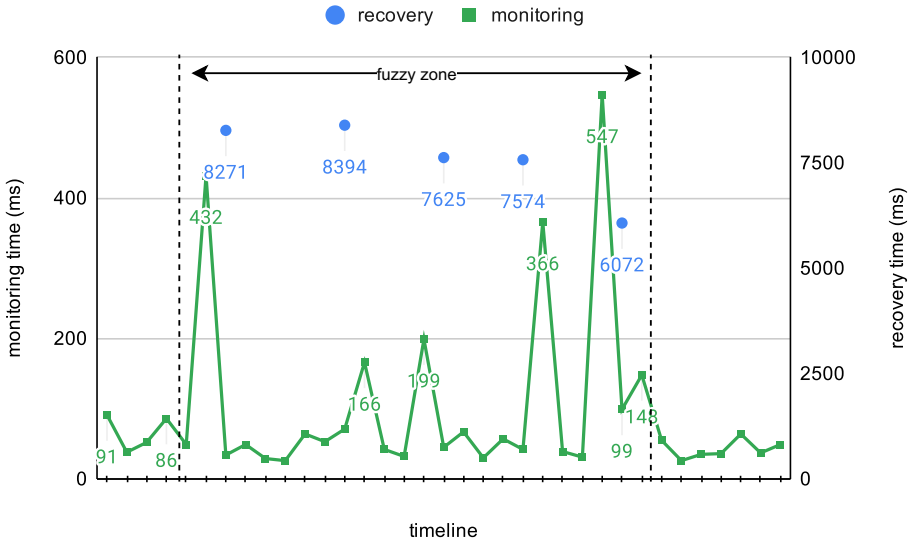


Figura 27 – Avaliação do Desempenho de Monitoramento.

e uma computação de monitoramento só termina quando a computação de recuperação foi iniciada e finalizada. Por exemplo, para recuperar um *container*, a fase de monitoramento identificou uma degradação/quebra (falta/falha/problema/estado desconhecido) e bloqueou o componente para que a próxima computação de monitoramento (análise do próximo evento recebido) não atue na recuperação do mesmo componente. Ao finalizar a recuperação, a fase de monitoramento libera o recurso (bloqueio de componente).

Para avaliar o monitoramento, e não somente recuperação, os experimentos realizados foram medidos desde a chegada de um evento na SHE até a finalização das funções/serviços que escutavam por aquele evento, ou seja, mediu-se os tempos de monitoramento e recuperação completos. Um *snapshot* dos resultados está apresentado na Figura 27. A linha do tempo – *timeline* – da figura indica um certo tempo de coleta de medição, ou seja, alguns eventos chegando na SHE. A *fuzzy zone* trata-se de um período de degradação, no qual os *containers* das SOEs estavam sendo destruídos, obrigando a SHE a tomar ações de redefinição.

A Figura 27 evidencia que os tempos de monitoramento fora de *fuzzy zone* (antes e depois) são relativamente baixos. Isso acontece porque a análise dos componentes armazenados no Catálogo não é complexa (trata-se de avaliar cada componente e comparar o *status* da métrica recebida). Quando há uma ação de recuperação, o tempo de monitoramento aumenta de forma diretamente proporcional. Os dois eixos verticais da figura permitem comparar o aumento do tempo de monitoramento (eixo vertical esquerdo) enquanto se compara o tempo de recuperação (eixo vertical direito).

Os experimentos desta subseção permitiram demonstrar algumas funções do NOSH em execução. Os experimentos focam em aplicações dos planos de Controle e Gerenciamento,

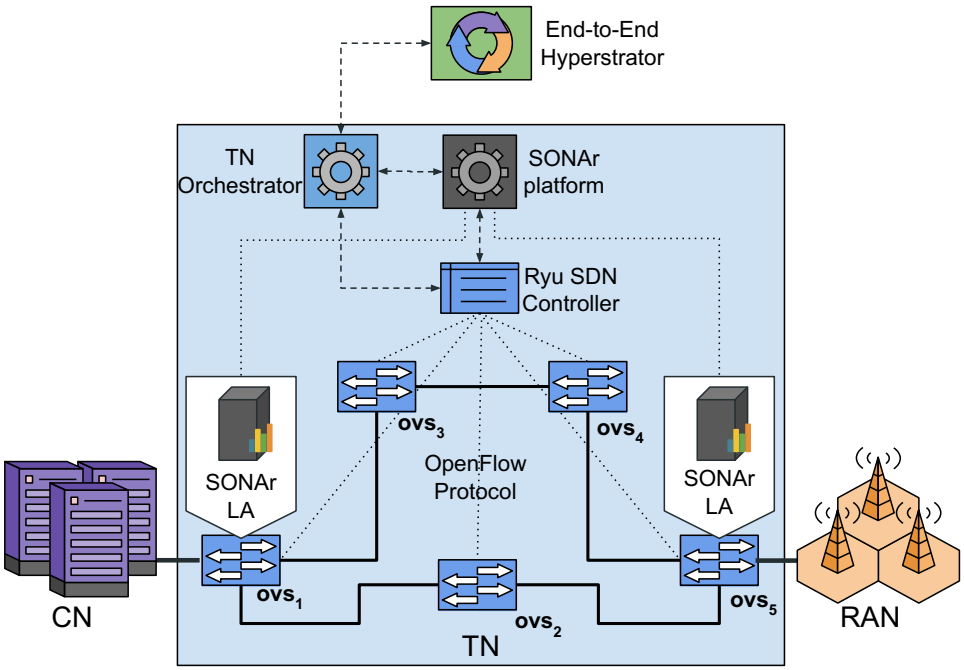


Figura 28 – Topologia de Rede para Experimentos de QoS.

para mostrar a interoperabilidade dentro do *framework*. Para avaliar o plano de Dados, ou seja, percepção de aplicações cliente/servidor, experimentos em uma rede E2E foram realizados, conforme descrito na Subseção 5.2.2.

5.2.2 Autocura em Redes Fim-a-Fim

Para avaliar o *framework* NOSH atuando como ferramenta OAM em uma rede E2E, configurou-se um ambiente de experimentação com elementos arquiteturais de SDN e NFV para orquestração de *Network Slices* (NSs). O ambiente em questão faz parte do *testbed* IRIS (IRIS, 2021). Estabeleceu-se um domínio E2E com três segmentos de rede, quais sejam CN, TN e RAN. A orquestração de NSs fim a fim fica a cargo do HOEN (SANTOS et al., 2018). Nesse esquema de orquestração, um super-orquestrador de redes denominado 'Hyperstrator' é responsável por gerenciar a instanciação de NSs na rede. O *Hyperstrator* garante a consistência do aprovisionamento de NSs, mas deixa que orquestradores de cada segmento de rede cuidem do aprovisionamento naquele segmento. Dessa forma, particularidades do segmento são melhor tratadas.

O NOSH foi implantado no TN para servir como ferramenta OAM. Dessa forma, um orquestrador cuida da fase de aprovisionamento de NSs, enquanto o NOSH garante a saúde da rede durante a fase de execução. A Figura 28 apresenta o ambiente de experimentação utilizado para os experimentos descritos nesta subseção. No segmento TN, observa-se a presença de um orquestrador 'TN Orchestrator' que recebe requisições de criação/deleção de NSs do *hyperstrator*, que por sua vez recebe requisições de SPs. O

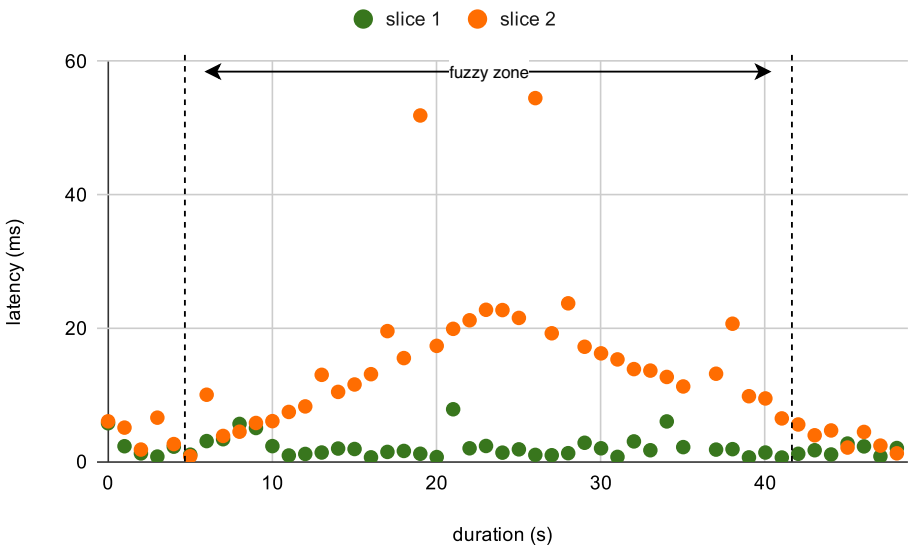


Figura 29 – Resiliência em URLLC com Detecção de Anomalias.

TN Orchestrator interage com um controlador SDN para aprovisionamento de recursos de rede (por exemplo fluxos de encaminhamento) que suportem os requisitos de QoS demandados pelo SP que solicitou o aprovisionamento do NS.

Uma vez que o TN Orchestrator define os recursos computacionais a serem criados, as rotas para encaminhamento de primitivas de dados são criadas pelo controlador SDN. No *setup* do IRIS, utilizou-se o controlador Ryu (Ryu SDN Framework Community, 2021) por ser o controlador já utilizado pela equipe do HOEN e para o qual o TN Orchestrator já possuía integração. Ressalta-se que a tecnologia é indiferente para a SONAr (qualquer controlador que implemente um protocolo de SBI é adequado) e que essa escolha não interfere nos propósitos dos experimentos desta seção. O Ryu está integrado com NEs, que na prática são OVSs. Portanto o protocolo SBI neste experimento é o OpenFlow.

As entidades da SONAr são aplicações desenvolvidas em Python e implantadas na SONAr *Platform* da figura – alguns detalhes de implementação estão descritos no Apêndice B. No *setup*, dois LAs foram implantados (um em cada NE de borda). A ideia do NOSH como OAM no ambiente HOEN é garantir que NSs estejam saudáveis durante o tempo de execução. Dessa forma, o TN Orchestrator garante aprovisionamento com a melhor configuração de recursos, enquanto o NOSH monitora e recupera, através de reconfigurações de recursos, os NSs baseando-se em QoS pré-estabelecidos. Para os experimentos aqui relatados, testou-se a comunicação de NSs de URLLC. Portanto, os LAs foram implantados em NEs de borda para coletar métricas de desempenho de multicaminhos no segmento TN.

5.2.2.1 Medindo Latência Fim-a-Fim

O experimento em questão procura comparar o aprovisionamento de um NS sem requisitos de QoS com um NS com requisitos de QoS de baixa latência. Primeiramente, o Hyperstrator aprovisiona os dois NSs: $slice_1$ possui o requisito de 5 ms de latência e o $slice_2$ não possui nenhum requisito. O Hyperstrator solicita a criação à CN e à RAN (fora do escopo deste trabalho) e finalmente à TN, para conectar os dois outros segmentos. O TN Orchestrator configura as filas nos NEs e define as melhores rotas, ou seja, aquelas que atendam 5 ms de latência. Nos experimentos realizados, o TN Orchestrator sempre escolheu as rotas com menores saltos. Portanto, a rota inicial é ovs_1–ovs_2–ovs_5 – a latência inicial sempre é baixa em ambas as rotas do *setup*.

Os LAs foram configurados autonomamente pelo NOSH, que atribuiu os fluxos e endereços mostrados previamente na Figura 15. Cada LA envia informações de desempenho (como RTT) de todos os caminhos entre CN e RAN – conforme procedimento descrito na Seção 4.4. A SONAr *Platform* (contendo o NOSH) avalia as métricas recebidas periodicamente, conforme procedimento explanado nas Seções 4.4 e 4.5. Nos experimentos desta seção, decidiu-se medir a latência nas pontas, visto que o intuito é avaliar a SONAr em uma rede E2E. Para isso, subiu-se uma aplicação servidor em um *container* no segmento CN e aplicações cliente na RAN em um notebook. Os clientes solicitam informações ao servidor periodicamente.

No atual experimento, forçamos um *delay* no ovs_2, aumentando em 1 ms a cada segundo durante 20 segundos e posteriormente diminuindo em 1 ms a cada segundo durante 20 segundos. Enfatiza-se que, em avaliações de sistemas com autocura, utiliza-se o termo *fuzzy zone* como o período de degradação da saúde da rede/sistema. A Figura 29 apresenta as métricas de latência coletadas em dois clientes: um cliente se comunica através do $slice_1$ enquanto o outro através do $slice_2$. Nota-se que o cliente do $slice_1$ não é afetado pela degradação do ovs_2. Isto ocorre pois o NOSH monitora os multi-caminhos e, ao identificar o aumento de *delay* no caminho ovs_1–ovs_2–ovs_5, interage com o TN Orchestrator para buscar quais NSs utilizam o caminho ovs_1–ovs_2–ovs_5. O NOSH reconfigura os recursos, ou seja, define novas rotas que atendam ao requisito de latência de 5 ms de $slice_1$. Neste caso, o NOSH reconfigurou a rota do $slice_1$ para ovs_1–ovs_3–ovs_4–ovs_5. Em contrapartida, o $slice_2$ não possui requisitos de latência e sofre com a degradação de ovs_2 de forma diretamente proporcional ao aumento e diminuição de *delay* de ovs_2 durante a *fuzzy zone*, conforme esperado.

5.2.2.2 Comparando Latência Fim-a-Fim com Diferentes Requisitos de QoS

Para este experimento, foi utilizada a mesma configuração descrita no experimento anterior, com o mesmo *delay* forçado no ovs_2, porém foram utilizadas três aplicações clientes e aprovisionados três NSs. Os NSs criados com seus respectivos requisitos de

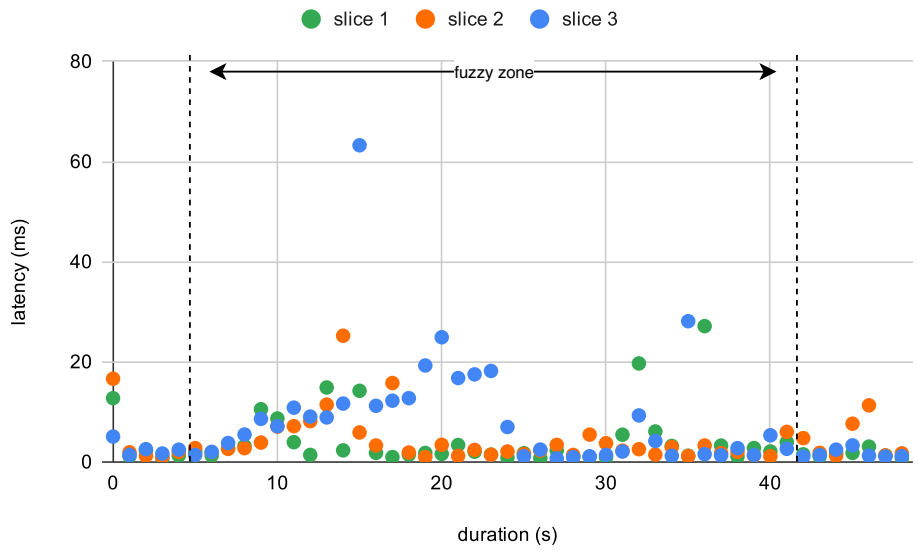


Figura 30 – Comparação de Recuperação de NSs com Diferentes Requisitos de QoS.

QoS (latência máxima) foram: slice_1 com 5 ms, slice_2 com 10 ms e slice_3 com 20 ms. O intuito deste experimento é demonstrar o momento que o NOSH, através da SHE, recuperou cada NS. A Figura 30 mostra a latência dos três NSs durante a *fuzzy zone*. Observa-se que cada NS teve sua latência normalizada (atendendo aos requisitos de QoS), alguns poucos segundos após a latência medida no caminho ovs_1–ovs_2–ovs_5 ter passado da latência requerida. Por exemplo, o slice_1 foi recuperado primeiro pois, a partir de 10 segundos de experimento, seu QoS não estava mais sendo atendido; analogamente para slice_2 e slice_3.

O experimento demonstra a SHE em funcionamento para NSs com diferentes requisitos de QoS, porém deixa nítido que a identificação de anomalias por detecção não é eficaz, sendo algumas vezes necessária a identificação por predição. O SLA dos NSs não foi atendido durante alguns segundos. Isso se dá pelo tempo gasto para que os LAs coletem informações, enviem-nas para o NOSH, a SHE analise as informações e identifique rotas com degradação de latência, o NOSH interaja com o TN Orchestrator para verificar quais NSs vão ter problemas de QoS (quais utilizam as rotas degradadas), o NOSH recalcule novas rotas, e finalmente, o NOSH interaja com o controlador Ryu para aplicar os fluxos de encaminhamento das novas rotas.

5.2.2.3 Comparando Recuperação em Tempo Real e Técnica *Buckets*

Para melhorar o tempo de recuperação de NSs, a técnica *Buckets* pode ser experimentada. Para isto, os mesmos três NSs utilizados no experimento anterior foram aprovisionados. Com a SHE aplicando a técnica *Buckets*, espera-se que, ao detectar a degradação de latência no slice_1, a SHE recupere também o slice_2, o qual pertence ao mesmo *bucket*

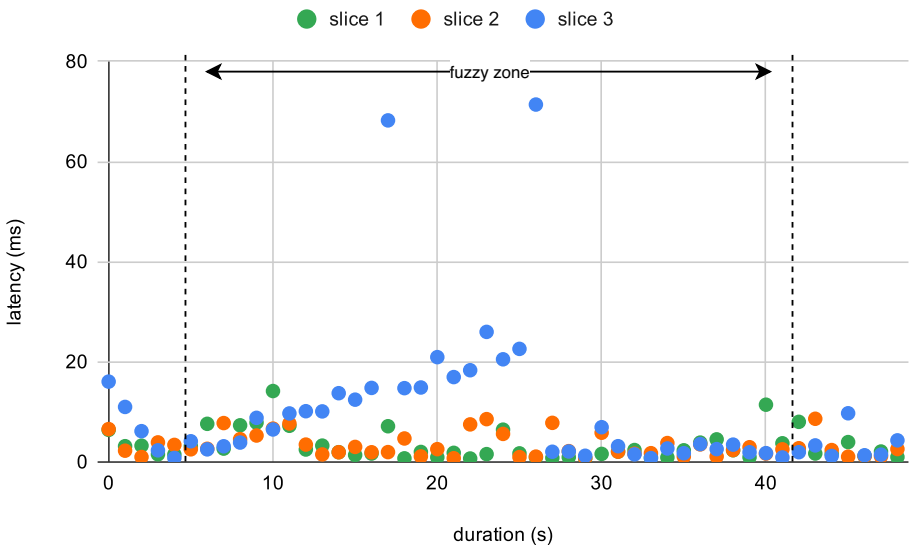


Figura 31 – Comparação de Recuperação em Tempo Real *vs* Técnica de *Buckets*.

de slice_1 (1–10). A Figura 31 demonstra essa técnica em ação, na qual slice_1 e slice_2 foram recuperados simultaneamente. O slice_3 apresentou degradação de latência durante mais tempo e portanto só foi recuperado quando o NOSH identificou a degradação de um NS no seu *bucket* (10–30).

A técnica de *Buckets* possui como objetivo a recuperação prévia de NSs ou outros recursos quando está claro que esses NSs vão ser afetados pelo problema de desempenho em breve. No protótipo experimentado aqui, a utilização de três NSs demonstra que houve recuperação, mas ressalte-se que o ganho maior está quando centenas de NSs estão rodando: como a recuperação depende de reconfiguração de recursos, a recuperação prévia pode garantir que não hajam atrasos que interfiram no QoS dos NSs.

5.2.2.4 Comparando Detecção e Predição

A técnica de *Buckets* é eficaz para recuperar previamente NSs que serão fatalmente afetados pelos problemas de degradação de desempenho. Porém, a técnica faz parte da fase de recuperação e não é eficaz para prever problemas. A predição está contida na fase de monitoramento, ou seja, o problema é identificado antes que ele ocorra e não apenas se recupera previamente. Para experimentar a identificação de falhas por predição, nitidamente recomendável para URLLC, mediu-se o desempenho (em latência) de dois clientes, sendo que o primeiro foi experimentado com a PSLE executando enquanto o segundo teve a degradação identificada pela SHE (a PSLE foi desabilitada).

A Figura 32 apresenta o desempenho de dois clientes que comunicam com seus servidores através de NSs com requisitos de QoS de 20 ms. Nota-se que quando o *framework* utiliza técnicas de predição, o aumento do *delay* de ovs_2 é identificado previamente e o

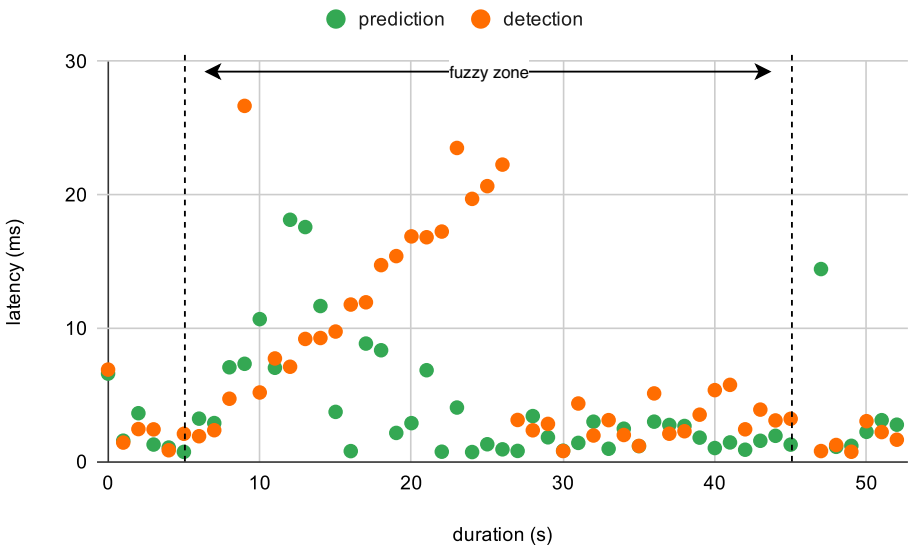


Figura 32 – Comparação de Identificação de Falhas com Detecção e Predição.

cliente não mede latência acima de 20 ms. Isso se deu pois a predição fez com que o NOSH reconfigurasse as rotas previamente, antes que a latência requerida pelo NS fosse ultrapassada. Todavia, sem a PSLE executando nota-se que o cliente possui sua latência normalizada apenas após 25 segundos de experimento; até 25 segundos, observa-se que a latência medida no cliente aumenta de forma diretamente proporcional ao aumento de *delay* do ovs_2. Ressalta-se que para este experimento o microserviço de predição desenvolvido na PSLE analisa as últimas 5 métricas recebidas e utiliza aritmética para definir em que momento a latência vai afetar um ou mais NSs.

5.2.2.5 Desempenho da Inicialização Autônoma

Os experimentos desta seção utilizaram os LAs como forma de coleta de métricas de desempenho nos NEs de borda. Os LAs foram inicializados autonomamente conforme procedimento descrito na Seção 4.3. Para avaliar esta inicialização, que é uma contribuição desta tese, a Figura 33 mostra os tempos gastos em cada fase da inicialização autônoma. As fases são descritas abaixo:

- *information*: esta fase compreende o recebimento da primitiva `config_req` do LA_1 e as funções executadas pelo NOSH para buscar informações dos elementos de rede, topologias e demais LAs armazenadas no NDB e buscar informações relevantes no orquestrador do segmento;

- *management*: nesta fase o NOSH calcula todas as rotas entre o novo LA e os demais LAs da topologia e constrói as regras de fluxos que serão aplicadas para que o novo

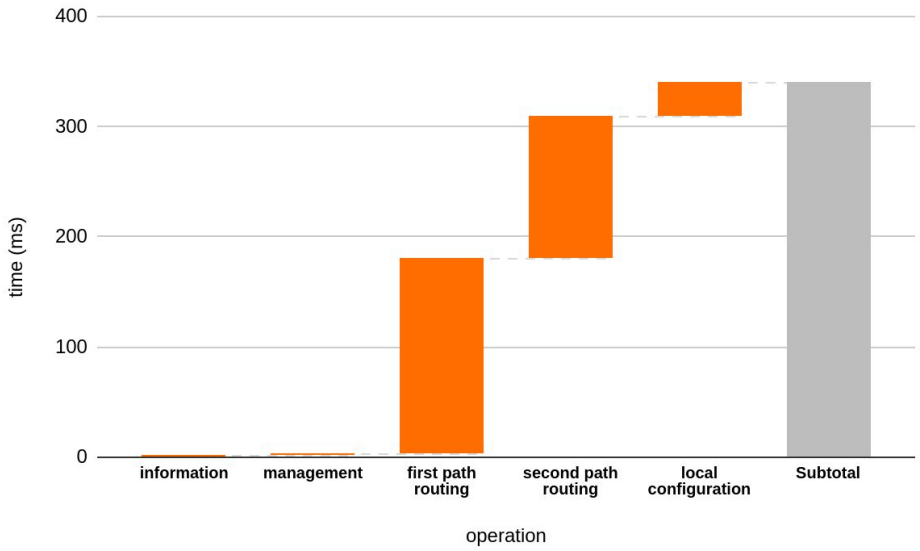


Figura 33 – Tempo Gasto nas Operações para Estabelecimento Autônomo de *Slices* de Gerenciamento.

LA se comunique com os demais através das diversas rotas. Também há a definição de endereços IP para as interfaces virtuais do novo LA;

- *first path routing*: fase em que o NOSH requisita e aguarda o controlador Ryu aplicar as regras de encaminhamento nos NEs da primeira rota, são eles ovs_1, ovs_2 e ovs_5;

- *second path routing*: fase em que o NOSH requisita e aguarda o controlador Ryu aplicar as regras de encaminhamento nos NEs da segunda rota possível, são eles ovs_1, ovs_3, ovs_4 e ovs_5;

- *local configuration*: fase em que o novo LA recebe a primitiva `config_resp` e aplica localmente as configurações de interfaces virtuais com suas respectivos endereços IP e rotas estáticas.

Os tempos reportados na Figura 33 indicam que as operações de *information* e *management* são executadas em poucos milissegundos. Isso é esperado pois as operações possuem basicamente funções algorítmicas. O tempo gasto para *local configuration* também é aceitável e depende da quantidade de configurações locais que serão executadas pelo LA. Os tempos das operações de *first path routing* e *second path routing* não são tão baixos como os demais, porém ressalta-se que essas operações são executadas pelo controlador SDN Ryu nos NEs, ou seja, em cada OVS. Em repetições do experimento, notou-se uma variação de 100 até 300 ms para que o Ryu aplicasse as modificações de fluxo nos OVSs.

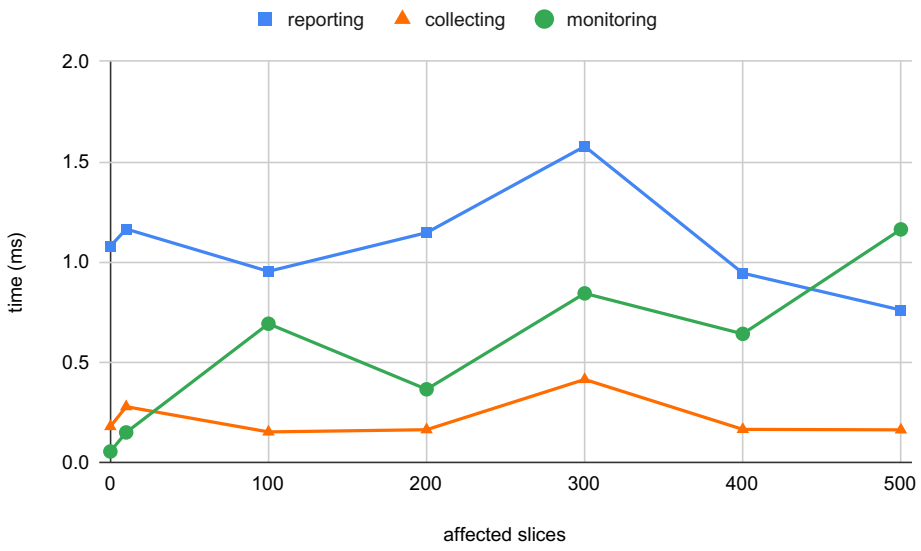


Figura 34 – Desempenho de Monitoramento.

5.2.2.6 Desempenho da Fase de Monitoramento

Aproveitou-se o *setup* utilizado nesta subseção para avaliar se os tempos gastos pelo NOSH nas fases de monitoramento e recuperação são adequados. Para isso, solicitou-se a criação de vários NSs para o TN Orchestrator e, portanto, o NOSH passou a cuidar da saúde desses vários NSs. A fase de monitoramento foi medida considerando: (i) etapa de *reporting*, que é o tempo gasto desde que uma métrica de desempenho é coletada pelos LAs até que ela chegue nas CoEs; (ii) etapa de *collecting*, que é o tempo gasto para que a métrica seja internalizada pela CoE para dentro do NOSH (SONAr *Platform*); e (iii) etapa de *monitoring*, que é o tempo de monitoramento propriamente dito, ou seja, o tempo gasto pela SHE ou PSLE para analisar a métrica coletada e identificar problemas.

A Figura 34 mostra os tempos reais gastos em cada etapa de monitoramento para centenas de NSs. O número de NSs aprovisionados é maior, porém a figura apresenta a quantidade de NSs que foram afetados por um problema de desempenho. Nota-se que não há crescimento em *reporting* ou *collecting*. Porém, o tempo de *monitoring* possui um pequeno crescimento, menor que 1 ms, a cada 100 NSs.

5.2.2.7 Desempenho da Fase de Recuperação

Os vários NSs afetados no experimento anterior foram autonomamente recuperados, isto é, tiveram suas rotas no segmento TN reconfiguradas pelo NOSH. O tempo de recuperação está apresentado na Figura 35. Diferentemente da fase de monitoramento, a fase de recuperação gasta mais tempo quando mais NSs estão afetados. Conforme esperado, a fase de monitoramento é constante para diferentes números de NSs pois a análise é feita

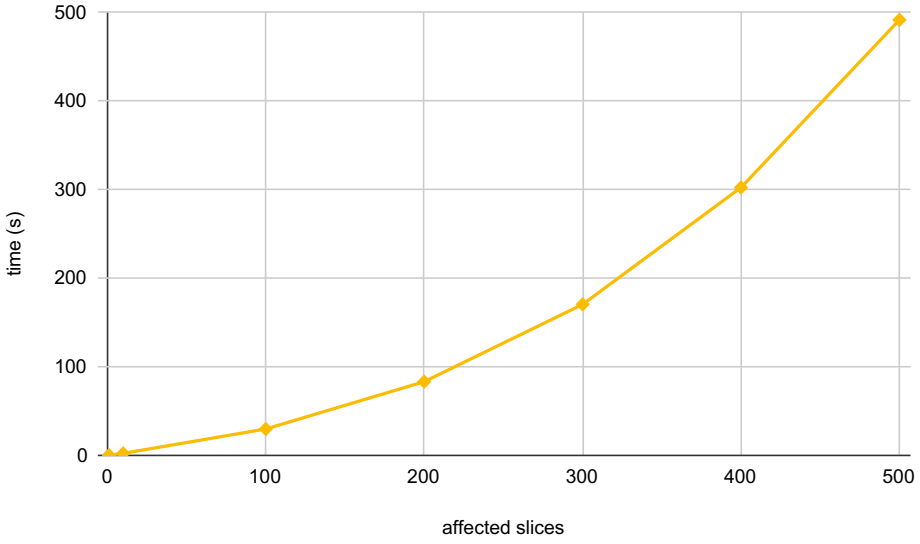


Figura 35 – Desempenho de Recuperação.

em cima dos recursos de redes ou computacionais, no caso dos experimentos aqui realizados os recursos são caminhos (rotas no segmento TN). Na fase de recuperação observa-se um crescimento cúbico no tempo gasto, o que pode ser explicado pela ação individual necessária em cada NS: para cada NS, há reconfiguração de fluxos de encaminhamento nos NEs. A montagem dos comandos pelo NOSH gastou um tempo irrelevante, porém a aplicação das regras pelo Ryu nos OVSs gastou um tempo relevante conforme mostra a Figura 35.

Ressalta-se que os experimentos das Figuras 34 e 35 são os mesmos. O tempo para aplicação das regras pelo Ryu nesses experimentos teve uma média de 200 a 800 ms. Esse desempenho pode ser melhorado de diversas maneiras. As mais intuitivas são: a implementação no NOSH de recuperação *multi-task*, de forma que as requisições do NOSH para o controlador sejam feitas de forma paralela; ou a utilização de outros controladores SDN, os quais já possuem funções assíncronas para aplicar modificações de regras em NEs.

5.3 Avaliação dos Resultados

Os resultados alcançados nos experimentos descritos na Seção 5.2 mostram a interoperabilidade e eficácia do NOSH para cura e autocura. Demonstrou-se a autocura do NOSH, a cura de aplicações de rede dos planos de Controle e Gerenciamento, e por fim a cura de NSs que fatalmente tinham seus recursos (caminhos) afetados por degradações no ambiente.

A inicialização autônoma do NOSH foi modificada para prover as funcionalidades do

Catálogo de Gerenciamento. Pelos experimentos, a inicialização depende da quantidade de aplicações de rede ou entidades SONAr que o ABM execute. Apenas o NDB e a SHE possuem particularidades e, esperadamente, fizeram com que o ABM aguardasse a execução não somente dos *containers*, mas também dos serviços do NDB e SHE. Ao estar no estado de execução, a coleta de métricas é executada e não há grande variação nos tempos de coleta de informações e consequente publicação dessas informações no formato de eventos SONAr. Isso prova a eficácia da separação de entidades coletoras das demais entidades: as CoEs possuem como função principal a coleta e transformação das informações coletadas em eventos.

Optou-se pela avaliação de duas técnicas de recuperação previstas no NOSH: reinicialização e redefinição. Era de se esperar que a primeira técnica executasse a recuperação levemente mais rápido que a segunda. De fato, a reinicialização só reexecuta um serviço que havia sido previamente estabelecido. Por outro lado, na redefinição, o serviço (aplicação) deve ser construído novamente e há um tempo extra para isso. Outras técnicas, como migração e reconfiguração de recursos, são previstas no NOSH e dependem exclusivamente de implementação. Além disso, a técnica de redefinição apenas recriou um *container* a partir de uma imagem já alocada no servidor que roda a *engine*. Pode-se em implementações futuras baixar a imagem de algum repositório externo.

Além dos resultados supradescritos, que focam em aplicações do plano de Controle e Gerenciamento, alguns resultados promissores foram obtidos ao se testar o NOSH como ferramenta OAM de uma rede E2E. A interoperabilidade do NOSH com orquestradores de NSs demonstra a eficácia de integração desse *framework* com controladores e orquestradores NFV.

Como era de se esperar, as técnicas de monitoramento por predição são recomendadas para determinados tipos de NSs, tais como aqueles utilizados em URLLC. Todavia, ressalte-se que não se pode descartar a técnica de detecção, uma vez que a predição pode não acontecer e, portanto, o ambiente eventualmente ficaria degradado.

As técnicas de recuperação em tempo real e *Bucket* mostram a flexibilidade do *framework* para aplicar ações de recuperação com diferentes aspectos. Ainda, os resultados obtidos, ao se estressar o ambiente com centenas de NSs afetados por um problema, mostram que a fase de monitoramento é rápida independentemente da quantidade de NSs. Isso se explica pois a avaliação não é em cima dos NSs, mas sim das rotas que são limitadas. Todavia, o tempo de recuperação aumentou consideravelmente na medida em que o número de NSs afetados também aumentou. Pode-se considerar que essa demora para a recuperação de vários NSs é uma limitação atual do *framework* e requer estudo futuro.

Pode-se afirmar que os resultados obtidos são no geral satisfatórios. Porém, experimentos exaustivos das várias funções do NOSH e avaliação de outras entidades (e não somente SHE e PSLE) são bem vindas. O Capítulo 6 traz algumas observações finais acerca desta pesquisa e lista os trabalhos futuros esperados na SONAr e no NOSH.

Capítulo 6

Conclusão

Este capítulo apresenta o fechamento desta tese, descrevendo as principais contribuições obtidas ao se propor um *framework* para introdução do fundamento de autocura em abordagens SDN e NFV. Ainda, discute-se os próximos passos em relação à evolução da arquitetura SONAr e do *framework* OAM entregue.

Para se atingir o objetivo geral desta tese – incorporar aspectos de autocura na arquitetura SONAr – percebeu-se que era necessário um aprofundamento do fundamento autocura nas tecnologias SDN e NFV. Isso se deu porque tais tecnologias introduzem novos planos que os trabalhos de autogerenciamento anteriores não preveem. Portanto, observou-se que uma análise completa era necessária para introduzir autocura seja na SONAr, seja em qualquer outra proposta de autogerenciamento de redes de computadores e redes móveis.

No Capítulo 2 foi apresentado o cenário de redes de computadores e móveis da atualidade, mergulhando em temas significativos como SDN, NFV e SON. Além disso, apresentou-se os principais trabalhos de Gerenciamento de Redes com vistas à computação autônoma. Ainda no capítulo, foi feita uma revisão dos planos de Dados, Controle e Gerenciamento, visando não suas vantagens, mas suas desvantagens principalmente no que tange funções de gerenciamento.

As desvantagens apresentadas no Capítulo 2 estão intrinsecamente relacionadas à separação do controle da rede, visto que isso traz novos elementos para serem gerenciados e por conseguinte maior complexidade. Para resolver o gerenciamento de redes nesses tipos de ambientes, optou-se pelo projeto de uma arquitetura de autogerenciamento para redes. Nesta arquitetura, SONAr, fundamentos de computação autônoma da engenharia de software e já empregados em SON foram empregados e a arquitetura garante a interoperabilidade entre eles.

A arquitetura em questão foi brevemente descrita no Capítulo 3. O intuito não é que este seja um trabalho definitivo, mas que entregue uma arquitetura de autogerenciamento de propósito geral, onde novas funções de gerenciamento possam ser introduzidas posteriormente sob a mesma arquitetura. Durante o desenvolvimento desta tese, duas

implementações da SONAr foram realizadas e também estão descritas no Capítulo 3. Ressalta-se que a primeira implementação focou em aspectos de autogerenciamento dos planos de controle e gerenciamento, enquanto a segunda focou em manutenção de qualidade de serviço em tempo de execução.

Cada fundamento de computação autônoma possui suas particularidades e necessita análise acentuada. Neste trabalho, optou-se pelo aprofundamento no fundamento de autocura devido a sua relevância na manutenção de planos de comunicação e QoS durante o tempo de execução. Como diversos outros trabalhos de alocação e configuração de recursos são encontrados para a fase de aprovisionamento, entende-se que esforços para a fase de tempo de execução são indispensáveis. Para aprofundar em autocura, decidiu-se criar um *framework* baseado na SONAr e relatar os fatores que são necessários em um *framework* que trabalhe com autocura; dessa forma, passa-se pelo fundamento de autocura de forma efetiva.

O Capítulo 4 descreve os novos conceitos idealizados nesta tese para criação do *framework* (NOSH), como as redes lógicas de controle e gerenciamento através dos *Slices* de Gerenciamento, o Catálogo de Gerenciamento, a inicialização autônoma do *framework* e as funções de monitoramento e recuperação de falhas que fazem com que a SONAr seja capaz de curar sintomas degradantes nos três planos, incluindo o plano no qual ela mesma está inserida, o que torna a arquitetura capaz de realizar 'autocura' no aspecto filosófico da expressão, e não apenas 'cura'.

Diversos experimentos foram abordados no Capítulo 5 para avaliar o *framework* proposto em funcionamento. Os diversos cenários experimentados mostram que a interoperabilidade da SONAr é eficaz para tratar diversos tipos de problemas, e, portanto, pode-se aferir que outros cenários podem ser atendidos futuramente sem alterar a arquitetura na qual os serviços serão executados. Pode-se afirmar que os tempos medidos nos experimentos são adequados em relação ao que se encontra na literatura em SDN e NFV.

Os objetivos específicos descritos na Seção 1.2 foram alcançados ao longo dos Capítulos 2, 3, 4 e 5. No Capítulo 4, a especificação da SONAr resolveu o objetivo de projetar componentes e interoperabilidade para autocura. As duas implementações descritas no Capítulo 3 e posteriormente experimentadas no Capítulo 5 tratam o objetivo de implementar serviços de monitoramento em diversas abordagens e planos. Os objetivos de aplicar e avaliar ações de monitoramento e recuperação foram atingidos através da proposta desta tese, descrita no Capítulo 4. Tais ações foram realizadas também de forma prática nos experimentos das Seções 5.2.1 e 5.2.2, de forma a tratar o objetivo de demonstrar a solução proposta.

Entende-se que a análise completa do fundamento de autocura e a proposta de um novo *framework* permitiu que o objetivo principal da tese – incorporar aspectos de autocura na arquitetura SONAr, para garantir o funcionamento de redes SDN/NFV, permitindo que falhas nos planos de Controle, Dados e Gerenciamento possam ser resolvidas com a mínima

intervenção humana – tenha sido plenamente atingido. Isto traz diversas contribuições para a área de pesquisa, incluindo pesquisas aplicadas, as quais estão detalhadas na Seção 6.1. A arquitetura SONAr e o *framework* NOSH ainda possuem muitas aberturas para investigações. Portanto, a Seção 6.2 descreve sugestões de trabalhos futuros no projeto SONAr como um todo. Diversas contribuições em produção bibliográficas foram possíveis a partir deste trabalho, e estão listadas na Seção 6.3.

6.1 Principais Contribuições

O trabalho apresentou uma análise do fundamento de autocura nos planos de Dados, Controle e Gerenciamento em redes de computadores e móveis que porventura utilizem SDN e NFV como tecnologias base. A contribuição geral consiste do levantamento das desvantagens correntes em SDN e NFV do ponto de vista de gerenciamento e da proposta de uma arquitetura e *framework* para resolver o gerenciamento nessas redes. O *framework* foi proposto visando a implementação de uma ferramenta OAM fortemente sustentada por conceitos de autogerenciamento.

A fundamentação teórica apresentada neste trabalho é uma contribuição para a área de redes de computadores e móveis pois descompôs desvantagens de SDN e NFV que, via de regra, não são abordadas nos trabalhos da área. Novas tecnologias em redes são exploradas veemente, mas a estrutura arquitetural dessas tecnologias não é projetada considerando características de gerenciamento. De fato, gerenciamento em redes de computadores é historicamente tratado como uma atualização ou *patch* implantado posteriormente. A análise das falhas associadas aos planos de Dados com seus QoS, Controle e Gerenciamento fica como uma contribuição e foi publicada em um periódico conforme será descrito na Seção 6.3.

De modo mais específico, pode-se afirmar que as contribuições do presente trabalho estão inerentemente relacionadas à arquitetura SONAr. Como a arquitetura foi projetada e especificada concomitantemente ao presente trabalho, é nítido que parte do trabalho desta tese é justamente especificar a arquitetura. Neste momento, não é possível afirmar que a SONAr será o modelo de referência para autogerenciamento em redes de computadores. Porém, considerando o que foi explanado ao longo desta tese, devemos tomar a liberdade de afirmar que o gerenciamento de redes de computadores vai utilizar uma arquitetura de autogerenciamento, qual seja a SONAr, qual seja outra proposta correlata.

As duas implementações realizadas durante esta pesquisa também são contribuições relevantes do trabalho. A primeira, FACOM/UFU, fica como legado para que outros alunos do grupo MEHAR ou outros grupos dentro da universidade possam utilizar como base para aprofundar em temas entusiásticos em gerenciamento de redes, como AI, auto-otimização, segurança etc. A segunda, HOEN, é uma contribuição ao projeto HOEN e serviu para introduzir aspectos de autogerenciamento de NSs ao esquema de orquestração

E2E do HOEN.

O *framework* OAM proposto foi implementado em duas tecnologias diferentes, Java e Python, e futuramente pode-se tornar um produto ou ser disponibilizado na comunidade *open-source*. Em ambos os casos, indústria ou academia podem se beneficiar da utilização ou aprimoramento. É importante destacar que a contribuição mais relevante neste caso não é o artefato implementado propriamente dito, mas a descrição das funções que compõem o *framework*. Elas dizem respeito aos serviços de monitoramento e recuperação que nós elucidamos como essenciais para autogerenciamento de redes auto-organizáveis/auto-adaptáveis ou não.

Além das funções de monitoramento e recuperação, a inicialização autônoma e o Catálogo de Gerenciamento são contribuições adicionais deste trabalho. Esses conceitos permitem intervenção humana mínima em um ambiente SDN e NFV e podem futuramente ser incorporadas em especificações das tecnologias.

Até onde pôde ir nossa investigação, nosso *framework* é o primeiro a lidar com autogerenciamento considerando as particularidades dos planos de Dados, Controle e Gerenciamento. Pode-se afirmar que o Plano de Dados possui algoritmos e técnicas eficazes, mas a lacuna na falta de abordagens para o plano de Controle e Gerenciamento permite afirmar que a ideia geral da presente tese fica como contribuição para a área de gerenciamento de redes de computadores e móveis.

Finalmente, foi possível concluir pela viabilidade da aplicação de autogerenciamento, em particular autocura, em SDN e NFV. Provou-se neste trabalho que autocura é possível para essas abordagens, mas requer funcionalidades adicionais não previstas nas tecnologias. Por fim, destaca-se que a implementação do *framework*, através de tecnologias padrão na indústria, elevam as chances de construção de um *framework* OAM como produto. Trata-se de uma contribuição do ponto de vista nacional, visto que normalmente tecnologias de OAM ou redes no geral são importadas de outros países. Portanto, o conhecimento deixado em território nacional é uma contribuição inestimável.

6.2 Trabalhos Futuros

Visto que a arquitetura SONAr está especificada por completo e inclusive pronta para utilização (como foi o caso da construção do *framework* OAM neste trabalho), infere-se que novos trabalhos podem ser conduzidos. O primeiro trabalho claro é o aprofundamento em novos fundamentos de computação autônoma. Até o presente momento, esta tese investigou minuciosamente autocura, enquanto outros dois trabalhos no grupo MEHAR investigaram autoconfiguração. Novos projetos podem aprofundar em outros fundamentos, como por exemplo autoproteção e auto-otimização, visto que a arquitetura suporta e fornece as bibliotecas essenciais (para coleta de métricas, inicialização autônoma e troca de eventos).

Quanto à arquitetura propriamente dita, introduzida no Capítulo 3, não há necessidade de alterações ou evoluções. Pode-se afirmar que a SONAr é capaz de suportar os serviços para os quais foi projetada, conforme foi avaliado no Capítulo 5 desta tese. A incorporação de novas CoEs, SLEs e SOEs, por sua vez, pode ser realizada em outros trabalhos acadêmicos sem pormenores. Por exemplo, uma nova entidade coletora de um tipo de informação não descrito pode ser introduzida como uma nova CoE sem que haja alteração na organização da SONAr: a interoperabilidade continuaria a mesma.

Apesar de não haver necessidade de especificação adicional da arquitetura, as diversas entidades exemplificadas no Capítulo 3 merecem atenção especial e devem ser trabalhadas em breve. Por exemplo, descreveu-se brevemente o papel da *Tuning Self-learning Entity* (TSLE), mas nenhum detalhe de funcionamento e interoperabilidade foi mencionado. É crucial que se investigue e avalie a TSLE, para definir quais são os parâmetros a serem observados, o que se entende por qualidade dos cenários, quais níveis de serviço e limites devem ser levados em consideração para um possível *tuning* etc. De forma análoga, as outras entidades da SONAr também precisam de investigações particulares.

As entidades coletoras também precisam de novos trabalhos. Por exemplo, neste trabalho utilizou-se as entidades coletoras recebendo informações de LAs, bem como buscando informações na topologia. Todavia, não se avaliou as vantagens e desvantagens de uma ou outra abordagem. Essa avaliação pode indicar que uma técnica é melhor que outra para um tipo de topologia, por exemplo, independente se a coleta visa fornecer informações para a SHE, SOPE, SCE ou SPE.

Na perspectiva do fundamento de autocura, alguns trabalhos futuros também podem ser considerados. Um deles seria explorar falhas nos *slices* de gerenciamento intra/inter-domínio. Dessa forma, pode-se especificar protocolos de comunicação para manutenção desses *slices* e avaliá-los em diferentes domínios. Da mesma forma, o Catálogo de Gerenciamento pode ser avaliado com mais componentes do que foi nesta tese. Ambientes NFV possuem vários componentes de gerenciamento de diversos tipos de recursos que podem ser avaliados futuramente.

Na parte de identificação de problemas, este trabalho limitou-se a avaliar detecção de anomalias e predição com funções básicas. Porém, no caso de predição, algoritmos mais eficazes de ML podem ser aplicados para diversos cenários. Entende-se que um possível projeto adicional pode ser criado com grupos de pesquisa de AI para explorar as SLEs.

Ainda no fundamento de autocura, o *framework* proposto nesta tese limitou-se a avaliar questões de OAM do segmento TN. Utilizando a arquitetura e os conceitos do *framework*, pode-se avaliar outros segmentos, como CN, *Data Centre Network* (DCN), RAN, e outros. Do ponto de vista de funções de gerenciamento, não deve haver muitas alterações e os serviços especificados nas fases de monitoramento e recuperação devem ser suficientes. Porém, as métricas avaliadas e as ações de correção variam em diferentes segmentos e portanto podem ser investigadas em trabalhos futuros.

Acerca desta tese, entende-se que os experimentos ficaram limitados aos casos de uso mais relevantes, por uma questão de tempo e espaço para detalhar inúmeros experimentos. No entanto, novos experimentos utilizando os mesmos *setups* das Subseções 5.2.1 e 5.2.2 podem ser realizados. Por exemplo, avaliou-se NSs de URLLC nesta tese, mas outros tipos de comunicação podem ser benefícios com o uso da SONAr, como por exemplo *Enhanced Mobile Broadband* (eMBB).

Ainda considerando os mesmos *setups* desta tese, a integração com sistemas diferentes pode ser benéfica e deve ser investigada. Por exemplo, utilizou-se Ryu e ONOS como controladores SDN nas avaliações deste documento. Em trabalhos futuros, pode-se explorar ODL ou outros controladores para demonstrar a não complexa integração da SONAr com elementos distintos ou para identificar vantagens que esses controladores podem trazer.

Conforme explanado nesta seção, alguns experimentos adicionais podem ser exploradas futuramente utilizando os mesmos *setups* desta tese, investigações de outros fundamentos de computação autônoma podem ser tratados utilizando a SONAr e também outros projetos completos podem ser gerados a partir deste trabalho. Estes projetos serão conduzidos em partes por pesquisadores do grupo MEHAR, mas espera-se que outros grupos de pesquisas internos ou externos à UFU possam se aproveitar da análise e conceitos fornecidos neste trabalho.

6.3 Contribuições em Produção Bibliográfica

Ao longo deste trabalho foram explorados assuntos referentes à arquitetura SONAr, em particular ao fundamento de autocura, e orquestração de NSs em tempo de execução. Alguns artigos foram publicados e submetidos em conferências e periódicos classificados no índice *qualis* restrito, concluindo-se que o tema SONAr e seus diversos aspectos têm sido bem aceito na comunidade acadêmica. Nesta seção, elencam-se as contribuições em produção bibliográfica.

6.3.1 Artigos Publicados

Artigos publicados em congressos e revistas.

- de Souza Neto N.V., Oliveira D.R.C., Gonçalves M.A., de Oliveira Silva F., Rosa P.F. (2021) Self-healing in the Scope of Software-Based Computer and Mobile Networks. In: Ferguson D., Pahl C., Helfert M. (eds) Cloud Computing and Services Science. CLOSER 2020. Communications in Computer and Information Science, vol 1399. Springer, Cham. https://doi.org/10.1007/978-3-030-72369-9_14.

- Neto, N.; Oliveira, D.; Gonçalves, M.; Silva, F. and Rosa, P. (2020). A Self-healing Platform for the Control and Management Planes Communication in Softwarized

and Virtualized Networks.In Proceedings of the 10th International Conference on Cloud Computing and Services Science – Volume 1: CLOSER, ISBN 978-989-758-424-4, pages 415-422. DOI: 10.5220/0009465204150422.

- Oliveira, D.; Neto, N.; Gonçalves, M.; Silva, F. and Rosa, P. (2020). Network Self-configuration for Edge Elements using Self-Organizing Networks Architecture (SONAr).In Proceedings of the 10th International Conference on Cloud Computing and Services Science – Volume 1: CLOSER, ISBN 978-989-758-424-4, pages 408-414. DOI: 10.5220/0009465104080414.

- Gonçalves, M.; Neto, N.; Oliveira, D.; Silva, F. and Rosa, P. (2020). Bootstrapping and Plug-and-Play Operations on Software Defined Networks: A Case Study on Self-configuration using the SONAr Architecture.In Proceedings of the 10th International Conference on Cloud Computing and Services Science – Volume 1: CLOSER, ISBN 978-989-758-424-4, pages 103-114. DOI: 10.5220/0009406901030114.

- J. F. Santos; Wei Liu; Xianjun Jiao; Natal V. Neto; Sofie Pollin; Johann M. Marquez-Barja; Ingrid Moerman and Luiz A. DaSilva. "Breaking Down Network Slicing: Hierarchical Orchestration of End-to-End Networks," in IEEE Communications Magazine, vol. 58, no. 10, pp. 16-22, October 2020, doi: 10.1109/MCOM.001.2000406.

- Oliveira, D.; Gonçalves, M.; Neto, N.; Silva, F. and Rosa, P. (2021). Specialized Network Self-configuration: An Approach using Self-Organizing Networks Architecture (SONAr). In Proceedings of the 11th International Conference on Cloud Computing and Services Science - CLOSER, ISBN 978-989-758-510-4, pages 161-168. DOI: 10.5220/0010403601610168.

6.3.2 Artigos Submetidos e em Revisão

Artigos submetidos e em revisão. Algumas informações ainda não estão disponíveis.

- N. V. de Souza Neto; Maurício A. Gonçalves; D. R. C. Oliveira; F. O. Silva and P. F. Rosa. "Management Slices: Enabling Self-management for SDN and NFV Communication Layers" in IEEE Communications Magazine.

- N. V. de Souza Neto; Maurício A. Gonçalves; D. R. C. Oliveira; F. O. Silva and P. F. Rosa. "VFMP: A Protocol that Provides OAM's Awareness to Ensure Virtual Network Functions' Health". XXXIX Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos SBRC 2021.

Referências

3GPP. *Telecommunication management; Self-configuration of network elements; Concepts and requirements*. [S.l.], 2018. Disponível em: <http://www.3gpp.org/-DynaReport/32501.htm>.

3GPP. *Telecommunication management; Self-Organizing Networks (SON); Concepts and requirements*. [S.l.], 2018. Disponível em: <http://www.3gpp.org/DynaReport/-32500.htm>.

3GPP. *Telecommunication management; Self-Organizing Networks (SON) Policy Network Resource Model (NRM) Integration Reference Point (IRP); Requirements*. [S.l.], 2018. Disponível em: <http://www.3gpp.org/DynaReport/32521.htm>.

3GPP. *Telecommunication management; Self-Organizing Networks (SON); Self-healing concepts and requirements*. [S.l.], 2018. Disponível em: <http://www.3gpp.org/-DynaReport/32541.htm>.

3GPP. *Telecommunication management;Study on management and orchestration of network slicing for next generation network*. [S.l.], 2018. Version 15.1.0. Disponível em: <http://www.3gpp.org/DynaReport/28801.htm>.

Abdallah, S. et al. A network management framework for sdn. In: **2018 9th IFIP International Conference on New Technologies, Mobility and Security (NTMS)**. [s.n.], 2018. p. 1–4. Disponível em: <https://doi.org/10.1109/NTMS.2018.8328672>.

ABDELSALAM, M. A. **Network Application Design Challenges and Solutions in SDN**. Tese (Doutorado) — Carleton University, 2018. Disponível em: <https://doi.org/10.22215/etd/2018-13338>.

Afolabi, I. et al. Network slicing and softwarization: A survey on principles, enabling technologies, and solutions. **IEEE Communications Surveys Tutorials**, v. 20, n. 3, p. 2429–2453, thirdquarter 2018. ISSN 2373-745X. Disponível em: <https://doi.org/10.1109/COMST.2018.2815638>.

AL-OQILY, I. et al. A survey for self-healing architectures and algorithms. In: **International Multi-Conference on Systems, Sygnals Devices**. [s.n.], 2012. p. 1–5. Disponível em: <https://doi.org/10.1109/SSD.2012.6198057>.

Apache Software Foundation. **Apache Cassandra**. 2021. Disponível em: <https://cassandra.apache.org/>.

AZAB, M.; FORTES, J. A. B. Towards proactive sdn-controller attack and failure resilience. In: **2017 International Conference on Computing, Networking and Communications (ICNC)**. [s.n.], 2017. p. 442–448. Disponível em: <https://doi.org/10.1109/ICCNC.2017.7876169>.

BERGSTRA, J.; BURGESS, M. **Handbook of Network and System Administration**. [S.l.]: Elsevier, 2011. Google-Books-ID: NUoZ7fKOITQC. ISBN 978-0-08-055358-0.

BIANCHI, G. et al. Superfluidity: a flexible functional architecture for 5g networks. **Transactions on Emerging Telecommunications Technologies**, Wiley Online Library, v. 27, n. 9, p. 1178–1186, 2016. Disponível em: <https://doi.org/10.1002/ett.3082>.

BLANCO, B. et al. Technology pillars in the architecture of future 5G mobile networks: NFV, MEC and SDN. **Computer Standards & Interfaces**, v. 54, p. 216–228, 2017. Publisher: Elsevier. Disponível em: <https://doi.org/10.1016/j.csi.2016.12.007>.

BONÉR, J. et al. The reactive manifesto. **Dosegljivo: http://www. reactivemanifesto. org/.[Dostopano: 21. 08. 2017]**, 2014. Disponível em: <https://www.reactivemanifesto.org/>.

BOSCHI, S.; SANTOMAGGIO, G. **RabbitMQ cookbook**. [S.l.]: Packt Publishing Ltd, 2013.

BOURAS, C.; KOLLIA, A.; PAPAZOIS, A. Sdn & nfv in 5g: Advancements and challenges. In: IEEE. **2017 20th Conference on innovations in clouds, internet and networks (ICIN)**. 2017. p. 107–111. Disponível em: <https://doi.org/10.1109/ICIN.2017.7899398>.

CANINI, M. et al. A self-organizing distributed and in-band sdn control plane. In: **2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS)**. [s.n.], 2017. p. 2656–2657. ISSN 1063-6927. Disponível em: <https://doi.org/10.1109/ICDCS.2017.328>.

CASSANDRA, A. Apache cassandra. **Website. Available online at http://planetcassandra. org/what-is-apache-cassandra**, p. 13, 2014.

CHANDRASEKARAN, B.; BENSON, T. Tolerating sdn application failures with legosdn. In: **Proceedings of the 13th ACM workshop on hot topics in networks**. [s.n.], 2014. p. 1–7. Disponível em: <https://doi.org/10.1145/2670518.2673880>.

CHANDRASEKARAN, B.; TSCHAEN, B.; BENSON, T. Isolating and tolerating sdn application failures with legosdn. In: **Proceedings of the Symposium on SDN Research**. New York, NY, USA: ACM, 2016. (SOSR '16), p. 7:1–7:12. ISBN 978-1-4503-4211-7. Disponível em: <http://doi.acm.org/10.1145/2890955.2890965>.

Chen, H. et al. Ultra-reliable low latency cellular networks: Use cases, challenges and approaches. **IEEE Communications Magazine**, v. 56, n. 12, p. 119–125, December 2018. ISSN 1558-1896. Disponível em: <https://doi.org/10.1109/MCOM.2018.1701178>.

CLEMM, A. **Network Management Fundamentals**. [S.l.]: Cisco Press, 2006. ISBN 978-1-58720-137-0.

CONSTANTINE, B. et al. **Framework for TCP Throughput Testing**. RFC Editor, 2011. RFC 6349. (Request for Comments, 6349). Disponível em: <https://rfc-editor.org/rfc/rfc6349.txt>.

COX, J. H. et al. Advancing software-defined networks: A survey. **IEEE Access**, v. 5, p. 25487–25526, 2017. Disponível em: <https://doi.org/10.1109/ACCESS.2017.2762291>.

DALAL, A. C. A framework for self-healing home networks. In: **10th International Conference on Heterogeneous Networking for Quality, Reliability, Security and Robustness**. [s.n.], 2014. p. 135–136. Disponível em: <https://doi.org/10.1109/QSHINE.2014.6928674>.

DING, J. **Advances in Network Management**. 0. ed. Auerbach Publications, 2016. ISBN 978-0-429-12091-6. Disponível em: <https://doi.org/10.1201/9781420064551>.

DORIA, A. et al. **Forwarding and Control Element Separation (ForCES) Protocol Specification**. RFC Editor, 2010. RFC 5810. (Request for Comments, 5810). Disponível em: <https://rfc-editor.org/rfc/rfc5810.txt>.

ENNS, R. et al. **Network Configuration Protocol (NETCONF)**. RFC Editor, 2011. RFC 6241. (Request for Comments, 6241). Disponível em: <https://rfc-editor.org/rfc/rfc6241.txt>.

ETSI, G. Network functions virtualisation (nfv): Architectural framework. **ETSI GS NFV**, v. 2, n. 2, p. V1, 2014.

FONSECA, P. C. d. R.; MOTA, E. S. A survey on fault management in software-defined networks. **IEEE Communications Surveys Tutorials**, v. 19, n. 4, p. 2284–2321, Fourthquarter 2017. Disponível em: <https://doi.org/10.1109/COMST.2017.2719862>.

Foukas, X. et al. Network slicing in 5g: Survey and challenges. **IEEE Communications Magazine**, v. 55, n. 5, p. 94–100, May 2017. ISSN 1558-1896. Disponível em: <https://doi.org/10.1109/MCOM.2017.1600951>.

GACANIN, H.; LIGATA, A. Wi-fi self-organizing networks: Challenges and use cases. **IEEE Communications Magazine**, v. 55, n. 7, p. 158–164, 2017. ISSN 0163-6804. Disponível em: <https://doi.org/10.1109/MCOM.2017.1600523>.

GANEK, A. G.; CORBI, T. A. The dawning of the autonomic computing era. **IBM systems Journal**, IBM, v. 42, n. 1, p. 5–18, 2003. Disponível em: <https://doi.org/10.1147/sj.421.0005>.

GHOSH, D. et al. Self-healing systems-survey and synthesis. **Decision support systems**, Elsevier, v. 42, n. 4, p. 2164–2185, 2007. Disponível em: <https://doi.org/10.1016/j.dss.2006.06.011>.

HAN, B. et al. Network function virtualization: Challenges and opportunities for innovations. **IEEE Communications Magazine**, IEEE, v. 53, n. 2, p. 90–97, 2015. Disponível em: <https://doi.org/10.1109/MCOM.2015.7045396>.

HEGERING, H.-G. **Integrated Management of Networked Systems: Concepts, Architectures and Their Operational Application**. [S.l.]: Morgan Kaufmann, 1999. Google-Books-ID: pssD4DE8JlsC. ISBN 978-1-55860-571-8.

IRIS. **IRIS Testbed**. 2021. Disponível em: <http://iristestbed.eu/>.

ITU. **ITU-T Recommendation X.700 : MANAGEMENT FRAMEWORK FOR OPEN SYSTEMS INTERCONNECTION (OSI) FOR CCITT APPLICATIONS**. [S.l.], 1993.

Kubernetes. **Kubernetes**. 2021. Disponível em: <https://kubernetes.io/>.

Linux Foundation. **Open vSwitch**. 2021. Disponível em: <https://www.openvswitch.org/>.

LLDPD. **lldpd - Implementation of IEEE 802.1AB (LLDP)**. 2019. Disponível em: <https://vincentbernat.github.io/lldpd/>.

LÓPEZ, L. B. et al. Key technologies in the context of future networks: operational and management requirements. **Future Internet**, Multidisciplinary Digital Publishing Institute, v. 9, n. 1, p. 1, 2017. Disponível em: <https://doi.org/10.3390/fi9010001>.

MERKEL, D. Docker: lightweight linux containers for consistent development and deployment. v. 2014, n. 239, p. 2, 2014.

MIZRAHI, T. et al. **An Overview of Operations, Administration, and Maintenance (OAM) Tools**. RFC Editor, 2014. RFC 7276. (Request for Comments, 7276). Disponível em: <https://rfc-editor.org/rfc/rfc7276.txt>.

NEUMANN, J. C. **The Book of GNS3**. 1st. ed. San Francisco, CA, USA: No Starch Press, 2014. ISBN 1593275544, 9781593275549.

NEVES, P. et al. The selfnet approach for autonomic management in an nfv/sdn networking paradigm. **International Journal of Distributed Sensor Networks**, SAGE Publications Sage UK: London, England, v. 12, n. 2, p. 2897479, 2016. Disponível em: <https://doi.org/10.1155/2016/2897479>.

ONOS. **ONOS**. 2021. Disponível em: <https://wiki.onosproject.org/>.

OpenDaylight. **OpenDaylight**. 2021. Disponível em: <https://www.opendaylight.org/>.

Openstack. **Open Source Cloud Computing Infrastructure**. 2021. Disponível em: <https://www.openstack.org/>.

PADMA, V.; YOGESH, P. Proactive failure recovery in openflow based software defined networks. In: **2015 3rd International Conference on Signal Processing, Communication and Networking (ICSCN)**. [s.n.], 2015. p. 1–6. Disponível em: <https://doi.org/10.1109/ICSCN.2015.7219846>.

Parvez, I. et al. A survey on low latency towards 5g: Ran, core network and caching solutions. **IEEE Communications Surveys Tutorials**, v. 20, n. 4, p. 3098–3130, Fourthquarter 2018. ISSN 2373-745X. Disponível em: <https://doi.org/10.1109/COMST.2018.2841349>.

POSTEL, J. **INTERNET CONTROL MESSAGE PROTOCOL**. RFC Editor, 1981. RFC 792. (Request for Comments, 792). Disponível em: <https://rfc-editor.org/rfc/rfc792.txt>.

PSAIER, H.; DUSTDAR, S. A survey on self-healing systems: approaches and systems. **Computing**, Springer, v. 91, n. 1, p. 43–73, 2011. Disponível em: <https://doi.org/10.1007/s00607-010-0107-y>.

RAMIREZ-PEREZ, C.; RAMOS, V. Sdn meets sdr in self-organizing networks: fitting the pieces of network management. **IEEE Communications Magazine**, v. 54, n. 1, p. 48–57, January 2016. ISSN 0163-6804. Disponível em: <https://doi.org/10.1109/MCOM.2016.7378425>.

RAMIRO, J.; HAMIED, K. **Self-organizing networks: self-planning, self-optimization and self-healing for GSM, UMTS and LTE**. John Wiley & Sons, 2011. Disponível em: <https://doi.org/10.1002/9781119954224>.

Rehman, A. U.; Aguiar, R. L.; Barraca, J. P. Fault-tolerance in the scope of software-defined networking (sdn). **IEEE Access**, v. 7, p. 124474–124490, 2019. ISSN 2169-3536. Disponível em: <https://doi.org/10.1109/ACCESS.2019.2939115>.

Ryu SDN Framework Community. **Ryu SDN Framework**. 2021. Disponível em: <https://ryu-sdn.org/>.

SADIGHI, A. et al. Design methodologies for enabling self-awareness in autonomous systems. In: **2018 Design, Automation & Test in Europe Conference & Exhibition (DATE)**. IEEE, 2018. p. 1532–1537. Disponível em: <https://doi.org/10.23919/DATE.2018.8342259>.

SANCHEZ, J. et al. Softwarized 5g networks resiliency with self-healing. In: **1st International Conference on 5G for Ubiquitous Connectivity**. [s.n.], 2014. p. 229–233. Disponível em: <https://doi.org/10.4108/icst.5gu.2014.258123>.

SANTOS, J. F. et al. **Orchestration next-generation services through end-to-end networks slicing**. 2018. Disponível em: <https://orca-project.eu/wp-content/uploads/sites/4/2018/10/orchestrating_e2e_network_slices_Final.pdf>.

SNMP net. **Net-SNMP**. 2019. Disponível em: <http://www.net-snmp.org/>.

Spotify. **Docker-client java api**. 2021. Disponível em: <https://www.javadoc.io/doc/com.spotify/docker-client/latest/index.html>.

STEIN, A. et al. A concept for proactive knowledge construction in self-learning autonomous systems. In: **2018 IEEE 3rd International Workshops on Foundations and Applications of Self* Systems (FAS* W)**. IEEE, 2018. p. 204–213. Disponível em: <https://doi.org/10.1109/FAS-W.2018.00048>.

STEVENSON, D. W. Network management: What it is and what it isn't. **White Paper**, v. 4, 1995.

SUBRAMANIAN, M. **Network Management: Principles and Practice**. [S.l.]: Dorling Kindersley, 2010. Google-Books-ID: VGDMIIfL6XcC. ISBN 978-81-317-2759-1.

The ZeroMQ authors. **ZeroMQ**. 2021. Disponível em: <https://zeromq.org/>.

THORAT, P. et al. Optimized self-healing framework for software defined networks. In: ACM. **Proceedings of the 9th International Conference on Ubiquitous Information Management and Communication**. 2015. p. 7. Disponível em: <https://doi.org/10.1145/2701126.2701235>.

Tsagkaris, K. et al. Customizable autonomic network management: Integrating autonomic network management and software-defined networking. **IEEE Vehicular Technology Magazine**, v. 10, n. 1, p. 61–68, March 2015. Disponível em: <https://doi.org/10.1109/MVT.2014.2380633>.

VERMESAN, O. et al. Internet of robotic things: converging sensing/actuating, hypoconnectivity, artificial intelligence and IoT Platforms. 2017.

VILCHEZ, J. M. S.; YAHIA, I. G. B.; CRESPI, N. Self-healing mechanisms for software defined networks. In: **8th International Conference on Autonomous Infrastructure, Management and Security (AIMS 2014)**. [S.l.: s.n.], 2014.

VMware. **RabbitMQ**. 2021. Disponível em: <https://www.rabbitmq.com/>.

_____. **Spring Boot**. 2021. Disponível em: <https://spring.io/projects/spring-boot>.

WANG, C.; YAN, S. Scaling sdn network with self-adjusting architecture. In: **2016 IEEE International Conference on Electronic Information and Communication Technology (ICEICT)**. [s.n.], 2016. p. 116–120. Disponível em: <https://doi.org/10.1109/ICEICT.2016.7879664>.

WICKBOLDT, J. A. et al. Software-defined networking: management requirements and challenges. **IEEE Communications Magazine**, v. 53, n. 1, p. 278–285, January 2015. ISSN 0163-6804. Disponível em: <https://doi.org/10.1109/MCOM.2015.7010546>.

XU, L. et al. Cognet: A network management architecture featuring cognitive capabilities. In: IEEE. **2016 European Conference on Networks and Communications (EuCNC)**. 2016. p. 325–329. Disponível em: <https://doi.org/10.1109/EuCNC.2016.7561056>.

ZAIDI, Z. et al. Will SDN Be Part of 5g? **IEEE Communications Surveys & Tutorials**, v. 20, n. 4, p. 3220–3258, 2018. ISSN 1553-877X, 2373-745X. Disponível em: <https://doi.org/10.1109/COMST.2018.2836315>.

ZHANG, J.; XIE, W.; YANG, F. An architecture for 5g mobile network based on sdn and nfv. IET, 2015. Disponível em: <https://doi.org/10.1049/cp.2015.0918>.

Zhou, X. et al. Network slicing as a service: enabling enterprises' own software-defined cellular networks. **IEEE Communications Magazine**, v. 54, n. 7, p. 146–153, July 2016. ISSN 1558-1896. Disponível em: <https://doi.org/10.1109/MCOM.2016.7509393>.

Apêndices

APÊNDICE **A**

Detalhes de Implementação FACOM

Neste capítulo apresenta-se heuristicamente alguns detalhes da implementação NOSH na FACOM/UFU. Esses detalhes permitem que o leitor perceba algumas particularidades de implementação, especialmente relacionadas ao bloqueio de componentes no Catálogo de Gerenciamento e consequente procedimentos necessários para autocura do próprio NOSH.

A.1 Implementação

Este apêndice pretende mostrar ao leitor a implementação FACOM/UFU utilizada nos experimentos da Subseção 5.2.1. Devido a característica agnóstica de tecnologia da SONAr, esta não é uma implementação definitiva e por isso está registrada nos apêndices. Por outro lado, permite ao leitor utilizar os detalhes para sua própria implementação.

O primeiro serviço ou aplicação a ser inicializado no ambiente FACOM/UFU é o *Containerised Infrastructure Manager* (CIM). Criou-se esse componente para gerenciar *containers* ao longo da infraestrutura. Portanto, o CIM pode orquestrar *containers* não somente no servidor local que está executando, mas também em outros servidores ao longo do domínio. O CIM se conecta à *engine* Docker dos servidores para executar operações em *containers* (por exemplo inicializar, parar e destruir). O CIM é uma aplicação Java construída utilizando-se o Spring Boot versão 2.1.2.RELEASE e a biblioteca de cliente Docker utilizada é a Spotify Docker Client API versão 8.15.1 (Spotify, 2021).

Essencialmente, o CIM expõe uma API HTTP REST versão 1 com *endpoints* para que as demais aplicações SONAr possam requisitá-lo. Por exemplo, a API 'container' permite que outras entidades requisitem um HTTP POST para iniciar um *container*, caso conheçam previamente o ID ou nome pré-estabelecido daquele *container*. Após o CIM estar executando, inicia-se o *Auto-boot Manager* (ABM), que vai inicializar autonomamente todos os demais *containers* (com aplicações da SONAr ou aplicações de rede diversas). O pseudoalgoritmo da Figura 36 mostra o procedimento de inicialização do ABM.

Conforme mostrado, o passo a passo de inicialização do ABM é iniciar o NDB e aguardar até que o mesmo esteja inicializado para executar o *schema* para criação das tabelas da SONAr e do Catálogo de Gerenciamento. O ABM também aguarda que uma SHE se

```
    runComponent(NDB);
    runComponent(NEM);
 3: runComponent(SDNC);
    while NDB is not initialised do
      sleep(2000);
 6:   continue;
    end while
    runNDBSchema();
 9: runOtherVNFs();
    runOtherSONArEntities();
    waitSHE();
12: sendContainerInformationToNEM();
```

Figura 36 – Algoritmo de Inicialização do ABM.

inicie para enviar as informações de todos os *containers* executados para ela. A função `runComponent` basicamente monta as propriedades (variáveis de ambiente e propriedades dos *containers*) e requisita a API do CIM para iniciar o *container* da aplicação. As imagens de Docker utilizadas serão posteriormente disponibilizadas no repositório Docker Hub, no link <https://hub.docker.com/u/meharsonar>, para que outros pesquisadores possam reproduzir ou criar novos experimentos. As imagens dessa implementação estão listadas abaixo:

- ❏ meharsonar/self-healing-entity
- ❏ meharsonar/self-protection-entity
- ❏ meharsonar/self-planning-entity
- ❏ meharsonar/self-optimization-entity
- ❏ meharsonar/self-management-entity
- ❏ meharsonar/self-configuration-entity
- ❏ meharsonar/analysis-self-learning-entity
- ❏ meharsonar/prediction-self-learning-entity
- ❏ meharsonar/tunning-self-learning-entity
- ❏ meharsonar/intent-self-learning-entity
- ❏ meharsonar/rating-self-learning-entity
- ❏ meharsonar/metrics-self-collector-entity
- ❏ meharsonar/logs-self-collector-entity

```
for all Metric Event Received do
  status = parseEvent();
  components = queryManagementCatalog();
  for each component do
    lock(component);
    if status not contains component then
      resetComponent(component);
    else if component status is not RUNNING then
      rebootComponent(component);
    end if
    while component is not running do
      sleep(4000);
    end while
    unlock(component);
  end for
end for
```

Figura 37 – Algoritmo do Serviço de Saúde de *Containers* na SHE.

- ❏ meharsonar/topology-self-collector-entity
- ❏ meharsonar/samples-self-collector-entity
- ❏ meharsonar/alarms-self-collector-entity
- ❏ meharsonar/sonar-dhcp-server
- ❏ meharsonar/sonar-controller-interceptor
- ❏ meharsonar/rabbitmq
- ❏ meharsonar/cassandra
- ❏ meharsonar/onos

Uma vez que a SHE recebe as informações do ABM, ela armazena no Catálogo de Gerenciamento que está criado no NDB e logo após monitora e recupera a rede baseando-se em eventos recebidos pela MCoE. A MCoE basicamente possui um *schedule* que roda periodicamente para requisitar um GET na API *container* que está exposta no CIM. Ressalta-se que todas as propriedades são inicializadas pelo ABM. Por exemplo, a propriedade contendo o IP do CIM é conhecida pelo *container* executando a MCoE. Dessa forma, ela sabe para onde requisitar o método HTTP.

A SHE pode ter diversos microserviços para tratar eventos diferentes. Exemplifica-se aqui o microserviço 'ContainerHealthService'. Nesse microserviço, a SHE subscreve-se a eventos do tópico `METRIC`. As informações coletadas pela MCoE são inseridas nesse tópico. O pseudoalgoritmo da Figura 37 mostra o funcionamento da SHE nesse caso.

Nota-se que a SHE bloqueia o componente no Catálogo para que eventos seguintes não tratem o mesmo componente. Para cada componente, neste exemplo, a SHE verifica as informações de status recebidas através do NEM (coletadas pela MCoE). Observa-se que, se não há informação de status de um componente, aquele *container* deixou de existir e portanto a SHE aplica uma recuperação por redefinição. Se por outro lado há a informação de status mas o *container* não está no status `RUNNING`, a SHE aplica uma recuperação por reinicialização. As recuperações são executadas a partir de requisições diretas ao CIM ou submetendo um evento ao NEM, que será lido pelo NSB e tratado.

A implementação desta tese contém diversos outros serviços e entidades. Porém, neste apêndice listou-se apenas algumas partes para que o leitor perceba como o NOSH funciona. Para referência futura, pretende-se manter a página do projeto SONAr atualizada. Nessa página, é possível encontrar os artigos, apresentações e informações gerais do projeto. O link é <https://sonar.facom.ufu.br/>. Os códigos do projeto são armazenados no repositório GIT da FACOM/UFU, no link <http://gitlab.facom.ufu.br/sonar/>.

APÊNDICE **B**

Detalhes de Implementação HOEN

Neste capítulo apresenta-se heuristicamente alguns detalhes da implementação NOSH integrado ao HOEN. Esses detalhes permitem que o leitor perceba algumas particularidades de implementação, especialmente relacionadas com a interoperabilidade entre o NOSH e um orquestrador de NSs.

B.1 Implementação

Na implementação do HOEN há o estabelecimento de LAs para coleta de métricas de rede. Nesse estabelecimento autônomo, cada LA requisita, através da primitiva `config_req`, as informações de configuração para a MCoE. Esta por sua vez busca as informações de topologia no NDB e as informações de LAs já cadastrados previamente. Com essas informações, a MCoE calcula todos os caminhos entre o LA que está sendo configurado e todos os outros demais. Além disso, gera endereços IP de gerenciamento para as interfaces virtuais. Após o cálculo das rotas e definições de IP, a MCoE monta todos os comandos de aplicação de fluxos de encaminhamento nos NEs. Nessa implementação, os NEs são *Open vSwitches* (OVSs) e, portanto, a MCoE requisita ao Ryu o estabelecimento de todos os fluxos de encaminhamento.

O funcionamento dos LAs pode ser exemplificado através do pseudoalgoritmo da Figura 38. Basicamente, ao iniciar, um LA busca as configurações na MCoE e as aplica localmente. Posteriormente, o LA entra em um laço para requisitar latência, através dos utilitários `ping` (POSTEL, 1981) e `iperf3` (CONSTANTINE et al., 2011), em busca de métricas de latência para todos os LAs em todos os possíveis caminhos.

A MCoE recebe as métricas enviadas pelos LAs a todo instante e as publica no tópico `METRIC`. A SHE recebe todos os eventos do tópico `METRIC`. Para cada evento, a SHE analisa a métrica informada. Essencialmente, a SHE busca todos os caminhos da topologia armazenados e solicita ao orquestrador de NSs informações sobre todos os NSs. A SHE então avalia quais NSs possuem latência como requisito e verifica se a métrica recebida (contendo uma rota e um valor de latência) indica a mesma rota que o NS utiliza. Em caso positivo, a SHE verifica se a latência requerida pelo NS é menor que a latência atual.

```
config = sendConfigReq();
configLocalInterfaces(config);
while true do
  for each other LA do
    metrics = pingLAfromAllPaths();
    for each metric in metrics do
      reportMCoE(metric);
    end for
  end for
  sleep(500);
end while
```

Figura 38 – Algoritmo do LA em uma Rede Fim-a-Fim.

```
for each event in topic metrics do
  paths = queryPathsInNDB();
  slices = querySlicesToTNOrchestrator();
  for each slice in slices do
    route = paths.path(slice);
    if current metric is greater than slice required lantecy then
      recoverySlice(slice);
    end if
  end for
end for
```

Figura 39 – Algoritmo de Análise da SHE.

Se for, o processo de recuperação se inicia. O fluxo é apresentado no pseudocódigo da Figura 39.

O processo de recuperação pode ser em tempo real ou baseado nos *buckets*. Basicamente, a SHE recalcula caminhos que permitam ao NS cumprir seu requisito de latência. Para isso, a SHE verifica a latência atual de cada rota nas últimas métricas recebidas de cada rota. Após isso, a SHE monta os fluxos de encaminhamento e requisita ao controlador Ryu a aplicação de modificações de fluxo.

A implementação desta tese contém diversos outros serviços e entidades. Porém, neste apêndice listou-se apenas algumas partes para que o leitor perceba como o NOSH trabalha como OAM em um ambiente de uma rede E2E. Para referência futura, pretende-se manter a página do projeto SONAr atualizada com informações desta e outras integrações que porventura o NOSH seja submetido. O link é <https://sonar.facom.ufu.br/>.