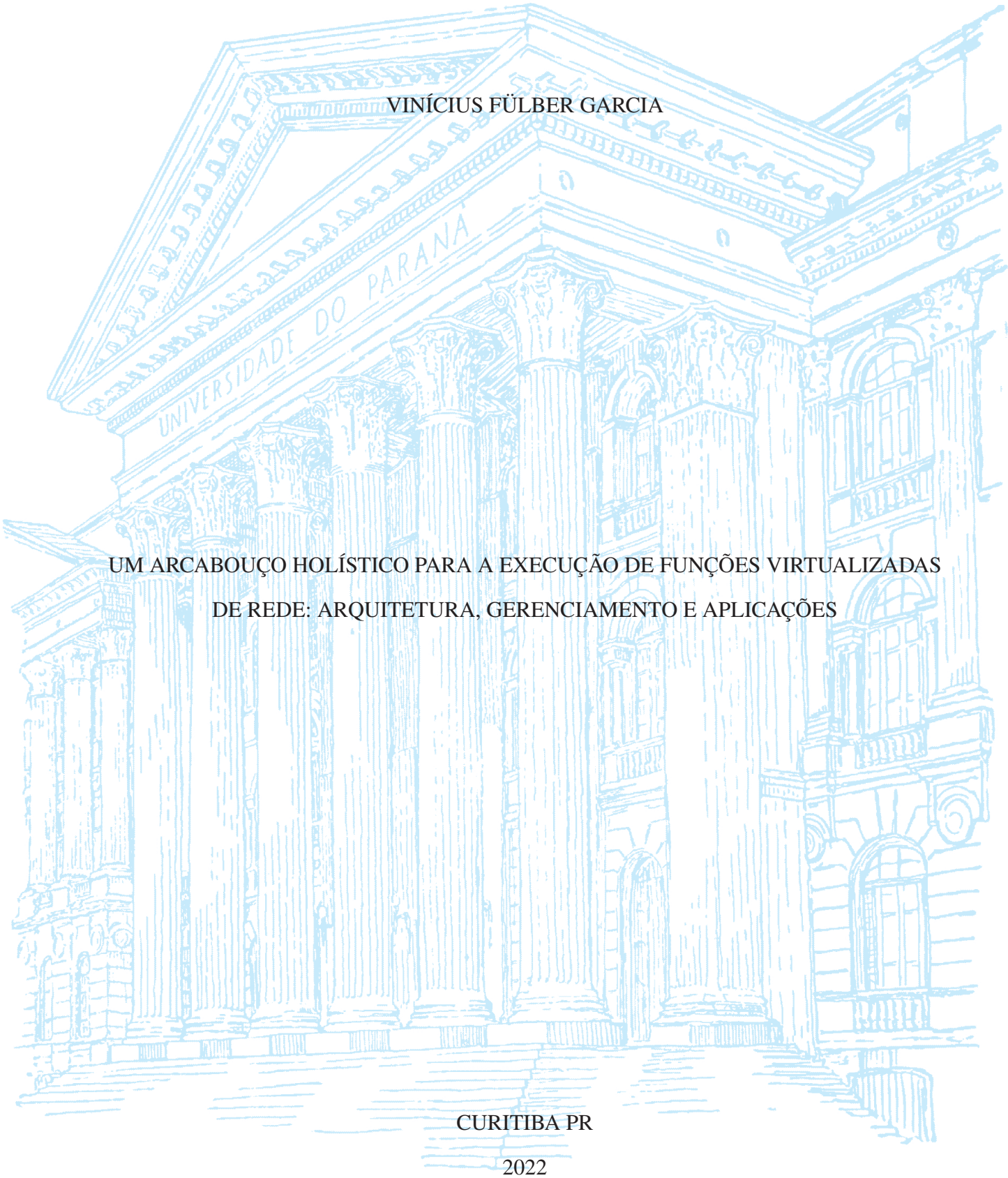
UNIVERSIDADE FEDERAL DO PARANÁ

VINÍCIUS FÜLBER GARCIA



UM ARCABOUÇO HOLÍSTICO PARA A EXECUÇÃO DE FUNÇÕES VIRTUALIZADAS DE REDE: ARQUITETURA, GERENCIAMENTO E APLICAÇÕES

CURITIBA PR

2022

VINÍCIUS FÜLBER GARCIA

UM ARCABOUÇO HOLÍSTICO PARA A EXECUÇÃO DE FUNÇÕES VIRTUALIZADAS DE REDE: ARQUITETURA, GERENCIAMENTO E APLICAÇÕES

Tese apresentada como requisito parcial à obtenção do grau de Doutor em Ciência da Computação no Programa de Pós-Graduação em Informática, Setor de Ciências Exatas, da Universidade Federal do Paraná.

Área de concentração: *Ciência da Computação*.

Orientador: Elias Procópio Duarte Júnior.

CURITIBA PR

2022

DADOS INTERNACIONAIS DE CATALOGAÇÃO NA PUBLICAÇÃO (CIP)
UNIVERSIDADE FEDERAL DO PARANÁ
SISTEMA DE BIBLIOTECAS – BIBLIOTECA DE CIÊNCIA E TECNOLOGIA

Garcia, Vinícius Fülber
Um arcabouço holístico para a execução de funções virtualizadas de rede : arquitetura, gerenciamento e aplicações / Vinícius Fülber Garcia. – Curitiba, 2022.
1 recurso on-line : PDF.

Tese (Doutorado) - Universidade Federal do Paraná, Setor de Ciências Exatas, Programa de Pós-Graduação em Informática.

Orientador: Elias Procópio Duarte Júnior

1. Arquitetura de redes de computador. 2. Redes locais de computadores. 3. Ambientes virtuais compartilhados. 4. Redes virtualizadas. I. Universidade Federal do Paraná. II. Programa de Pós-Graduação em Informática. III. Duarte Júnior, Elias Procópio. IV. Título.

Bibliotecário: Elias Barbosa da Silva CRB-9/1894

TERMO DE APROVAÇÃO

Os membros da Banca Examinadora designada pelo Colegiado do Programa de Pós-Graduação INFORMÁTICA da Universidade Federal do Paraná foram convocados para realizar a arguição da tese de Doutorado de **VINÍCIUS FÜLBER GARCIA** intitulada: **Um Arcabouço Holístico para a Execução de Funções Virtualizadas de Rede: Arquitetura, Gerenciamento e Aplicações**, sob orientação do Prof. Dr. ELIAS PROCÓPIO DUARTE JÚNIOR, que após terem inquirido o aluno e realizada a avaliação do trabalho, são de parecer pela sua APROVAÇÃO no rito de defesa.

A outorga do título de doutor está sujeita à homologação pelo colegiado, ao atendimento de todas as indicações e correções solicitadas pela banca e ao pleno atendimento das demandas regimentais do Programa de Pós-Graduação.

CURITIBA, 16 de Dezembro de 2022.

Assinatura Eletrônica
19/12/2022 11:20:41.0
ELIAS PROCÓPIO DUARTE JÚNIOR
Presidente da Banca Examinadora

Assinatura Eletrônica
19/12/2022 17:00:13.0
RAFAEL PASQUINI
Avaliador Externo (UNIVERSIDADE FEDERAL DE UBERLÂNDIA)

Assinatura Eletrônica
19/12/2022 14:05:21.0
WAGNER MACHADO NUNAN ZOLA
Avaliador Interno (UNIVERSIDADE FEDERAL DO PARANÁ)

Assinatura Eletrônica
19/12/2022 16:01:37.0
ALBERTO EGON SCHAEFFER FILHO
Avaliador Externo (UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL)

Aos meus pais, pessoas amadas e comunidade acadêmica brasileira

AGRADECIMENTOS

Ser humano é ser pensante. Todos nós que compartilhamos da sorte do desenvolvimento da cognição somos naturalmente intelectuais da nossa própria história. Porém, o desejo pelo consumo e produção formal de ciência nos leva a tão sublime academia. Esta que, hoje, enfrenta uma batalha de Termópilas, encurralada por medidas de desmanche e repúdio vindas do covil reacionário da humanidade. Por sua constante e incontestável luta pelo progresso, obrigado academia! Em particular, obrigado academia brasileira por me proporcionar os anos mais desafiadores, mas também os mais emocionantes da minha vida até então.

Aos mestres, devo minha gratidão completa. Pois estes me guiaram não apenas através do conhecimento, mas também ao longo da ignorância. Reconheço hoje que somos todos pequenos quanto a vastidão do conhecimento possível, e quanto a infinitude daquele ainda incógnito. Na figura dos meus orientadores Elias Procópio Duarte Jr., Carlos Raniery Paula dos Santos, e Sergio Luis Sardi Mergen, agradeço imensamente todos os meus professores.

Aos alunos, agradeço pelas suas aulas, uma vez que reconheço em vocês o meu melhor eu. Exercer a docência não se trata de palestrar, mas sim de cooperar e construir. O conhecimento técnico é apenas a base formal de um processo muito mais relacionado ao amor: ensinar.

Aos colegas, agradeço pelo companheirismo. Vocês me permitiram o exercício pleno da admiração e a felicidade em vibrar por cada uma de suas conquistas. Eu os acompanho e acompanharei em todas as suas providências profissionais e pessoais e, em júbilo, espalharei aos quatro cantos suas vitórias.

Aos amigos, agradeço pelos instantes, momentos. Para os camaradas que não mais tenho contado, agradeço pelo passado; àqueles ainda em constante convivência, agradeço pelo presente; para àqueles que ainda não tive o deleite de conhecer, agradeço pelo futuro. Na figura do Rafael Gomes de Castro, me remeto especialmente aos amigos de Curitiba, extremamente presentes e atenciosos nos últimos anos.

Às musas, agradeço pela sabedoria. Foi na visão de mundo do feminino que encontrei as melhores respostas para questões intrínsecas à condição humana. Aprendo e quero ser um eterno aprendiz de seus conhecimentos. Na figura da minha amada mãe Nadir Fülber Garcia e da minha querida amiga Camila Antunes de Oliveira, faço uma exaltação a todas as mulheres, em especial àquelas presentes em minha vida.

Aos irmãos, agradeço simplesmente por existirem. Apesar de o sangue não ser o motivo do vínculo entre nós, o amor que nutrimos nos une de forma absoluta. Em um universo caótico, existiam diversos cenários em que nunca nos encontraríamos. Porém, contra todas essas possibilidades, vocês se tornaram pessoas indispensáveis em minha vida. Obrigado Maurício Dal Pozzo Schneider e Guilherme de Freitas Gaiardo.

Por fim, e como não poderia deixar de ser, agradeço aos meus pais, Nadir Fülber Garcia e Celso Oscar de Souza Garcia. Vocês foram motivo e finalidade, vocês foram base e cobrança, vocês foram certezas e dúvidas, vocês foram e são exatamente como pais perfeitos devem ser! Amo vocês de todo o meu coração!

RESUMO

O paradigma de Virtualização de Funções de Rede (*Network Function Virtualization* - NFV) visa desacoplar as funções de rede de hardware dedicado, utilizando tecnologias de virtualização para implementar as funções em software. A arquitetura de referência NFV tem sido amplamente adotada, sendo esta composta por três domínios: Infraestrutura Virtualizada (*Virtualized Infrastructure* - VI), Gerenciamento e Orquestração de NFV (*NFV Management and Orchestration* - NFV-MANO), além das próprias Funções Virtualizadas de Rede (*Virtualized Network Functions* – VNF). Já serviços virtualizados são definidos como composições de funções virtualizadas, organizados através de topologias que estabelecem o fluxo de processamento do tráfego de rede. Apesar do grande potencial do paradigma NFV, ainda existem importantes desafios para garantir que seja amplamente adotado nas infraestruturas modernas de telecomunicação. Esta Tese tem como objetivo principal contribuir para a gerência do ciclo de vida de funções e serviços virtualizados de rede. Neste sentido assume-se a premissa de que é essencial garantir previsibilidade operacional e padronização dos elementos que atuam no domínio de VNF, *i.e.*, das plataformas de execução de VNF e dos Sistemas de Gerenciamento de Elemento (*Element Management System* – EMS). São propostas na Tese arquiteturas que descrevem um modelo de execução e gerenciamento padronizado tanto para plataformas de execução de VNF, como para o EMS, compatíveis com a arquitetura de referência NFV-MANO. As arquiteturas propostas preveem protocolos, interfaces de comunicação e orientações de integração com demais elementos e sistemas NFV. Ambas as arquiteturas foram implementadas e estão disponíveis como software livre: a plataforma COVEN de execução de VNF e o EMS HoLMES. Resultados de avaliação experimental e estudos de caso são apresentados e discutidos. Também, uma investigação do impacto das arquiteturas propostas em três contextos distintos é então introduzida. Inicialmente, é explorado seu impacto como facilitador para a gerência de NFV. Em seguida, o foco fica no compartilhamento de instâncias de funções e serviços de rede, e na emulação NFV. A Tese apresenta também contribuições no contexto de serviços virtualizados de rede. Em particular, o mapeamento multidomínio consiste na implantação de serviços em infraestruturas de virtualização distribuídas. Assim, foi proposta uma estratégia de mapeamento baseado em heurísticas genéticas, denominada GeSeMa. A estratégia permite que seus operadores definam diferentes critérios de avaliação e otimização em busca de mapeamentos de serviço adequados às suas necessidades específicas. Neste sentido, o GeSeMa representa um avanço do estado da arte, pois outras soluções operam de maneira monolítica, não permitindo que operadores e gerentes de rede personalizem o esquema de otimização adotado (*e.g.*, métricas a serem avaliadas, pesos a serem considerados e restrições de mapeamento relacionadas a cada serviço em particular). A solução foi testada em múltiplos estudos de caso e os resultados demonstram sua aplicabilidade e flexibilidade ao lidar com diferentes cenários de otimização de mapeamentos multidomínio. Finalmente, outras duas contribuições da Tese relacionadas ao paradigma NFV são apresentadas como apêndices, nos contextos de tolerância a falhas e engenharia de tráfego.

Palavras-chave: NFV. VNF. EMS. Gerenciamento. Implantação. Mapeamento.

ABSTRACT

The Network Function Virtualization (NFV) paradigm aims to decouple network functions from dedicated hardware, employing virtualization technologies to implement functions in software. The NFV reference architecture has been widely adopted. This architecture, in turn, is composed of three domains: Virtualized Infrastructure (VI), NFV Management and Orchestration (NFV-MANO), and Virtualized Network Functions (VNF). Compositions of virtualized functions define virtualized services, organized as topologies through which network traffic is steered. Despite the extraordinary potential of the NFV paradigm, there are still relevant challenges to ensure its wide adoption by modern telecommunication infrastructures. The main objective of this Thesis is to contribute to the life cycle management of virtualized network functions and services. In this context, we consider it essential to guarantee the operational predictability and organization of the elements of the VNF domain, *i.e.*, VNF execution platforms, and the Element Management System (EMS). Thus, this Thesis presents architectures that describe a standardized execution and management model for both VNF execution platforms and EMS. The proposed architectures are compliant with the NFV-MANO reference architecture and provide protocols, communication interfaces, and guidelines for their integration with other NFV elements and systems. Both architectures have been implemented and are available as open-source software: the COVEN VNF execution platform and the HoLMES EMS. Experimental evaluation results and case studies are presented and discussed. An investigation of the impact of the proposed architectures in three different contexts is also introduced. First, we explore the opportunities regarding the proposed architectures as enablers for NFV management. Then, the focus goes to sharing instances of network functions and services and NFV emulation. In addition, the Thesis presents contributions in the context of virtualized network services. Multidomain mapping consists of deploying services on distributed virtualization infrastructures. A mapping strategy based on genetic heuristics, called GeSeMa, was proposed. GeSeMa enables its operators to define multiple optimization criteria for searching candidate service mappings tailored to their specific requirements. The proposed strategy represents a state-of-the-art advance, as other solutions operate in a monolithic manner: they do not allow operators and network managers to customize the adopted evaluation setup (*e.g.*, metrics to be evaluated, weights to be considered, and mapping constraints related to each specific service). We tested GeSeMa in multiple case studies; the results demonstrate its applicability and flexibility in dealing with different multidomain mapping optimization scenarios. Finally, two other contributions related to the NFV paradigm, approaching fault tolerance and traffic engineering, are presented as appendices.

Keywords: NFV. VNF. EMS. Management. Deployment. Mapping.

LISTA DE FIGURAS

2.1 Modelos de Infraestrutura de Rede 24
2.2 Arquitetura NFV Simplificada 25
2.3 Representação em Alto Nível de VNF 26
2.4 Categorias e Classes de EMS 28
2.5 Arquitetura SFC Simplificada. 30

3.1 Arquitetura Geral do ClickOS 34
3.2 Arquitetura Geral do Click-On-OSv 34
3.3 Arquitetura Geral do Leaf. 35
3.4 Arquitetura Geral do SampleVNF 36
3.5 Arquitetura Geral do NFF-Go 36
3.6 Arquitetura Geral do NetVM 37
3.7 Arquitetura Geral do OpenNetVM 37
3.8 Arquitetura Geral do HyperNF 39
3.9 Arquitetura Geral do MVMP 39
3.10 Arquitetura Geral do CliMBOS. 40
3.11 Arquitetura Geral do SafeLib 40
3.12 Fluxo de Mensagens Abstraído do Cloud-Init 43
3.13 Fluxo de Mensagens Abstraído do Day-Like 43
3.14 Fluxo de Mensagens Abstraído do EMS do Open Baton 43
3.15 Fluxo de Mensagens Abstraído do EMS do CloudStack Vines. 44
3.16 Arquitetura Geral do EMS do CloudStack Vines. 44

4.1 Arquitetura de Plataformas de Execução de VNF 51
4.2 Modelos de Tráfego de Rede e Funções de Rede Suportados pela Arquitetura . . 55
4.3 Processo de Configuração de Plataformas de Execução de VNF. 56
4.4 Topologia de Serviço DeMONS 60
4.5 Topologia de Serviço do DeMONS + NSH. 61
4.6 Tempo para Consulta de Reputação DeMONS (FW). 62
4.7 Tempo de Processamento por Pacote DeMONS (FW) 63
4.8 *Firewall* de Camada Sete Baseado em Componentes. 65
4.9 RTT das Possíveis Combinações de Componentes PA e PF 66

5.1 Arquitetura do Elemento Operacional de EMS. 70
5.2 Entrada e Saída de Dados no EMS 74

5.3 Operação de Configuração no EMS . . . 75
5.4 Operação de VNF no EMS . . . 76
5.5 Ambiente de Execução para Testes de EMS . . . 80
5.6 Distribuição - *Get Status* . . . 80
5.7 Média - *Get Status* . . . 80
5.8 CPDF - *Get Status*. . . . 80
5.9 Distribuição - *Post NF* . . . 81
5.10 Média - *Post NF*. . . . 81
5.11 CPDF - *Post NF*. . . . 81
5.12 Distribuição - *Post Start*. . . . 82
5.13 Média - *Post Start* . . . 82
5.14 CPDF - *Post Start* . . . 82

6.1 Representação em Dimensões das Características de Monitores . . . 87
6.2 Tempo de Resposta para a Obtenção de Diferentes Métricas. . . . 90
6.3 Definição Simplificada de Topologia de Serviço com Compartilhamento . . . 92
6.4 Cenário de Testes de Compartilhamento de VNF . . . 94
6.5 Tempo de Processamento da Função de Rede por Serviço Cadastrado. . . . 95
6.6 Exemplo de Ambiente NFV do NIEP. . . . 97
6.7 Cenário de Testes para Emulação em NFV . . . 99
6.8 Execução de Operações de Gerenciamento em Cenários Emulados e Não-emulados 100

7.1 Modelo de Requisição GeSeMa . . . 104
7.2 Sumarização do Processamento GeSeMa. . . . 107
7.3 Conv. (7 VNFs) . . . 109
7.4 Conv. (9 VNFs) . . . 109
7.5 Conv. (11 VNFs) . . . 109
7.6 Tempo de Execução do GeSeMa (7 VNFs). . . . 110
7.7 Tempo de Execução do GeSeMa (9 VNFs). . . . 110
7.8 Tempo de Execução do GeSeMa (11 VNFs) . . . 110
7.9 Comparação de Fronteiras (7 VNFs) . . . 112
7.10 Comparação de Fronteiras (9 VNFs) . . . 112
7.11 Comparação de Fronteiras (11 VNFs) . . . 112
7.12 Comparação do Tempo de Execução (7 VNFs). . . . 112
7.13 Comparação do Tempo de Execução (9 VNFs). . . . 112
7.14 Comparação do Tempo de Execução (11 VNFs) . . . 112
7.15 Comparação de Fronteiras (Genéticos) . . . 114
7.16 Comparação do Tempo de Execução (Genéticos) . . . 114

A.1 Arquitetura FIT-SFC Simplificada 132
A.2 Cenário de Impossibilidade de Quórum (Falhas Bizantinas) 132
A.3 Cenário de Conluio com Cliente Malicioso 133
A.4 Atraso Decorrente da Arquitetura FIT-SFC 134
A.5 Atraso em Cenários com Falhas por Parada e Intrusão 135
B.1 Fluxograma de Execução do srTBP *Policer* 138
B.2 Fluxograma de Execução do srTCM *Policer* 139
B.3 Fluxograma de Execução do trTCM *Policer* 140
B.4 Fluxograma de Execução do srTCM *Policer Color-Aware* 141
B.5 Fluxograma de Execução do trTCM *Policer Color-Aware* 141
B.6 Fluxograma de Execução do srTBS *Shaper* 142
B.7 Fluxograma de Execução do LB *Shaper* 143
B.8 srTBP (*Bucket* 4000) 145
B.9 LB (Taxa Constante) 145
B.10 srTBP (10Mbps) 146
B.11 LB (10Mbps) 146

LISTA DE TABELAS

3.1 Sumarização das Plataformas de Execução de VNF 41
3.2 Sumarização das Soluções de EMS e Serviços Similares 44
3.3 Sumarização de Características das Soluções de Mapeamento 46

4.1 Transições de Estados de Plataformas de Execução de VNF 57

7.1 Modificação da Função Objetivo 110
7.2 Número de Candidatos Recuperados das Soluções de Mapeamento 112

B.1 Perfis de Tráfego de Rede 144

LISTA DE ACRÔNIMOS

AA	*Authentication Agent*
AIf	*Access Interface*
AMQP	*Advanced Message Queuing Protocol*
AS	*Access Subsystem*
BH	*Base Header*
BSS	*Bussiness Support System*
CGNAT	*Carrier Grade Network Address Translate*
CH	*Context Header*
CM	*Configuration Module*
CMR	*Click Modular Router*
CoAP	*Constrained Application Protocol*
COVEN	*COmprehensive VirtualizEd NF*
CPDF	*Cummulative Probability Flow Diagram*
CR	*Channel Router*
CRUD	*Create, Read, Update, Delete*
DDoS	*Distributed Denial of Service*
DeMONS	*DDoS MitigatiOn NFV Solution*
DINF	Departamento de Informática
DM	*DeMONS Manager*
DPDK	*Data Plane Development Kit*
DPI	*Deep Packet Inspector*
DRL	*Deep Reinforcement Learning*
EA	*Extended Agent*
EMIB	EMS *Management Information Base*
EMS	*Element Management System*
EsCAPE	*Extensible Service Chain Prototyping Environment*
ETSI	*European Telecommunications Standards Institute*
FA	*Flow Allocator*
FIT-SFC	*Fault- & Intrusion-Tolerant SFC*
FTC	*Fault Tolerant Chaining*
FW	*Firewall*
GbE	*Gigabit Ethernet*
GeSeMa	*Genetic Service Mapping*
HoLMES	*Holistic, Lightweight and Malleable EMS Solution*
HTTP	*Hypertext Transfer Protocol*

HUC	*Half Uniform Crossover*
IETF	*Internet Engineering Task Force*
IDS	*Intrusion Detection System*
ILP	*Integer Linear Programming*
IO	*Input-Output*
IoT	*Internet of Things*
IPS	*Intrusion Prevetion System*
IR	*Internal Router*
ITF	*Internal Traffic Forwarder*
JNI	*Java Native Interface*
JSON	*JavaScript Object Notation*
LB	*Leaky Bucket*
MA	*Management Agent*
MeDICINE	*Multi Datacenter Service Chain Emulator*
MIf	*Monitoring Interface*
MS	*Monitoring Subsystem*
NAT	*Network Address Translator*
NF	*Network Function*
NFF-Go	*Network Function Framework for Go*
NFV	*Network Function Virtualization*
NFV-MANO	*NFV Management and Orchestration*
NFV-RA	NFV *Resource Allocation*
NFV-TE	NFV *Traffic Engineering*
NFVaaS	*NFV-as-a-Service*
NFVO	*NFV Orchestrator*
NIEP	*NFV Infrastructure Emulation Platform*
NS	*Network Service*
NSGAII	*Nondominated Sorting Genetic Algorithm II*
NSH	*Network Service Header*
NSHP	*NSH Processor*
OA	*Operation Agent*
OPNFV	*Open Platform for NFV*
OSM	*Open Source MANO*
OSS	*Operation Support System*
PA	*Packet Analyzer*
PC	*Priority Classifier*
PF	*Packet Filter*
PMC	*Partially Mapped Crossover*
PNF	*Physical Network Function*

PPGINF	Programa de Pós-Graduação em Informática
PPS	*Packet Processing Subsystem*
QoS	*Quality of Service*
REACH	*REliability-Aware service CHaining*
REST	*Representational State Transfer*
RTT	*Round Trip Time*
SBX	*Simulated Binary Crossover*
SCAG	*Service ChAin Grammar*
SDN	*Software-Defined Network*
SF	*Service Function*
SFC	*Service Function Chain*
SFF	*Service Function Forwarder*
SGX	*Software Guard Extensions*
SI	*Service Index*
SPH	*Service Path Header*
SPEA2	*Strength Pareto Evolutionary Algorithm 2*
SPI	*Service Path Identifier*
srTBP	*Single Rate Token Bucket Policer*
srTBS	*Single Rate Token Bucket Shaper*
srTCM	*Single Rate Three Color Marker*
SSC	*Subtour Selection Crossover*
ST	*Service Topology*
TP	*Traffic Policer*
trTCM	*Two Rate Three Color Marker*
TTL	*Time To Live*
UDP	*User Datagram Protocol*
UFPR	Universidade Federal do Paraná
VACL	*VLAN Access Control List*
VALE	*VirtuAl Local Ethernet*
VEM	*Virtualized Elements Manager*
VI	*Virtualized Infrastructure*
VIf	*VNF Interface*
VIM	*VI Manager*
VNF	*Virtualized Network Function*
VNFC	*VNF Component*
VNFM	*VNF Manager*
VNFP	*VNF Package*
VNS	*Virtual Network Subsystem*
VS	VNF *Subsystem*

YAML *YAML Ain't Markup Language*

SUMÁRIO

1 INTRODUÇÃO 17
2 O PARADIGMA DE VIRTUALIZAÇÃO DE FUNÇÕES DE REDE 23
2.1 NFV EM SÍNTESE 23
2.2 DOMÍNIO DE TRABALHO VNF 25
2.3 SERVIÇOS VIRTUALIZADOS DE REDE 29
2.4 SUMARIZAÇÃO E DISCUSSÃO 31
3 TRABALHOS RELACIONADOS 33
3.1 PLATAFORMAS DE EXECUÇÃO DE VNF 33
3.2 IMPLEMENTAÇÕES DO EMS 41
3.3 SOLUÇÕES DE MAPEAMENTO MULTIDOMÍNIO 45
3.4 SUMARIZAÇÃO E DISCUSSÃO 47
4 ARQUITETURA DE PLATAFORMAS DE EXECUÇÃO DE VNF 50
4.1 DEFINIÇÕES ARQUITETURAIS E OPERACIONAIS. 50
4.2 IMPLEMENTAÇÃO DO PROTÓTIPO COVEN 57
4.2.1 Interconexão dos Módulos Operacionais 59
4.3 EXPERIMENTAÇÃO E RESULTADOS 59
4.3.1 Estudo de Caso: Mitigação de Ataques DDoS DeMONS 60
4.3.2 Estudo de Caso: Implantação de Função de Rede com Múltiplos Componentes . 64
4.4 SUMARIZAÇÃO E DISCUSSÃO 67
5 ARQUITETURA DO ELEMENTO OPERACIONAL EMS. 69
5.1 DEFINIÇÕES ARQUITETURAIS E OPERACIONAIS. 69
5.2 IMPLEMENTAÇÃO DO PROTÓTIPO HOLMES 77
5.2.1 Interconexão dos Módulos Operacionais 78
5.3 EXPERIMENTAÇÃO E RESULTADOS 79
5.3.1 Estudo de Caso: Sobrecarga de Tempo de Operação 79
5.4 SUMARIZAÇÃO E DISCUSSÃO 83
6 CENÁRIOS DE APLICAÇÃO DAS ARQUITETURAS DE VNF E EMS . . 86
6.1 MONITORAMENTO ABRANGENTE DE VNF 86
6.1.1 Experimentação e Resultados 88
6.2 COMPARTILHAMENTO DE INSTÂNCIAS DE VNF ENTRE CLIENTES . . . 91
6.2.1 Experimentação e Resultados 93
6.3 EMULAÇÃO DE AMBIENTES NFV 96
6.3.1 Experimentação e Resultados 98
6.4 SUMARIZAÇÃO E DISCUSSÃO 101

7 MAPEAMENTO PERSONALIZADO DE SERVIÇOS DE REDE BASEADO EM HEURÍSTICAS GENÉTICAS 103
7.1 HEURÍSTICAS GENÉTICAS 103
7.2 *GENETIC SERVICE MAPPING* 104
7.2.1 Modelo de Requisição 104
7.2.2 Metodologia Genética de Mapeamento Multidomínio 105
7.3 EXPERIMENTAÇÃO E RESULTADOS 107
7.3.1 Mapeamento de Caches Multimídia Hierárquicas 108
7.3.2 Comparação Entre GeSeMa e GA+LCB 113
7.4 SUMARIZAÇÃO E DISCUSSÃO 115
8 CONCLUSÃO 116
REFERÊNCIAS 119
APÊNDICE A – ARQUITETURA SFC TOLERANTE A FALHAS E INTRUSÃO. 130
A.1 FIT-SFC: ARQUITETURA. 131
A.1.1 Experimentação e Resultados Preliminares. 133
A.1.2 Considerações Finais 135
APÊNDICE B – ENGENHARIA DE TRÁFEGO BASEADA EM NFV . . 136
B.1 O *FRAMEWORK* NFV-TE 137
B.1.1 *Policers* 137
B.1.2 *Shapers* 141
B.2 EXPERIMENTAÇÃO E RESULTADOS 143
B.3 CONSIDERAÇÕES FINAIS 146
APÊNDICE C – PUBLICAÇÕES E SUBMISSÕES 147
C.1 TRABALHOS NO ÂMBITO DA TESE 147
C.2 TRABALHOS NO ÂMBITO DE OUTRAS PESQUISAS 148

1 INTRODUÇÃO

Infraestruturas de comunicação, atualmente, dependem de uma variedade de funções de rede para operarem de maneira adequada (*e.g.*, roteadores, balanceadores de carga, filtros de tráfego, servidores de nomes, mitigadores de ataques, entre outros). Funções de rede são tradicionalmente implementadas em equipamentos proprietários e dedicados, chamados de *physical appliances* ou *middleboxes* (Resma et al., 2019). A utilização desses equipamentos – que consistem de *hardware* dedicado – costuma resultar em um excelente desempenho no processamento de tráfego de rede. Entretanto, a adoção de equipamentos dedicados também resulta em limitações de gerenciamento, mobilidade e atualização (Sherry et al., 2012). Exemplos práticos dessas limitações são a restrição pelo uso de sistemas de gerenciamento compatíveis com protocolos proprietários, dificuldade da migração física de equipamentos, retenção de mercado devido à necessidade de produção conjunta de *hardware* e *software*, e elevação dos custos operacionais e de capital para manutenção/atualização da infraestrutura. Essas limitações escalam rapidamente dada a crescente necessidade de expansão das redes de computadores e de otimização da qualidade de serviço e experiência para os seus usuários.

As redes virtualizadas surgem como uma alternativa promissora para solucionar diversos desafios das redes de computadores, até mesmo a própria ossificação da Internet (Handley, 2006). Entre os principais representantes de virtualização estão os paradigmas de Redes Definidas por Software (*Software-Defined Networks* - SDN) e de Virtualização de Funções de Rede (*Network Function Virtualization* - NFV). O paradigma SDN separa o plano de controle do plano de dados de uma rede de computadores: o plano de dados fica assim dedicado exclusivamente ao encaminhamento de pacotes, enquanto o plano de controle realiza funções de gerenciamento em um nível superior (Benzekki et al., 2016). Já o paradigma NFV desacopla as funções de rede de seu *hardware* associado. Assim, as funções são implementadas em *software* e executadas em equipamentos genéricos utilizando tecnologias de virtualização amplamente disponíveis (*e.g.*, virtualização completa, paravirtualização ou conteinerização) (Yi et al., 2018). Os paradigmas de redes virtualizadas apresentados são capazes de coexistir e cooperar entre si, porém é importante ressaltar que estes são independentes e autocontidos, podendo ser adotados individualmente.

O paradigma NFV vem sendo amplamente explorado tanto na academia quanto na indústria. Organizações como o Instituto Europeu de Padrões de Telecomunicações (*European Telecommunications Standards Institute* - ETSI) e a Força-tarefa de Engenharia da Internet (*Internet Engineering Task Force* - IETF) proveem uma série de documentos contendo definições e recomendações relacionadas ao paradigma NFV. Entre os documentos mais relevantes destaca-se aquele que define a arquitetura de referência do paradigma (NFVISG, 2014), a qual é composta de três domínios de trabalho: Infraestrutura Virtualizada (*Virtualized Infrastructure* - VI), Gerência e Orquestração NFV (*NFV Management and Orchestration* - NFV-MANO) e Função de Rede Virtualizada (*Virtualized Network Function* - VNF). Esses domínios contêm uma série de elementos operacionais responsáveis pela instanciação, execução e manutenção de funções e serviços de rede dentro de um ambiente NFV. Entretanto, a arquitetura de referência limita-se a descrever definições gerais de competências e interfaces dos elementos operacionais por ela especificados. Sendo assim, trabalhos complementares apresentam detalhes de implementação e expandem a arquitetura NFV de forma a satisfazer necessidades práticas de aplicações distintas, além de adaptá-la para áreas específicas das telecomunicações (Venâncio et al., 2021) (Venâncio et al., 2020) (Fulber-Garcia et al., 2019a) (Jaeger, 2015).

Além de habilitar funções de rede individuais, atividades complexas podem ser realizadas em NFV através da execução conjunta de múltiplas funções de rede, ou seja, um Serviço de Rede (*Network Service* - NS). Nesse caso, as funções são dispostas em uma topologia de serviço que, em conjunto com múltiplos metadados, gera uma estrutura conhecida como Cadeia de Função de Serviço (*Service Function Chain* - SFC) (Fulber-Garcia et al., 2020a). A arquitetura de referência SFC é proposta pela IETF e considera um conjunto de elementos operacionais e protocolos de comunicação capazes de encaminhar o tráfego de rede através de uma topologia de serviço (Halpern e Pignataro, 2015). O paradigma NFV é autossuficiente em relação à criação, implantação e manutenção de serviços de rede, utilizando para isso a arquitetura SFC junto ao Cabeçalho de Serviço de Rede (*Network Service Header* - NSH) (Quinn et al., 2018). Porém, é também comum que serviços virtualizados sejam organizados por controladores de rede e *switches* virtuais que empregam o protocolo OpenFlow (Hu et al., 2014), sendo assim enquadrados no paradigma SDN (Kaur et al., 2020).

Um dos principais desafios relacionados ao paradigma NFV diz respeito ao gerenciamento de funções e serviços de rede. Nesse contexto, técnicas e ferramentas precisam ser desenvolvidas no sentido de facilitar, flexibilizar e, até mesmo, automatizar ações de gestão e manutenção da infraestrutura de rede virtualizada. A arquitetura de referência NFV dispõe de uma série de elementos operacionais, como o (i) Orquestrador NFV (*NFV Orchestrator* - NFVO), (ii) Gerente de VNF (*VNF Manager* - VNFM), (iii) Gerente de Infraestrutura Virtualizada (*Virtualized Infrastructure Manager* - VIM), (iv) Sistema de Suporte a Operações e Serviços (*Operation and Bussiness Support System* - OSS/BSS) e (v) Sistema de Gerenciamento de Elemento (*Elemenet Management System* - EMS). Os três primeiros elementos citados pertencem ao domínio NFV-MANO, o quarto elemento é executado externamente aos domínios da arquitetura NFV e o quinto elemento pertence ao domínio VNF. Além desses elementos operacionais, é indispensável ressaltar que Funções de Rede Virtualizadas (*Virtualized Network Functions* - VNF) devem também prover uma interface para a execução de operações de gerenciamento interno, a ser utilizada tanto pelo EMS quanto pelo VNFM.

Em relação aos elementos de gerência, uma VNF consiste em um código capaz de processar tráfego de rede (*i.e.*, uma função de rede) integrado à plataforma que a executa, que neste trabalho chamamos plataforma de execução de VNF (Fulber-Garcia et al., 2019b). Exemplos dessas plataformas de execução de VNF incluem *e.g.*, ClickOS (Martins et al., 2014), Click-On-OSv (Marcuzzo et al., 2017) e OpenNetVM (Zhang et al., 2016b). A operacionalização de uma VNF através de tecnologias de virtualização cria uma instância de VNF. Observa-se que, nesse caso, o interfaceamento e a execução de operações de gerenciamento interno à VNF são exercidos pela plataforma de execução. O EMS, como definido nos documentos originais, é destinado ao gerenciamento de uma ou múltiplas instâncias de VNF, sendo capaz de abstrair e padronizar operações de gerenciamento interno destas através de um protocolo padrão reconhecido pelo VNFM e OSS/BSS. Além disso, o EMS realiza diferentes categorias de monitoramento nas instâncias de VNF, disparando alarmes e alertas quando necessário. O OSS/BSS é um sistema externo aos domínios da arquitetura NFV e contempla operações relacionadas, por exemplo, a interpretação e aplicação de políticas de serviço e negociação de funções de rede e/ou instâncias de VNF. O VIM dedica-se ao gerenciamento do domínio VI, atuando em conjunto com hipervisores e gerentes de recursos na criação e manutenção de instâncias de VNF. O VNFM comunica-se com o domínio VNF, executando operações de ciclo de vida diretamente nas instâncias de VNF ou através de seus EMS. Por fim, o NFVO é responsável pelo gerenciamento de mais alto nível do ambiente NFV, seja de recursos computacionais, de funções de rede virtualizadas ou de serviços de rede. Esse elemento também provê a interface de comunicação entre o domínio NFV-MANO e os operadores externos à arquitetura NFV.

Considerando os elementos de gerência internos aos domínios de trabalho da arquitetura de referência NFV, aqueles presentes no domínio NFV-MANO vêm sendo amplamente explorados em trabalhos recentes. Entre esses trabalhos destacam-se os projetos relacionados ao desenvolvimento de sistemas que implementam o NFV-MANO, como Open Source Mano (OSM) (ETSI, 2021), Open Baton (Berlin, 2021), OpenStack Tacker (OpenStack, 2021) e CloudStack Vines (Flauzino et al., 2020), além de propostas de especificações arquiteturais internas e protocolos de gerência para esses elementos (Venâncio et al., 2021) (Yousaf et al., 2019) (Gonzalez et al., 2018). Entretanto, diferente do que ocorre com o domínio NFV-MANO, o domínio VNF é marginalmente explorado em termos de seus elementos operacionais. No modelo de referência original, tanto o elemento VNF, quanto o de EMS não contam com uma arquitetura interna especificada, sendo livremente construídos sem nenhuma padronização de funcionamento. Esse tipo de desenvolvimento cria diversos obstáculos do ponto de vista de gerenciamento, uma vez que as interfaces e os módulos internos desses elementos são imprevisíveis e apresentam comportamentos completamente heterogêneos entre suas diferentes implementações. Esse é o problema tratado nesta Tese.

Como resultado do cenário apresentado, sistemas que implementam o NFV-MANO tipicamente se restringem a gerenciar o ciclo de vida de instâncias de VNF através de suas máquinas virtuais e contêineres (*e.g.*, criar instância, ligar/desligar instância, modificar configurações de máquina e escalar recursos computacionais). Já o gerenciamento de ciclo de vida interno de uma instância de VNF (*e.g.*, alocar uma função de rede, iniciar/parar a execução da função e recuperar métricas de monitoramento da plataforma de execução), quando disponível, é normalmente limitado às plataformas de execução de VNF que, no máximo, implementam agentes de gerência dedicados a sistemas NFV-MANO específicos. Além disso, o EMS é quase sempre negligenciado por esses mesmos sistemas NFV-MANO. Estendendo a discussão para o elemento OSS/BSS, que é tipicamente limitado a prover compatibilidade de operações com plataformas de execução de VNF previamente configuradas, também negligenciando a possível existência de um EMS no ambiente NFV (Bondan et al., 2019) (Xilouris et al., 2014). De forma geral, as insuficiências apresentadas, além de culminarem em dificuldades de gerenciamento do ambiente NFV, infringem dois fundamentos deste paradigma (NFVISG, 2013): portabilidade (plataformas e sistemas devem ser intercambiáveis e cooperarem para proverem ambientes NFV) e integração (funções e serviços devem integrar-se em um ambiente NFV independentemente das tecnologias adotadas e de seus fornecedores).

Nesse contexto, esta Tese propõe uma especificação arquitetural para os elementos operacionais do domínio de trabalho VNF da arquitetura de referência NFV. Com isso, duas arquiteturas são propostas, a primeira aplica-se às plataformas de execução de VNF, e a segunda ao elemento operacional EMS. Essas arquiteturas estabelecem, sobretudo, as interfaces de comunicação e os módulos operacionais internos dos elementos, descrevendo explicitamente suas competências e sugerindo modelos de desenvolvimento. Ambas as arquiteturas propostas são completamente compatíveis à arquitetura de referência NFV, adotando, sempre que disponíveis, os protocolos de comunicação definidos pela ETSI. Através dessas especificações arquiteturais, três objetivos principais são almejados: (i) diferenciar e estabelecer padrões para os planos de dados e de gerência dos elementos operacionais do domínio VNF; (ii) viabilizar o elemento EMS para tratamento e abstração da heterogeneidade de implementações de plataformas de execução de VNF, de NFV-MANO e de OSS/BSS; e (iii) aprimorar a interoperabilidade de gerência entre os elementos operacionais da arquitetura NFV ETSI e as plataformas de execução de VNF, satisfazendo assim os fundamentos de portabilidade e integração.

Além da especificação arquitetural dos elementos operacionais do domínio VNF, este trabalho também objetiva fazer demonstrações práticas destas arquiteturas através do

desenvolvimento e avaliação de protótipos. Para realizar as experimentações, operações típicas, porém com características inovadoras são utilizadas e propostas. Exemplos dessas operações são: monitoramento de métricas particulares a uma função de rede sendo executada em uma plataforma de execução de VNF; compartilhamento de funções de rede entre diferentes serviços; e emulação de funções e serviços virtualizados.

Porém, as limitações de gerenciamento no contexto do paradigma NFV não se restringem às funções de rede. Diversos processos vinculados a serviços virtualizados de rede também apresentam insuficiências nesse contexto. Entre tais processos, destaca-se a implantação de serviços no substrato físico disponível. A implantação pode ser resumida em três tarefas principais (Herrera e Botero, 2016): composição, integração e programação de execução. Apesar de todas as tarefas desempenharem um papel fundamental na operacionalização de um serviço virtualizado de rede, aquela que tipicamente mais se destaca pela complexidade e heterogeneidade de cenários de execução é a integração (Santos et al., 2022). A tarefa de integração é responsável pela alocação das funções de rede de um serviço em pontos de presença e servidores de execução específicos, sendo esta dividida em três técnicas principais: mapeamento multidomínio; seleção de funções de rede; e alocação em servidores. Em particular, a técnica de mapeamento multidomínio visa distribuir de forma otimizada um serviço de rede através de diferentes domínios de execução existentes. Por mais que represente uma necessidade frequente, essa técnica ainda é pouco explorada, principalmente quando comparada à técnica de alocação em servidores.

Atualmente, soluções de mapeamento multidomínio se restringem a otimizar funções objetivo predefinidas em código-fonte, sendo estas monolíticas e pouco personalizáveis no que diz respeito ao seu modelo de operação e avaliação. Com isso, apesar de obterem bons resultados em vista das metodologias de otimização adotadas, estas podem não representar exatamente as necessidades dos cliente, operadores, e gerentes de um serviço virtualizado a ser mapeado em uma infraestrutura distribuída. Para que essas necessidades sejam completamente atendidas, então, as entidades interessadas podem optar pelo desenvolvimento de uma nova solução, que incorre em diferentes custos, ou na adaptação de uma solução já existente, que nem sempre é possível devido a restrições técnicas. Não sendo viáveis tais alternativas, a incapacidade de gerenciar em tempo de configuração a função objetivo (ou seus respectivos modelos de otimização) das soluções de mapeamento pode resultar em perdas significativas de desempenho e de qualidade de serviço.

Nesse contexto, esta Tese apresenta uma nova solução de mapeamento multidomínio em NFV, chamada *Genetic Service Mapping* (GeSeMa). A solução GeSeMa permite que a configuração da avaliação seja completamente personalizada, proporcionando alta flexibilidade para se adaptar às diferentes necessidades das várias entidades participantes de um processo de implantação por mapeamento multidomínio, além de suportar outros importantes aspectos técnicos, como o mapeamento de serviços com topologia ramificada e com restrições de localização física. Para isso, as partes interessadas descrevem suas necessidades e outras características do serviço em um documento de requisição padrão. O GeSeMa então usa uma metaheurística de otimização multi-objetivo baseada em algoritmos genéticos para encontrar candidatos de mapeamento em tempo viável. Avaliamos a solução proposta por meio de estudos de caso abrangentes, incluindo comparações com estratégias clássicas alternativas e soluções estado da arte. Dessa forma, três objetivos são buscados: (i) analisar possíveis metodologias de personalização da função objetivo de um algoritmo de mapeamento multidomínio para serviços NFV; (ii) identificar metaheurísticas adequadas para a criação de soluções personalizáveis para mapeamento multi-domínio; e (iii) desenvolver uma solução funcional de se adapte às necessidades de seus usuários para a execução da tarefa de integração.

Também são apresentadas, como apêndices, outras contribuições para o paradigma NFV realizadas no contexto desta Tese. Essas contribuições consistem na proposição de uma

arquitetura para a criação e manutenção de serviços virtualizados de rede tolerante a falhas bizantinas, além de um *framework* destinado ao desenvolvimento de funções virtualizadas relacionadas a engenharia de tráfego de rede (desenvolvido através de um projeto de conclusão de curso coorientado pelo autor desta Tese).

Em síntese, as contribuições deste trabalho são: (i) especificação arquitetural dos elementos operacionais do domínio VNF da arquitetura de referência NFV; (ii) implementação e disponibilização de protótipos para avaliação das arquiteturas propostas; (iii) demonstração documentada dos protótipos desenvolvidos na execução de operações típicas, porém com características inovadoras; (iv) determinação de modelos de definição e avaliação de funções objetivo personalizáveis para a tarefa de integração em NFV; (v) desenvolvimento de uma solução parametrizável para a execução da técnica de mapeamento multidomínio de serviços virtualizados de rede; (vi) demonstração documentada da solução de mapeamento desenvolvida em diferentes estudos de caso; e (vii) proposição ampla e disponibilização de algoritmos, soluções, modelos e protocolos relacionados ao paradigma NFV.

Destaca-se, finalmente, que as contribuições desta Tese objetivam não apenas apresentar aspectos tecnológicos inovadores, mas também delinear oportunidades de pesquisa que tenham impactos significativos em curto/médio prazo no contexto socioeconômico brasileiro.

No cenário educacional, evidencia-se as oportunidades de uso do paradigma NFV na criação e manutenção de redes locais para fornecimento conteúdos didáticos. Segundo levantamento de 2019 (Dino e Costa, 2021), apesar de 99% das escolas públicas urbanas brasileiras contarem com acesso à Internet, apenas 63% destas permitem o uso da rede por parte do corpo discente. O desafio é ainda maior quando apresentado no contexto rural, onde apenas 40% das escolas públicas contam com acesso à Internet. Através do paradigma NFV, pequenos computadores de placa única (*e.g.*, *Raspberry Pi* (Richardson e Wallace, 2012)) podem ser adaptados para criar infraestruturas de redes locais e fornecer conteúdos didáticos em um perímetro escolar. Tal abordagem não necessita de acesso contínuo à Internet, demandando o mesmo apenas para a realização de atualizações de software e conteúdo nos equipamentos. Ressalta-se, porém, que tal estratégia não objetiva substituir o necessário acesso à Internet nos ambientes de ensino, mas sim mitigar a ausência do mesmo e alfabetizar tecnologicamente alunos de áreas desconectadas.

Outra oportunidade consiste na utilização do paradigma NFV para aprimorar e facilitar a construção de redes em ambientes conflituosos ou catastróficos. Como previamente mencionado, em NFV, pequenos computadores podem suportar a execução de funções suficientes para manter rede locais operacionais. Assim, a interconexão de tais computadores, utilizando equipamentos simples de transmissão, consegue criar redes metropolitanas em um curto espaço de tempo. Tais características são desejáveis no contexto de redes em campos de batalha (Chen et al., 2019) ou em redes criadas para facilitar a comunicação e troca de dados em ambientes catastróficos (Das et al., 2017). Tais oportunidades já foram investigadas considerando o paradigma SDN (Nobre et al., 2016), porém permanecem como aplicações marginalmente exploradas no contexto do paradigma NFV.

Sendo assim, argumenta-se que o desenvolvimento do paradigma NFV não significa unicamente uma oportunidade de modernização, flexibilização e barateamento das infraestruturas de redes de computadores. Na verdade, este paradigma também pode ser compreendido como uma oportunidade de popularização tecnológica e de suporte a ações humanitárias.

O restante deste trabalho está organizado como segue. O Capítulo 2 traz a fundamentação teórica do paradigma NFV, descrevendo desde conceitos básicos até detalhes relacionados ao fluxo de gerenciamento de funções e serviços de rede. O Capítulo 3 apresenta alguns dos principais trabalhos relacionados, investigando particularidades das plataformas de execução

de VNF disponíveis e as diferentes características e utilizações do elemento operacional EMS. O Capítulo 4 detalha a proposta de arquitetura para plataformas de execução de VNF, assim como a implementação de seu protótipo e experimentos preliminares. De maneira análoga, o Capítulo 5 apresenta uma proposta de arquitetura para EMS, descreve a implementação de seu protótipo e discute resultados preliminares. O Capítulo 6 descreve um conjunto de cenários de utilização de sistemas e soluções pautadas nas arquiteturas para plataformas de execução de VNF e para o elemento EMS propostas. O Capítulo 7 descreve os modelos de personalização de função objetivo e de otimização para a tarefa de integração de serviços virtualizados de rede, apresentando uma solução de mapeamento distribuído baseada em metaheurísticas genéticas e discutindo testes e resultados relacionados a mesma. Finalmente, o Capítulo 8 conclui o trabalho.

2 O PARADIGMA DE VIRTUALIZAÇÃO DE FUNÇÕES DE REDE

Neste capítulo os principais conceitos e fundamentos do paradigma NFV são apresentados e discutidos. O capítulo é dividido em quatro seções. A Seção 2.1 trata das motivações e aspectos arquiteturais gerais relacionados a virtualização de funções de rede. A Seção 2.2 detalha o domínio de trabalho VNF da arquitetura de referência do paradigma NFV. Nela, as particularidades de cada elemento operacional são descritas e discutidas, assim como os protocolos de comunicação disponíveis e as atuais limitações de desenvolvimento. A Seção 2.3 exibe informações relacionadas a serviços virtualizados de rede, dando destaque às suas organizações arquiteturais e aos processos de operacionalização dos mesmos em um ambiente virtualizado de rede. Finalmente, a Seção 2.4 apresenta uma sumarização dos conceitos abordados e uma discussão geral do capítulo.

2.1 NFV EM SÍNTESE

Infraestruturas de rede atuais dependem da execução das mais variadas funções para operar corretamente. Exemplos dessas funções incluem roteadores, servidores de nome, balanceadores de carga, inspetores de tráfego, mitigadores de ataques, entre outros. As funções de rede, ao longo dos anos, têm sido implementadas através de equipamentos dedicados que unem *hardware* e *software*. Esses equipamentos, chamados de *middleboxes* ou *physical appliances*, apresentam diversos benefícios no que diz respeito ao seu desempenho no tratamento de tráfego de rede, geralmente alcançando alta vazão de pacotes e baixo atraso de processamento (Sekar et al., 2012). Entretanto, diversos desafios decorrem da ampla utilização desses equipamentos. Entre esses desafios é possível destacar a dificuldade de gerência da infraestrutura de rede, em vista da necessidade de programas de gerenciamento proprietários, a imposição de migração e escalabilidade física, alto custo para a manutenção e atualização da infraestrutura e limitação do mercado de funções de rede devido à necessidade de desenvolvimento conjunto de *hardware* e *software*.

A virtualização da infraestrutura de rede se tornou uma alternativa para solucionar os problemas relacionados aos equipamentos dedicados que provêm funções e serviços de rede. Essa tendência se concretizou, principalmente, através de dois paradigmas: Redes Definidas por Software (SDN) e Virtualização de Funções de Rede (NFV). Uma representação de alto nível desses paradigmas é apresentada na Figura 2.1. O paradigma SDN (Benzekki et al., 2016) desacopla o plano de controle do plano de dados da infraestrutura rede. Assim, o plano de controle passa a ser implementado por *switches* virtuais e um controlador central (ambos em *software*), normalmente comunicando-se com o plano de dados através do protocolo OpenFlow (Hu et al., 2014). Já o paradigma NFV (Yi et al., 2018) visa desacoplar as funções de rede de seu *hardware* associado, passando a executá-las em um plano de *software* através de tecnologias de virtualização variadas, como virtualização completa, paravirtualização e conteinerização. Diferente do paradigma SDN, em NFV ambos os planos de controle e de dados são virtualizados e, naturalmente, o controle das funções é descentralizado. Entretanto, é importante ressaltar que, apesar de não haver dependência entre esses paradigmas de virtualização, os mesmos são compatíveis entre si, podendo ser utilizados em conjunto (Kaur et al., 2020).

Em particular, o paradigma NFV é fundamentado por cinco requisitos de alto nível definidos pela ETSI (NFVISG, 2013): (i) portabilidade, (ii) desempenho, (iii) integração, (iv) gerência e orquestração e (v) escalabilidade. O requisito de portabilidade diz respeito à capacidade

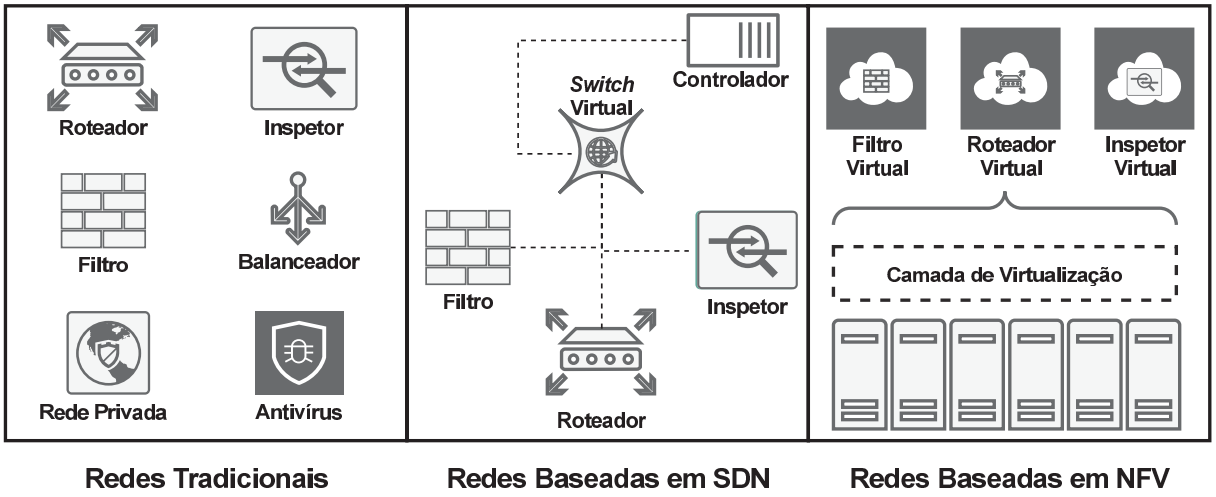


Figura 2.1: Modelos de Infraestrutura de Rede

de vários sistemas de diferentes origens cooperarem entre si para prover um ambiente NFV. Dessa forma, ao serem migrados para diferentes ambientes NFV, esses sistemas devem manter suas características de funcionamento. O requisito desempenho diz respeito ao desafio de mitigar ao máximo possível a degradação de performance intrínseca ao paradigma NFV. Essa degradação é relacionada à passagem da execução das funções de rede de um *hardware* dedicado para um *software* executando em equipamentos genéricos. Em relação à integração, o requisito define que funções e serviços de rede devem ser integrados e executarem em harmonia no ambiente NFV independentemente das tecnologias, plataformas e fornecedores dos mesmos. A integração é tipicamente viabilizada pela adoção de protocolos de comunicação padronizados. A gerência e orquestração dita que, no mínimo, as operações protocolares devem ser implementadas e estar amplamente acessíveis aos agentes a quem elas se destinam. Por fim, a escalabilidade ressalta a necessidade de que o ambiente NFV seja dimensionado sob demanda, evitando ao máximo gargalos de processamento assim como desperdício de recursos computacionais.

Considerando os requisitos fundamentais do paradigma NFV, a ETSI lançou vários documentos de especificação relacionados ao mesmo. Entre esses documentos, é necessário destacar a arquitetura de referência do paradigma NFV (NFVISG, 2014). Essa arquitetura, como ilustrada na Figura 2.2, consiste em três domínios de trabalho (*i.e.*, VI, NFV-MANO e VNF) e múltiplos elementos operacionais. O VI é o domínio responsável pela manutenção e alocação de recursos computacionais. Nele os recursos físicos de servidores de propósito geral são virtualizados e, então, disponibilizados para o ambiente NFV nas porções requeridas por cada instância de função de rede. O NFV-MANO realiza grande parte das operações de gerenciamento do ambiente NFV. Nesse domínio estão presentes três elementos operacionais: VIM, VNFM e NFVO. De maneira geral, o VIM executa requisições para o VI, manipulando recursos computacionais e de virtualização. O VNFM comunica-se com os elementos operacionais do domínio VNF para realizar operações e monitoramentos tanto genéricos (*e.g.*, checagem de estado das instâncias virtuais e ativação/desativação das instâncias virtuais), quanto específicos (*e.g.*, monitoramento de número de pacotes descartados por um filtro, modificação da versão da função de rede instalada). O NFVO dedica-se a orquestrar serviços de rede, executar operações de gerência de alto nível, além de, em certas circunstâncias, funcionar como interface do NFV-MANO para usuários externos. Por fim, o domínio VNF conta com dois elementos operacionais: VNF propriamente dita e EMS. A VNF é o principal elemento do paradigma NFV, sendo ela responsável pelo processamento do tráfego de rede através de uma função. Já o EMS é

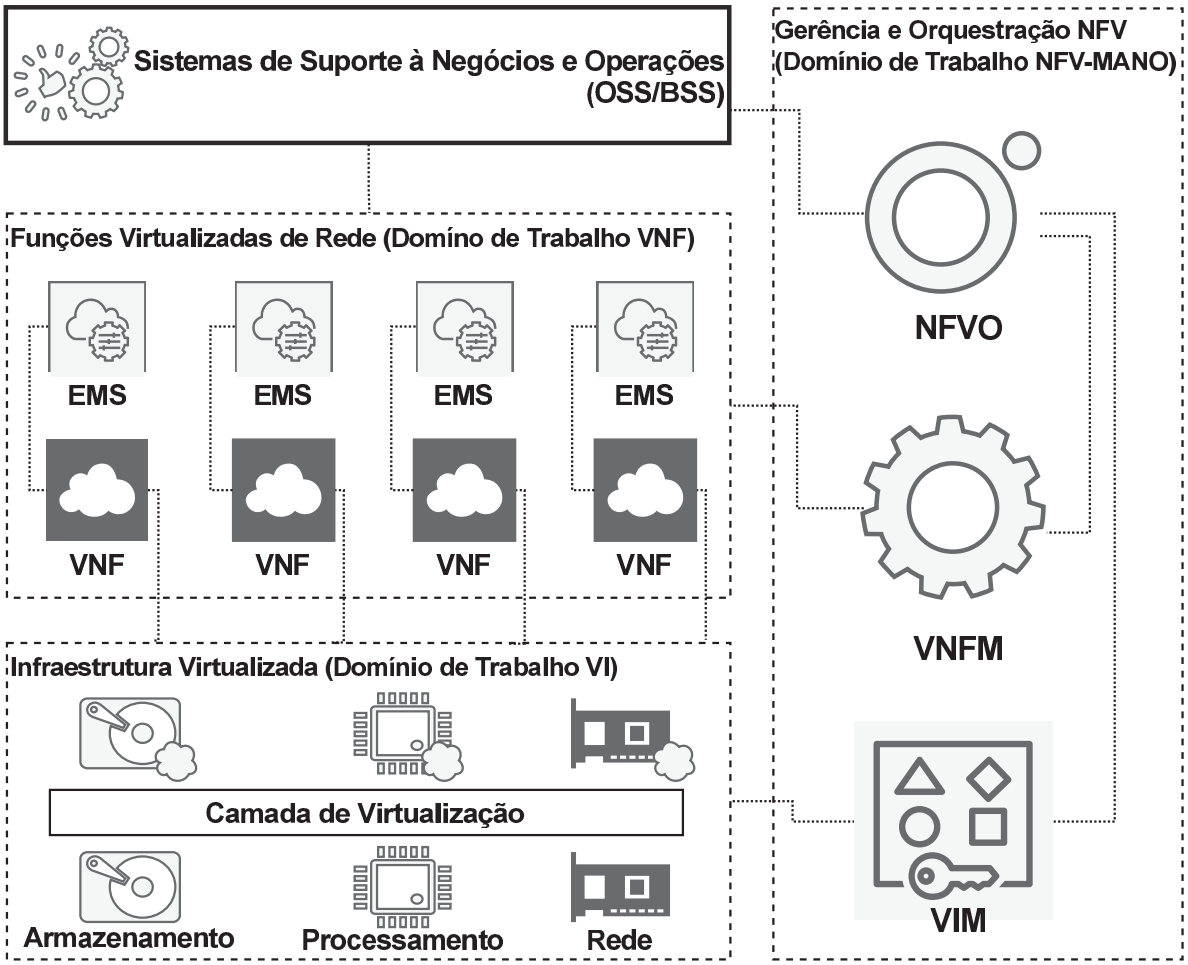


Figura 2.2: Arquitetura NFV Simplificada

o elemento de gerenciamento direto das instâncias de funções de rede, sendo ele responsável por abstrair detalhes de implementação e tecnologias específicas das plataformas de execução de VNF utilizadas.

2.2 DOMÍNIO DE TRABALHO VNF

O domínio de trabalho VNF é responsável pelo processamento do tráfego de rede na arquitetura de referência NFV. Esse domínio é composto de dois elementos operacionais: VNF e EMS. A VNF executa as funções de rede, sendo o elemento central do paradigma de NFV. Um EMS atua no gerenciamento de instâncias de VNF através da execução de rotinas de controle, monitoramento e configuração. Além disso, o EMS é responsável por interfacear a comunicação entre VNF e demais elementos operacionais da arquitetura NFV. Ambos VNF e EMS não apresentam arquiteturas internas padronizadas, sendo projetados e desenvolvidos livremente. Apesar de essa dinâmica resultar em uma grande variedade de implementações, muitas com características específicas segundo o domínio ao qual serão aplicadas, ela dificulta a criação de ambientes heterogêneos (*i.e.*, com diferentes implementações de VNF e EMS) que sejam, em simultâneo, interoperáveis. Esse cenário, de maneira geral, rompe com dois fundamentos do paradigma NFV: a capacidade dos elementos operacionais de integrar-se através de protocolos predefinidos (integração) e de serem utilizados naturalmente em diferentes ambientes de virtualização de redes (portabilidade).

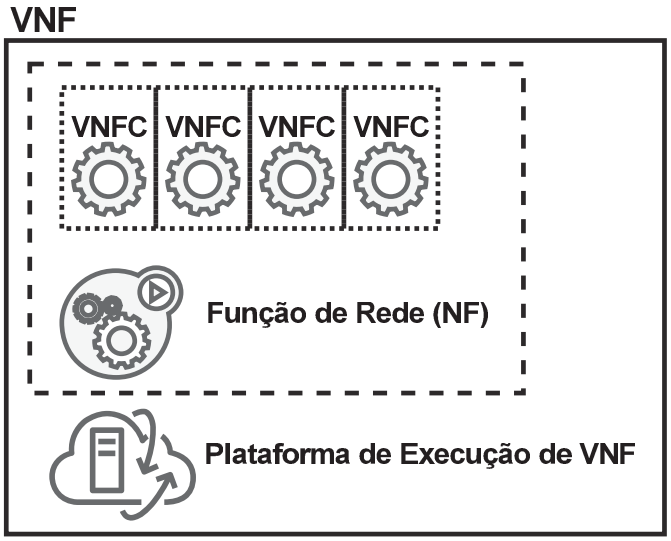


Figura 2.3: Representação em Alto Nível de VNF

Uma VNF, representada em alto nível na Figura 2.3, pode ser decomposta em duas partes principais (Fulber-Garcia et al., 2019b): plataforma de execução de VNF e Função de Rede (*Network Function* – NF). A plataforma de execução de VNF representa o ambiente virtualizado e seus recursos computacionais disponíveis para a hospedagem e execução de uma função de rede. Também é a plataforma que habilita a comunicação entre o elemento operacional VNF e os demais elementos da arquitetura NFV. Diversas plataformas com diferentes características de virtualização estão à disposição atualmente, como aquelas baseadas em sistemas minimalistas do tipo *unikernel* (ClickOS (Martins et al., 2014) e Click-On-OSv (Marcuzzo et al., 2017)), aquelas baseadas em sistemas virtualizados multiprocessos (COVEN (Fulber-Garcia et al., 2019a) e Leaf (Flauzino et al., 2020)) e aquelas baseadas em contêineres (OpenNetVM (Zhang et al., 2016b)). A NF, por outro lado, consiste do código que será executado nessas plataformas com a finalidade de processar tráfego de rede. Esses códigos podem ser compreendidos como, por exemplo, rotinas de roteamento, tradução de nomes, filtragem, análise e balanceamento de carga. Uma NF pode ser implementada com diversas linguagens de programação (*e.g.*, Python, Java, C e C++) e *frameworks* (*e.g.*, Click Modular Router (Kohler et al., 2000), Click Middleboxes (Laufer et al., 2016) e libVNF (Naik et al., 2018)). Porém, é necessário observar a existência de suporte para essas ferramentas de implementação nas plataformas de execução de VNF a serem utilizadas.

Plataformas de execução de VNF também podem habilitar funcionalidades sofisticadas e pouco exploradas do paradigma NFV. Entre essas funcionalidades, destacam-se o processamento nativo de NSH (Quinn et al., 2018) e o suporte a execução de Componentes de VNF (*VNF Components* - VNFC) (NFVISG, 2018a). Particularmente, o processamento NSH em plataformas de execução de VNF é realizado por módulos internos que reconhecem e desencapsulam este cabeçalho dos pacotes recebidos como entrada. Da mesma forma, o cabeçalho é atualizado e reinserido nos pacotes antes do encaminhamento dos mesmos para fora dos domínios da plataforma. Nota-se que esse processo somente é necessário quando a VNF faz parte de uma SFC. Nesse contexto, plataformas que não conseguem processar NSH demandam um *proxy* acoplado para serem utilizadas em um serviço de rede virtualizado.

Por sua vez, um VNFC consiste em um componente interno que implementa operações específicas de uma NF. Normalmente, um VNFC é responsável por uma operação de baixa complexidade, sendo a combinação de múltiplos componentes o que permite a criação de uma função de rede completa e complexa. Um VNFC é instanciado como um elemento interno e

independente (tipicamente através de processos ou *threads*), porém seu ciclo de vida é subordinado ao ciclo de vida da VNF da qual faz parte. O conceito de VNFC, de forma geral, visa aprimorar o reúso de código e agilizar o desenvolvimento de funções de rede em NFV (*e.g.*, um verificador de assinaturas de *payload* pode ser usado tanto como parte de um inspetor de pacotes quanto como parte de um balanceador de carga em camada sete).

O EMS (também chamado de *Element Manager* - EM) foi proposto pela primeira vez em 1988 no contexto das Redes de Gerenciamento de Telecomunicações (*Telecommunications Management Networks* - TMN) (Sector, 2000), estas apresentadas pela União Internacional de Telecomunicações (*International Telecommunication Union* - ITU). Nesse cenário, processos TMN podem assumir papéis de gerenciados ou gerenciadores. Enquanto papéis gerenciados se referem ao provimento de informações relacionadas a determinados recursos, os papéis gerenciadores recebem e disparam notificações, além de atenderem requisições por informações dos recursos gerenciados. Equipamentos, sejam eles físicos ou virtuais, que executam tanto processos gerenciados quanto gerenciadores em uma TMN são considerados instâncias de EMS.

Já em 2018, o Projeto de Parceria de 3ª Geração (*3rd Generation Partnership Project* - 3GPP) lançou um documento especificando uma nova arquitetura de gerenciamento baseado em serviços (Services e Aspects, 2018). Com isso, o EMS se tornou o elemento responsável por gerenciar e orquestrar funções e serviços de rede. Particularmente, o EMS passou a ser chamado de Serviço de Gerenciamento (*Management Service* - MnS), mas assumindo os mesmos papéis previamente estabelecidos nas especificações das TMN – processos gerenciados ou gerenciadores. A nomenclatura de tais processos, entretanto, também foi alterada, sendo os processos gerenciados chamados de Produtores MnS, e os processos gerenciadores de Consumidores MnS. Na arquitetura 3GPP, um MnS é provido e acessado por entidades lógicas chamadas de Funções de Gerenciamento (*Management Functions* – MnF), que podem ser tanto funções de rede quanto elementos independentes.

Finalmente, a 3GPP enquadrou a arquitetura de gerenciamento baseado em serviços na arquitetura NFV proposta pela ETSI. Assim, uma VNF deve carregar um Produtor MnS executando como um Agente de Gerência (Fulber-Garcia et al., 2019a), e o EMS é tratado como um Consumidor MnS para tal VNF, enquanto também trabalha como um Produtor MnS para os demais elementos operacionais da arquitetura NFV, como o VNFM, o NFVO, e o OSS/BSS. Entretanto, o 3GPP para nesse ponto. Ou seja, não existem indicações de como esses serviços de gerenciamento devem ser implementados, nem como tratar os modelos de dados e operações de gerenciamento especificadas pela ETSI para o paradigma NFV (NFVISG, 2020a).

Assumindo puramente as definições providas pela ETSI, entretanto, o EMS é considerado o principal elemento de interoperabilidade entre o ambiente NFV e o domínio de trabalho VNF. Esse elemento emprega interfaces reativas e proativas para comunicar-se com instâncias de VNF, com o VNFM e com sistemas OSS/BSS. Enquanto a comunicação do EMS com instâncias de VNF ocorre exclusivamente de maneira proativa (*i.e.*, o EMS toma a iniciativa do envio de mensagens), a comunicação com o VNFM e sistemas OSS/BSS pode acontecer de maneira tanto reativa (*i.e.*, outros elementos operacionais iniciam o envio de mensagens para o EMS), quanto proativa. Em relação às instâncias de VNF, é de competência do EMS o conhecimento das interfaces particulares de cada plataforma de execução de VNF em uso. A partir das operações específicas providas por essas interfaces, operações generalizadas podem ser programadas e utilizadas independentemente da plataforma de execução de VNF que deve ser comunicada. Um exemplo dessas operações generalizadas é a operação "*VNF Indicators*" (NFVISG, 2020c). Essa operação em particular deve recuperar todos os indicadores (*i.e.*, métricas de monitoramento) disponíveis para uma dada instância de VNF. Observa-se que essa operação, assim como outras, é relacionada à comunicação entre EMS e VNFM e encontra-se definida na interface "Ve-Vnfm-em"

da arquitetura de referência NFV (NFVISG, 2020a). Tal interface tem sua implementação padronizada através do Protocolo de Transferência de Hipertexto (*Hypertext Transfer Protocol* - HTTP) (NFVISG, 2020c). Finalmente, não existem especificações, nem mesmo recomendações, em relação à interface entre o EMS e sistemas OSS/BSS. Sendo assim, a mesma interface direcionada ao VNFM é comumente utilizada pelos mesmos.

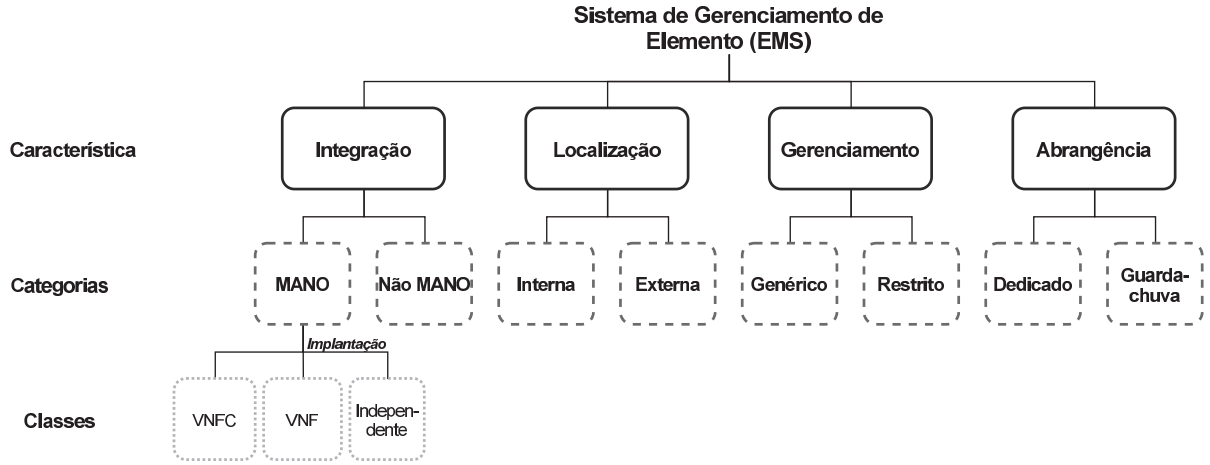


Figura 2.4: Categorias e Classes de EMS

Um EMS pode ser categorizado de diferentes formas considerando seu modo de operação e abrangência de atuação no ambiente NFV e, em especial, no domínio VNF. A Figura 2.4 sumariza as categorias e classes de EMS em vista de diferentes características. A primeira característica de EMS para a qual são apresentadas considerações consiste da **integração** deste à arquitetura de referência NFV. Nesse caso, duas categorias são possíveis: `MANO` e `não-MANO` (NFVISG, 2016). Um EMS MANO é consciente do ambiente NFV do qual faz parte, nativamente cooperando com o domínio NFV-MANO através do VNFM. Por outro lado, um EMS não-MANO pode ser compreendido como qualquer sistema de gerenciamento que atue sobre instâncias de VNF, sendo que o mesmo não permite, não reconhece ou não diferencia interações realizadas com o VNFM. Ainda, um EMS MANO pode ser classificado considerando sua **implantação** no ambiente NFV: `VNFC`, `VNF` e `Independente` (NFVISG, 2016) (NFVISG, 2018b). A classe de EMS MANO VNFC é executada internamente a uma instância de VNF, sendo reconhecida como um componente interno (*i.e.*, parte da NF) da mesma e estando subordinada ao ciclo de vida de sua respectiva função de rede. Um EMS da classe MANO VNF é implantado externamente às instâncias de VNF gerenciadas por ele, porém é interpretado e controlado pelos demais elementos operacionais da arquitetura NFV como uma instância de VNF. Por último, um EMS MANO Independente é integrado ao ambiente NFV, mas não está sob o controle de nenhum elemento operacional do mesmo, tendo um nível de independência similar ao de sistemas de OSS/BSS.

Quanto à **localização** de um EMS em relação às instâncias de VNF gerenciadas por ele, duas categorias são definidas: `Interna` e `Externa` (NFVISG, 2018b). Um EMS da categoria Interna é contido em uma instância de VNF. Dessa forma, o EMS pode substituir qualquer agente de gerência da plataforma de execução de VNF, executando diretamente no sistema operacional da mesma. Opcionalmente, o EMS também pode ser implantado como um VNFC (*i.e.*, EMS MANO VNFC). Por outro lado, a categoria Externa de EMS determina sua independência operacional das instâncias de VNF gerenciadas, realizando suas tarefas externamente a elas. Desse modo, um EMS pode ser implantado tanto como um sistema independente, quanto como uma instância de VNF subordinada ao NFV-MANO.

Atentando ao **gerenciamento** das instâncias de VNF possibilitado pelo EMS, outras duas categorias são aplicáveis: `Genérico` e `Restrito` (NFVISG, 2018b). Um EMS pertencente à categoria Genérico fornece apenas o conjunto básico de operações de gerenciamento (definido em (NFVISG, 2020a)) e uma interface de comunicação que permite o acesso às mesmas. Já um EMS Restrito, além de responder pelos mesmos requisitos de um EMS genérico, ainda provê operações restritas a uma ou mais plataformas de execução de VNF. Dado esse cenário, não há garantias de funcionamento de um EMS restrito (em particular de suas operações não padronizadas) para plataformas de execução de VNF diferentes daquelas presentes no conjunto de projeto do mesmo. Por fim, duas categorias são consideradas quanto a **abrangência** de gerenciamento de um EMS (*i.e.*, número de instâncias de VNF gerenciadas): `Dedicado` e `Guarda-chuva` (NFVISG, 2014, 2015, 2018b). Um EMS categorizado como dedicado é responsável pelo gerenciamento de uma única instância de VNF. Um EMS Guarda-chuva, entretanto, gerencia múltiplas instâncias de VNF em simultâneo.

2.3 SERVIÇOS VIRTUALIZADOS DE REDE

No paradigma NFV, serviços de rede virtualizados podem ser construídos através da conexão de diversas funções em uma estrutura conhecida como Topologia de Serviço (*Service Topology* - ST) (Fulber-Garcia et al., 2020a). Atualmente, ambos IETF e ETSI realizam esforços para definir e padronizar serviços de rede em NFV. A IETF apresenta o conceito de Cadeia de Função de Serviço (SFC) (Halpern e Pignataro, 2015). Uma SFC consiste em uma ST e de um conjunto de metadados complementares. Esses metadados definem, por exemplo, dependências entre funções de rede e políticas de operação de serviço. Já a ETSI apresenta o conceito de Serviço de Rede (*Network Service* – NS) (NFVISG, 2014). Um NS abrange todos os dados presentes em uma SFC e permite, ainda, a inclusão de *scripts* de ciclo de vida e modelos de configuração de serviço. Esses recursos extras são tipicamente utilizados para auxiliar na execução do processo de implantação de um serviço virtualizado. É importante ressaltar que, apesar das similaridades funcionais, SFC e NS apresentam nomenclaturas díspares para conceitos virtualmente iguais. Entre esses conceitos, destacamos a equivalência IETF (SFC)/ETSI (NS) entre "Nós de Borda"/"Pontos Finais" (*i.e.*, extremos de uma topologia de serviço), "Funções de Serviço"/"Funções de Redes Virtualizadas e Físicas" (*i.e.*, elementos de processamento de tráfego de rede) e "Caminho de Função de Serviço"/"Caminho" (*i.e.*, conexões entre funções de rede). Além disso, conceitos que são individualizados pela ETSI, como "Caminhos Fim-a-fim" (*i.e.*, segmento linear de uma ST) e "Grafo de Encaminhamento" (*i.e.*, ST completa), são simplesmente tratados como a própria "Cadeia de Função de Serviço" pela IETF.

Além das definições de SFC, a IETF também disponibiliza um modelo arquitetural para a implementação de serviços de rede virtualizados (Halpern e Pignataro, 2015). A arquitetura, ilustrada na Figura 2.5, conta com três elementos operacionais: Classificador, Encaminhador de Funções de Serviço (*Service Function Forwarder* – SFF) e Funções de Serviço (*Service Function* - SF). O Classificador é o elemento que recebe o tráfego de rede de entrada e, a partir de políticas, define para qual serviço ele deve ser encaminhado. A classificação de tráfego pode ser realizada através de cabeçalhos de protocolos de diferentes camadas ou, ainda, considerando dados de um pacote. Após definido o serviço de rede destino, o tráfego de entrada é repassado ao SFF. Esse elemento encaminha os pacotes recebidos para funções específicas presentes na ST do serviço indicado. No caso de topologias não lineares (*i.e.*, com pontos de ramificação), o SFF é também responsável por determinar a ramificação correta para encaminhamento do tráfego de rede. Normalmente, essa decisão é tomada a partir de marcações realizadas pelas próprias funções de rede nos pacotes. Por fim, as funções de serviço equivalem às instâncias de VNF e

são responsáveis pelo processamento do tráfego recebido. A arquitetura SFC também aborda a coexistência de Funções de Rede Físicas (*Physical Network Funtions* – PNF) com instâncias de VNF, criando um ambiente híbrido. Esse ambiente pode ser considerado uma alternativa para a realização da transição de uma infraestrutura de rede física para uma virtualizada.

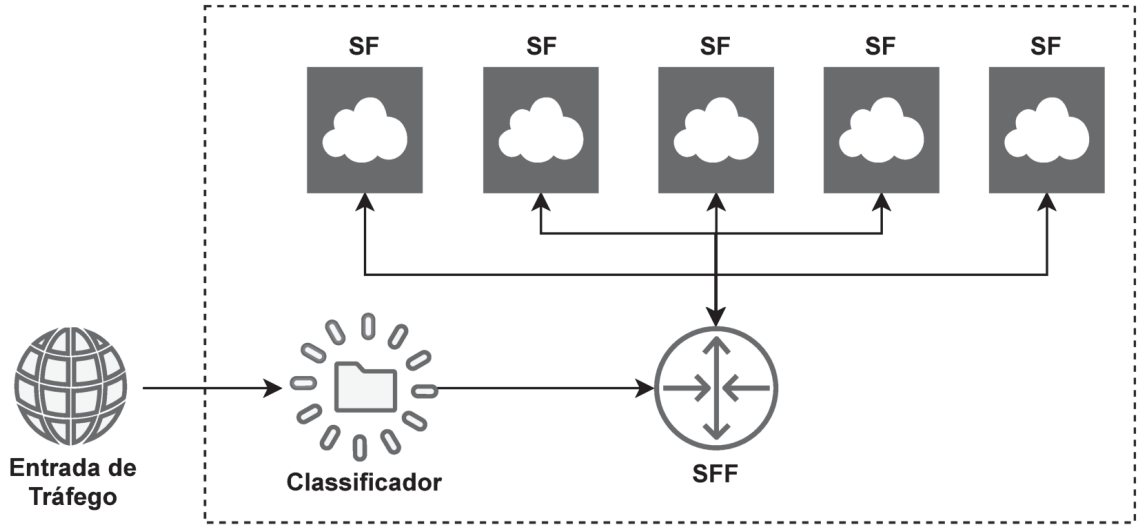


Figura 2.5: Arquitetura SFC Simplificada

Para que a arquitetura SFC seja capaz de comunicar-se entre seus elementos operacionais, a IETF também propôs o protocolo de Cabeçalho de Serviço de Rede (NSH) (Quinn et al., 2018). O NSH encapsula pacotes (camada três) para serem processados por uma SFC. Esse cabeçalho carrega informações, por exemplo, sobre o caminho percorrido pelo pacote, a sua localização atual na ST e metadados providos pelas instâncias de VNF inseridos durante seu processamento. O NSH é composto de três subcabeçalhos: Cabeçalho Base (*Base Header* – BH), Cabeçalho de Caminho de Serviço (*Service Path Header* – SPH) e Cabeçalho de Contexto (*Context Header* – CH). O BH armazena informações gerais sobre o protocolo e sobre os próximos subcabeçalhos. Para isso o BH utiliza quatro bytes divididos em cinco campos: Versão, OBit, *Time To Live* (TTL), Tamanho e Tipo de Metadados. O SPH também é formado por quatro bytes, estes representando dois campos: *Service Path Identifier* (SPI - que define a SFC selecionada) e *Service Index* (SI - que define a função da SFC em que o pacote se encontra). Finalmente, o CH pode ter tamanho fixo (com dezesseis bytes) ou variável (conforme a necessidade). Esse subcabeçalho carrega metadados genéricos decorrentes do processamento de um pacote pelas funções de serviço. Alternativamente, o paradigma NFV pode ser associado ao paradigma SDN, removendo a necessidade de utilização de NSH e passando a empregar o protocolo OpenFlow (ou similar) para realizar o encaminhamento do tráfego de rede (Matias et al., 2015; Kaur et al., 2020).

Além do modelo arquitetural e dos protocolos de comunicação, a criação e operacionalização de um serviço de rede virtualizado no ambiente NFV passa pela execução de um processo de implantação. De maneira geral, esse processo pode ser resumido em três tarefas principais (Herrera e Botero, 2016): composição, integração e programação de execução. Essas tarefas preparam o serviço de rede considerando um conjunto de requisitos de desempenho preestabelecidos. Requisitos são tipicamente definidos através de métricas (*e.g.*, atraso, vazão e sobrecarga) e objetivos (*e.g.*, estabilização, maximização e minimização), como minimizar a sobrecarga do serviço, maximizar a vazão e minimizar o consumo de memória. Em particular, a tarefa de composição é responsável pela definição da ST de um serviço de rede; a tarefa de integração aloca o serviço de rede no substrato físico disponível; e a programação da execução

determina quantos e quais processadores vão atender as funções de rede, além do tempo de processamento dedicado a cada uma delas.

Em especial, a tarefa de integração pode ser realizada por diferentes técnicas, selecionadas de acordo com o ambiente virtualizado disponível. As três técnicas mais comumente aplicadas são: mapeamento multidomínio; seleção de funções de rede; e alocação em servidores. O mapeamento multidomínio destina-se a determinar domínios administrativos e provedores capazes de suportar um serviço, ou parte dele, em sua infraestrutura. A seleção de funções de rede busca por funções já implantadas em um ambiente virtualizado, para utilizá-las de forma compartilhada entre diversos serviços. Finalmente, a alocação de servidores verifica, dentro de um domínio já definido, quais máquinas físicas devem hospedar e executar determinadas funções de rede. Essas técnicas são continuamente adaptadas para suprir necessidade de provedores, operadores de rede e clientes. Atualmente, além de cumprir o papel de operacionalização do serviço, essas técnicas de integração também são exploradas como um modelo de negócio em arquiteturas de *Network-as-a-Service* (Boubendir et al., 2016). Especificamente, o presente trabalho investiga novas possibilidades de integração através da técnica de mapeamento multidomínio.

Por fim, destaca-se que, devido à ampla necessidade de informações do ambiente NFV, de políticas de sistema e de serviço, dos acordos com provedores de recursos e de operações de instanciação, os algoritmos de implantação são tipicamente executados em sistemas OSS/BSS. Porém, esses mesmos algoritmos normalmente apresentam um comportamento operacional monolítico, provendo um modelo de otimização único para efetivar uma tarefa ou técnica de implantação de serviços. Dessa forma, os OSS/BSS que executam as tarefas de implantação precisam ser instrumentados com um conjunto de informações e configurações específicas, estas requeridas por cada um dos algoritmos disponibilizados para utilização.

2.4 SUMARIZAÇÃO E DISCUSSÃO

Tecnologias de virtualização são importantes catalisadores de inovações para as infraestruturas de rede. Essas inovações se justificam principalmente devido à contínua popularização da Internet, seja para uso doméstico ou corporativo. Além disso, novas aplicações e tecnologias também apresentam fortes demandas das redes de computadores para sua operação e cooperação. Exemplos dessas tecnologias são Internet das Coisas (*Internet of Things* - IoT) (Gupta et al., 2017), Redes *Ad-Hoc* Veiculares (*Veicular Ad Hoc Network* - VANET) (Hanan et al., 2017) e *Big Data* (Al-Salim et al., 2017). Todo esse cenário gera sobrecarga na rede devido à transmissão recorrente de grandes volumes de dados. Assim, suportar e gerenciar esses volumes crescentes de dados se tornou uma tarefa cada vez mais complexa. Isso, inevitavelmente, desencadeou uma busca por infraestruturas de redes flexíveis, elásticas e móveis, de modo a prover qualidade de serviço e de experiência sob demanda, além de evitar o desperdício de recursos computacionais, financeiros e energéticos. Para alcançar esses objetivos, oportunidades de virtualização de redes passaram a ser amplamente pesquisadas, originando, por exemplo, os paradigmas SDN e NFV.

No decorrer do desenvolvimento desses paradigmas tornou-se necessário definir protocolos e arquiteturas para padronizar e tornar interoperáveis as soluções e plataformas relacionadas aos mesmos. Em particular para NFV, a ETSI e a IETF lançaram documentos de especificação e de recomendação estruturando, por exemplo, a arquitetura de referência do paradigma e de implementação de serviços de rede virtualizados. Esses documentos têm orientado os esforços de pesquisa desde suas publicações. Entretanto, apesar de detalhados, eles ainda não contemplam todas as necessidades do paradigma NFV, não especificando protocolos de comunicação e modelos de desenvolvimento de um conjunto de elementos operacionais do mesmo.

Entre os elementos não padronizados destacam-se aqueles pertencentes ao domínio de trabalho VNF, ou seja, as próprias instâncias de VNF e as soluções de EMS não contam com a totalidade de seus protocolos de comunicação definidos e arquiteturas internas padronizadas. O elemento VNF é central para a realização do processamento de tráfego de rede, e o elemento EMS indispensável para permitir o gerenciamento das instâncias de VNF e a interoperabilidade entre diferentes plataformas de execução de VNF e o ambiente NFV em sua totalidade. Dessa forma, documentos que discutam suas necessidades operacionais, competências e possíveis modelos de implementação são necessários para garantir que diferentes desenvolvedores satisfaçam importantes fundamentos do paradigma NFV, como integração e portabilidade.

Ainda, através das especificações e sugestões de entidades como ETSI e IETF, o conceito de serviços virtualizados foi forjado em NFV. De maneira geral, um serviço virtualizado consiste em uma cadeia de funções que gera uma topologia capaz de prover funcionalidades complexas. Essas funcionalidades são aplicadas ao tráfego de rede pelo seu encaminhamento ordenado através da topologia de serviço. A operacionalização de um serviço de rede depende da execução de uma série de tarefas que compõem o processo de implantação: composição, integração (com técnicas como mapeamento multidomínio, seleção de funções, e alocação em servidores) e programação da execução.

Em específico, no contexto de implantação de serviços virtualizados de rede, a maioria das soluções disponíveis trabalham através de algoritmos bem definidos e pouco flexíveis, que executam, normalmente, em sistemas OSS/BSS. Apesar de tais soluções obterem sucesso na otimização da implantação dos serviços de rede dentro dos objetivos que se propõem, elas não permitem modificações em seus modelos operacionais para se adequarem a necessidades específicas de operadores e gerentes de rede. Devido a essa falta de flexibilidade, o processo de implantação de serviços pode não atender completamente os propósitos das partes interessadas, obtendo resultados de otimização apenas parcialmente satisfatórios.

3 TRABALHOS RELACIONADOS

Este capítulo apresenta e discute os trabalhos diretamente relacionados a esta Tese. Inicialmente, as principais plataformas de execução de VNF existentes, sejam elas categorizadas como genéricas, orientadas a SFC ou orientadas a domínio específico, são detalhadas e seus projetos de desenvolvimento evidenciados na Seção 3.1. Em seguida, na Seção 3.2, os esforços de desenvolvimento de soluções de EMS são discutidos tendo em vista as diversas categorias, relatadas no Capítulo 2, desse elemento operacional. Então, as principais soluções relacionadas à técnica de mapeamento multidomínio para integração de serviços rede virtualizados são apresentadas e discutidas na Seção 3.3. Finalmente, a Seção 3.4 sumariza o capítulo e discute oportunidades de aprimoramento do domínio de VNF e da técnica de mapeamento multidomínio de serviços virtualizados, em vista dos fundamentos do paradigma NFV.

3.1 PLATAFORMAS DE EXECUÇÃO DE VNF

Diversas de plataformas de execução de VNF foram propostas no decorrer dos anos recentes. Essas plataformas, entretanto, foram projetadas sem nenhuma arquitetura padronizada, sendo, por muitas vezes, desconsideradas quaisquer discussões relacionadas à interoperabilidade e gerenciamento das mesmas. Também, essas plataformas não satisfazem inteiramente os fundamentos do paradigma NFV, além de não suportarem nativamente funcionalidades inovadoras apresentadas no mesmo contexto (*e.g.*, VNFC e NSH). Porém, apesar dos desafios descritos, as plataformas de execução de VNF existentes são quem suportam o paradigma NFV, provendo a execução de funções de rede variadas com relativo sucesso e alto desempenho. A seguir são apresentadas e discutidas as principais plataformas de execução de VNF através da exposição e detalhamento de suas características arquiteturais e operacionais. As figuras apresentadas no decorrer desta seção exibem elementos e módulos através de retângulos (linhas contínuas para a plataforma de execução de VNF e linhas tracejadas para outros elementos associados) e comunicações através de setas (linhas contínuas para dados e pontilhadas para gerenciamento e monitoramento de ciclo de vida).

O **ClickOS** (Martins et al., 2014), cuja arquitetura é ilustrada na Figura 3.1, é uma plataforma de execução de VNF baseada no sistema operacional MiniOS. O MiniOS consiste em um *kernel* minimalista que executa um único processo por vez (*i.e.*, *unikernel*). Esse sistema é disponibilizado como um recurso independente do projeto Xen (MiniOS, 2021). Assim, dentro desse contexto, o ClickOS é projetado para ser paravirtualizado exclusivamente pelo hipervisor Xen. Essa plataforma utiliza o acelerador de pacotes *netmap* (Rizzo, 2012) como uma forma de aprimorar o seu desempenho no processamento de tráfego de rede. Entretanto, é importante ressaltar que a adoção desse acelerador de pacotes demanda uma série de alterações no hipervisor Xen tradicional. Essas alterações incluem, por exemplo, a integração do *switch VirtuAl Local Ethernet* (VALE) para realizar o direcionamento dos fluxos de rede. Do ponto de vista das funções de rede, o ClickOS suporta a execução de algoritmos desenvolvidos através de *Click Modular Router* (CMR), provendo compatibilidade integral com todas as suas bibliotecas nativas. Por fim, a plataforma também disponibiliza uma interface de monitoramento via *socket* (recurso do CMR), além de viabilizar o gerenciamento local das instâncias de VNF através de uma aplicação terceirizada, chamada XenStore. Apesar da existência desses recursos de gerência, as operações fornecidas são muito limitadas.

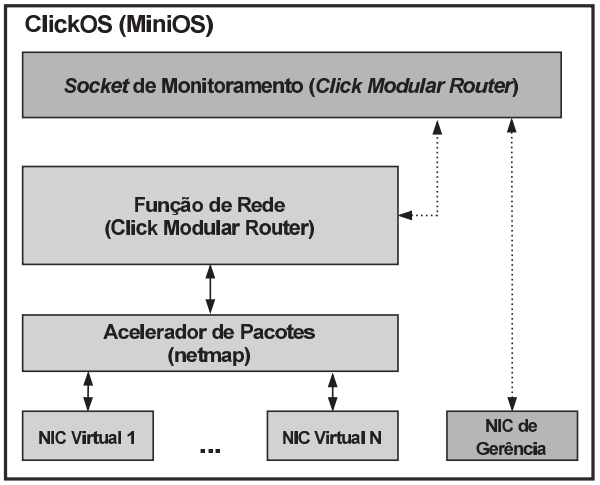


Figura 3.1: Arquitetura Geral do ClickOS

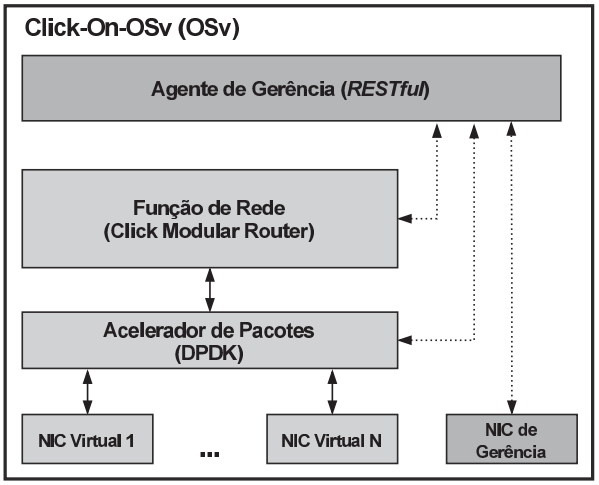


Figura 3.2: Arquitetura Geral do Click-On-OSv

A plataforma de execução de VNF **Click-On-OSv** (Marcuzzo et al., 2017), apresentada na Figura 3.2, é baseada em um sistema operacional minimalista e *unikernel* denominado OSv (Kivity et al., 2014). Esse sistema executa um único processo durante seu ciclo de vida. Porém, diferentemente do MiniOS, o OSv pode utilizar múltiplas *threads* subordinadas ao seu processo em particular. O Click-On-OSv é instanciado através de paravirtualização, sendo compatível com diferentes hipervisores, como KVM, Xen e VirtualBox. Também, para aprimoramento de desempenho, em particular da vazão, a plataforma emprega o acelerador de pacotes *Data Plane Development Kit* (DPDK) (DPDK, 2021). O suporte à execução de funções de rede é limitado a implementações em CMR. Entretanto, devido à presença do DPDK, a camada de enlace não pode ser diretamente acessada pela biblioteca padrão de *sockets* do CMR, sendo esta substituída pela biblioteca nativa de elementos DPDK do mesmo. Finalmente, essa plataforma disponibiliza um agente de gerência completo através de uma interface *Representational State Transfer* (REST). Esse agente oferece desde funções de monitoramento (*e.g.*, consumo de recursos computacionais e recuperação de relatórios de execução e erros), até operações de ciclo de vida interno da plataforma e da função de rede por ela hospedada (*e.g.*, inicialização e terminação da função de rede e/ou da máquina virtual e atualização da função de rede).

O **Leaf** (Flauzino et al., 2020) é uma plataforma de execução de VNF enquadrada no CloudStack Vines, que provê um sistema NFV-MANO completo e integrado à nuvem CloudStack. A plataforma é executada através da virtualização completa, gerando uma instância virtual executando o sistema operacional Ubuntu Cloud (Canonical, 2021b) e sendo compatível com grande parte dos hipervisores existentes. O Leaf não emprega nenhum acelerador de pacotes em sua versão atual, porém ferramentas como *netmap* e DPDK podem ser naturalmente integradas de forma terceirizada, dado que ambas possuem compilações para o seu sistema operacional subjacente. Em relação às funções de rede, o Leaf suporta a execução de códigos Python e C, mas não existe uma limitação definida de linguagens e bibliotecas que podem ser instaladas na plataforma após a sua instanciação. O diferencial da plataforma Leaf em relação às demais consiste do seu agente de gerência. Esse agente, disponibilizado através de uma interface REST, é responsável pela execução de todo o ciclo de vida da plataforma e da sua função de rede. Porém, não existe um conjunto predeterminado de operações de gerenciamento, mas sim um agente de execução de *scripts* que são enviados para a plataforma juntamente ao código da função de rede e outros documentos de especificação de VNF – esse conjunto de recursos é chamado de "Pacote de VNF" (*VNF Package* – VNFP (NFVISG, 2020b)). Então, após validados, os *scripts*

de gerência são apresentados aos usuários da plataforma como operações acessíveis através dela. A arquitetura abstrata do Leaf é mostrada na Figura 3.3.

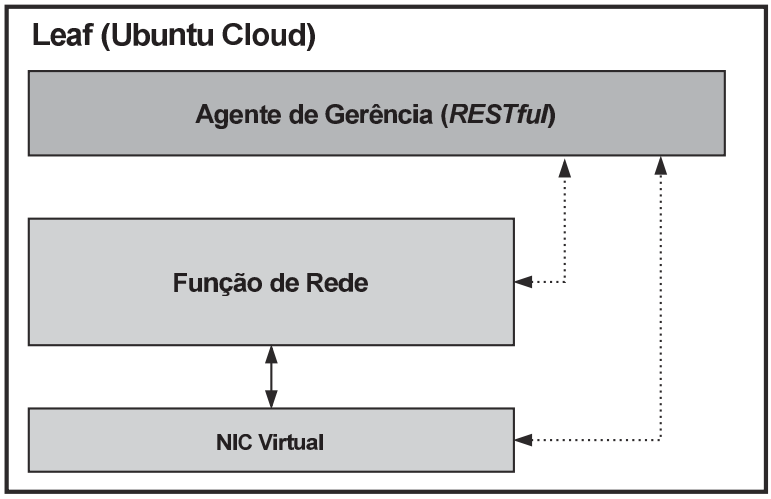


Figura 3.3: Arquitetura Geral do Leaf

O **SampleVNF** (OPNFV, 2021) consiste em uma plataforma de execução de VNF criada e mantida pelo projeto *Open Platform for* NFV (OPNFV). O OPNFV (Guerassimov et al., 2016) desenvolve diversas soluções relacionadas à virtualização de rede, como implementações de NFV-MANO, sistemas OSS/BSS e *benchmarks* de funções. Dentro desse cenário, o SampleVNF é compreendido como uma plataforma para execução de testes de VNF, sendo sua arquitetura apresentada na Figura 3.4. Porém, independente de suas finalidades analíticas, essa plataforma pode também ser utilizada para processar tráfego real em um ambiente virtualizado de rede. Nesse caso, o SampleVNF é instanciado como uma máquina virtual completa que executa o sistema operacional Ubuntu, além de empregar o acelerador de pacotes DPDK no gerenciamento e aprimoramento da vazão de suas interfaces virtuais de rede. Ainda, o SampleVNF não define nenhuma limitação de suporte à execução de funções de rede implementadas com determinadas linguagens, bibliotecas e ferramentas de programação, além de não haver nenhuma discussão sobre gerência do ciclo de vida interno e monitoramento. Por fim, cabe ressaltar que o SampleVNF provê, associada à plataforma de execução de VNF, uma coleção de funções de rede que incluem, por exemplo, *firewall*, *VLAN Access Control List* (VACL), *User Datagram Protocol* (UDP) *replay* e *Carrier Grade Network Address Translation* (CGNAT). Todas essas funções de rede são escritas na linguagem C e foram previamente validadas e testadas no contexto do SampleVNF.

O *Network Function Framework for Go* (**NFF-Go**) (Intel, 2021) é uma coleção de bibliotecas que permitem a implementação de funções de rede em linguagem de programação Go. Essas funções podem ser instanciadas como contêineres ou microsserviços (baseados em processos) em ambiente Linux. A arquitetura da plataforma de execução de VNF utilizada pelo NFF-Go é ilustrada na Figura 3.5. Nesse caso, o acelerador de pacotes DPDK (DPDK, 2021), além de ser responsável por aumentar a vazão da plataforma, eleva ao nível de usuário o controle das interfaces de rede acessadas pelas funções. Assim, esse acesso pode ser realizado diretamente através de elementos de rede de alto nível disponibilizados naturalmente pela plataforma. É importante ressaltar que, no contexto da plataforma de execução de VNF, não são apresentados agentes de gerência para controle e monitoramento do ciclo de vida das instâncias de VNF. Entretanto, a maior contribuição do NFF-Go não reside necessariamente na sua plataforma de execução de VNF, mas sim na adaptação da linguagem Go para suportar o desenvolvimento nativo e generalizado de funções de rede complexas. Ao viabilizar diversas ferramentas de

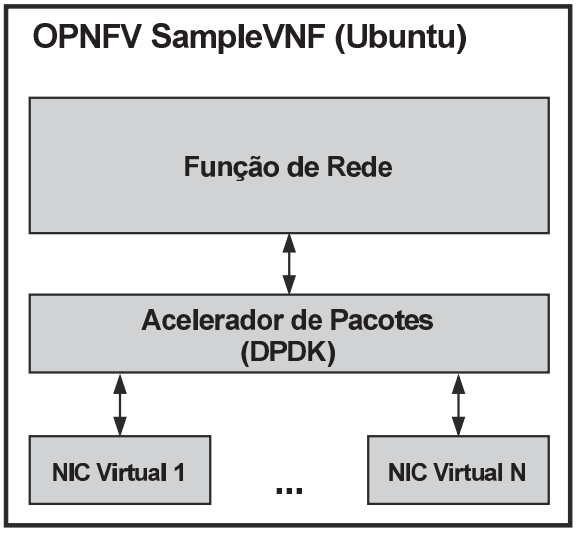


Figura 3.4: Arquitetura Geral do SampleVNF

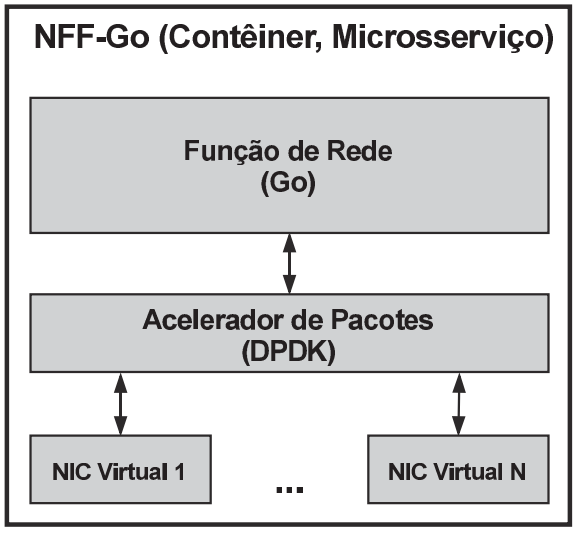


Figura 3.5: Arquitetura Geral do NFF-Go

programação, o NFF-Go agiliza o desenvolvimento de funções de rede, contribuindo ativamente com o paradigma NFV. Além das bibliotecas de elementos, a plataforma ainda oferece uma coleção de funções de rede que podem ser diretamente usadas ou personalizadas para executarem tarefas particulares. Entre as funções disponíveis estão *Intrusion Prevention System* (IPS), *Deep Packet Inspector* (DPI), *firewall* e *forwarder*.

As plataformas de execução de VNF relatadas até agora apresentam um modelo operacional genérico, ou seja, são abrangentes em relação às funções de rede suportadas e suas instanciações são independentes e autocontidas (*i.e.*, todos os recursos necessários para a execução da plataforma estão presentes na instância virtualizada da mesma). Porém, a utilização de outros modelos operacionais também é possível. Entre esses outros modelos estão as plataformas orientadas a SFC. Essas plataformas tipicamente objetivam o aprimoramento de desempenho e redução de demanda de recursos computacionais no processamento de pacotes por uma cadeia de serviço. Tais aprimoramentos são alcançados, por exemplo, através do compartilhamento de memória entre instâncias de VNF, acelerando a transmissão dos dados, desonerando a rede e permitindo a adoção de estratégias *zero-copy* (*i.e.*, evitar que múltiplas cópias de um mesmo dado sejam feitas ou mantidas no sistema). É comum que nesse modelo de plataforma não sejam discutidos meios específicos de desenvolvimento de funções de rede (*e.g.*, linguagens de programação e ferramentas associadas) e de gerenciamento do ciclo de vida das instâncias de VNF, sendo tais características definidas particularmente pelos seus operadores. A seguir são apresentadas quatro plataformas de execução de VNF orientadas a SFC: NetVM, OpenNetVM, HyperNF e MVMP.

O **NetVM** (Hwang et al., 2015) foi o precursor das plataformas de execução de VNF orientadas a SFC. Essa plataforma emprega virtualização completa provida pelo hipervisor KVM para criar seu ecossistema de execução. Também, o acelerador de pacotes DPDK (DPDK, 2021) é utilizado para gerenciamento das interfaces físicas de rede (*i.e.*, está presente no sistema operacional que habilita o ambiente virtualizado, e não internamente nas instâncias virtuais). É no contexto do DPDK que um agente NetVM gerencia e encaminha o tráfego de rede para as instâncias de VNF. Esse encaminhamento de tráfego acontece através de duas regiões de memória compartilhada: memória de descritores e memória de pacotes. A memória de descritores é normalmente pequena e estabelecida individualmente entre o hipervisor e cada uma das instâncias de VNF. É a partir dessa memória que as instâncias virtuais tomam conhecimento da existência de pacotes a serem processados por elas, além de adquirem informações da localização dos mesmos.

Essa localização se refere a uma posição na memória de pacotes, que consiste nas chamadas *huge pages*, que podem ser compartilhadas entre um grupo de instâncias de VNF confiáveis (*e.g.*, uma cadeia de serviço). É nessas *huge pages* que o agente NetVM aloca os pacotes recebidos pelo sistema, os mantendo nelas até que o seu processamento cesse ou que a próxima instância de VNF a processá-los não pertença ao mesmo grupo de confiança das anteriores. O NetVM mantém um canal de comunicação tanto com hipervisores como com instâncias de VNF. Nesse canal são informadas/detectadas inclusões ou exclusões de instâncias de VNF no sistema, disparando alocações ou liberações de memória compartilhada. Instâncias de VNF recém-incluídas podem fazer parte de uma ou mais cadeias de serviço. Nesse caso, o NetVM insere as políticas de encaminhamento dessas instâncias em uma tabela de fluxo, consultada antes da transmissão de pacotes no sistema.

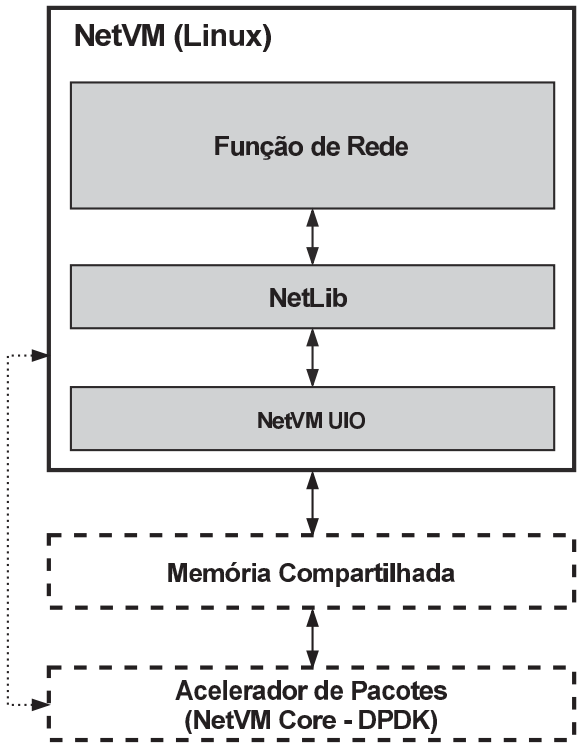


Figura 3.6: Arquitetura Geral do NetVM

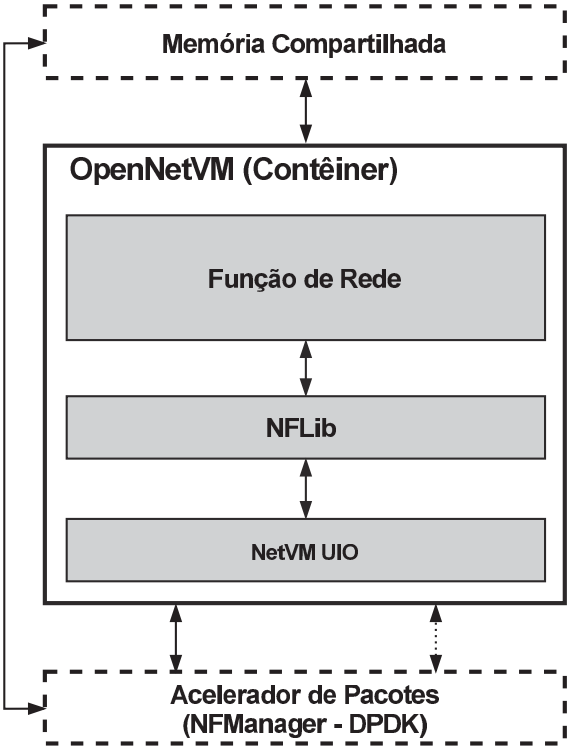


Figura 3.7: Arquitetura Geral do OpenNetVM

A Figura 3.6 ilustra a arquitetura geral do ecossistema NetVM, sendo que o núcleo da plataforma de execução de VNF é delimitado pelas margens do elemento identificado como "NetVM". A plataforma é compatível com ambientes Linux (nenhuma restrição é feita em relação a distribuições) e executa aplicações específicas de integração no sistema de memória compartilhada. Entre essas aplicações destacam-se a NetVM UIO e a NetLib. A primeira (NetVM UIO) é um *driver* que efetua o reconhecimento das regiões de memória compartilhada no sistema operacional hospedeiro do hipervisor. Além disso, ela também mapeia essas regiões de memória para se tornarem acessíveis ao domínio de usuário das instâncias de VNF. É importante ressaltar que, no caso particular do NetVM, essa aplicação interpreta a memória compartilhada como um dispositivo PCI conectado à máquina virtual. Já a NetLib é responsável pela criação e gerenciamento do canal de comunicação entre a memória de pacotes e a função de rede sendo executada na plataforma. De maneira prática, a NetLib monitora o espaço de memória compartilhado e, sempre que um novo pacote é identificado, este é passado como argumento em uma chamada de *callback* referente à função de rede implantada. Após a execução da *callback*, o seu retorno é interceptado pela NetLib que determina então a próxima ação a ser realizada

pelo agente NetVM. Exemplos dessas ações são: “transmita através da interface física de rede”, “encaminhe para a próxima VNF” ou “descarte o pacote”.

O **OpenNetVM** (Zhang et al., 2016b) foi proposto inspirado na arquitetura do NetVM, porém considerando um contexto de virtualização leve baseada em contêineres. Como mostra a Figura 3.7, essa plataforma de execução de VNF apresenta diversas semelhanças arquiteturais ao NetVM, como o uso de memória compartilhada e do acelerador de pacotes DPDK para acesso às interfaces físicas de rede, além da mesma organização nuclear das instâncias de VNF, sendo a NFLib (OpenNetVM) equivalente a NetLib (NetVM). Uma diferença fundamental entre OpenNetVM e NetVM consiste dos seus respectivos modelos de gerenciamento da memória compartilhada. Enquanto no NetVM é necessário estabelecer dispositivos PCI virtuais para habilitar o acesso das instâncias de VNF a essa memória, o OpenNetVM realiza apenas o informe dos endereços das memórias de descritores e de pacotes aos contêineres hospedeiros de VNF que, assim, podem acessá-los de maneira natural. Apesar do gerenciamento de memória compartilhada ser simplificado no OpenNetVM, todas as instâncias de VNF têm acesso integral às regiões de memória compartilhada. Essa dinâmica impõe a necessidade de validação operacional das funções de rede executadas, garantindo que todas sejam confiáveis e não prejudiciais ao sistema. Ademais, processos relacionados ao encaminhamento de pacotes, tabelas de fluxo e inclusão/exclusão de instâncias virtualizadas no sistema e nas cadeias de serviço são os mesmos em relação ao NetVM.

A plataforma de execução de VNF orientada a SFC **HyperNF** (Yasukata et al., 2017), ilustrada em alto nível na Figura 3.8, é paravirtualizada com um sistema operacional baseado em Linux contendo várias aplicações dedicadas. Apesar de realizar um processo de instanciação bastante genérico, a execução do HyperNF é dependente de hipervisores particulares associados a aceleradores de pacotes específicos. Atualmente, o hipervisor Xen aliado ao acelerador de pacotes *netmap* (Rizzo, 2012), assim como para o hipervisor KVM em conjunto com o acelerador de pacotes *ptnetmap* (Garzarella, 2021) são suportados pela plataforma. Diferente de outras plataformas orientadas a SFC, que utilizam o modelo de *split* no gerenciamento das interfaces virtualizadas de rede (*i.e.*, uma CPU física é dedicada ao *Input/Output* – I/O), o HyperNF adota um modelo descentralizado onde cada instância virtual é responsável pelo próprio I/O. A estratégia adotada pelo HyperNF exclui o paralelismo entre I/O e processamento (derivado do modelo *split*), demandando interrupções para a obtenção de pacotes provenientes das interfaces virtuais. Essa dinâmica resulta em perdas de ciclos de CPU e, por consequência, degradação imediata e incremental da vazão de, respectivamente, instâncias de VNF e serviços de rede. Entretanto, os autores alegam que o HyperNF reduz significativamente o desperdício de recursos computacionais, evitando a ociosidade de processadores e permitindo o ajuste fino no *scheduling* (*i.e.*, modo de distribuição do tempo de processamento) dos mesmos. Para reduzir as perdas de tempo de CPU, diretivas de encaminhamento de pacote são implementadas em nível de hipervisor (acessadas pelas instâncias de VNF através do *hypercall driver*) e um *switch* virtual VALE é instanciado como uma máquina virtual privilegiada pelo hipervisor (*dom0*). Essas modificações permitem que o I/O das instâncias virtuais seja feito exclusivamente através de chamadas de hipervisor, sem a necessidade de mudança de contexto entre máquinas virtuais e o sistema hospedeiro. Assim, o custo de cada interrupção para I/O é reduzido em até dezenas de vezes, tornando o HyperNF um sistema especialmente viável para otimizar a alocação de recursos computacionais para funções e serviços de rede.

O **MVMP** (Zheng et al., 2018) é uma plataforma de execução de VNF proposta com base na arquitetura operacional do OpenNetVM, como demonstrado na Figura 3.9. Entre as semelhanças dessas plataformas estão o uso de virtualização baseada em contêineres, do acelerador de pacotes DPDK e de espaços de memória compartilhada para encaminhamento de

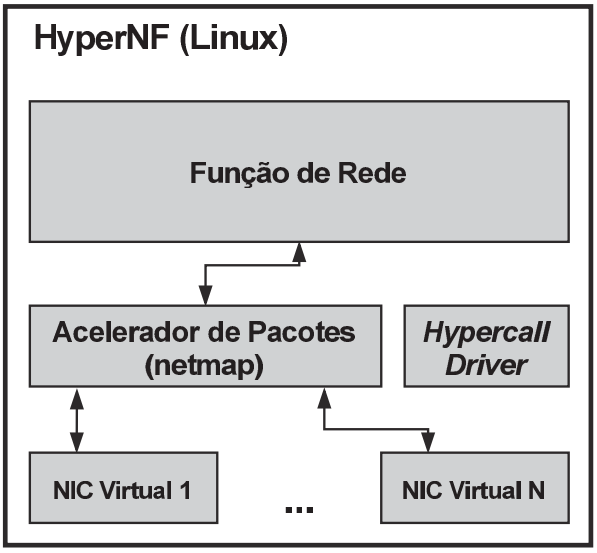


Figura 3.8: Arquitetura Geral do HyperNF

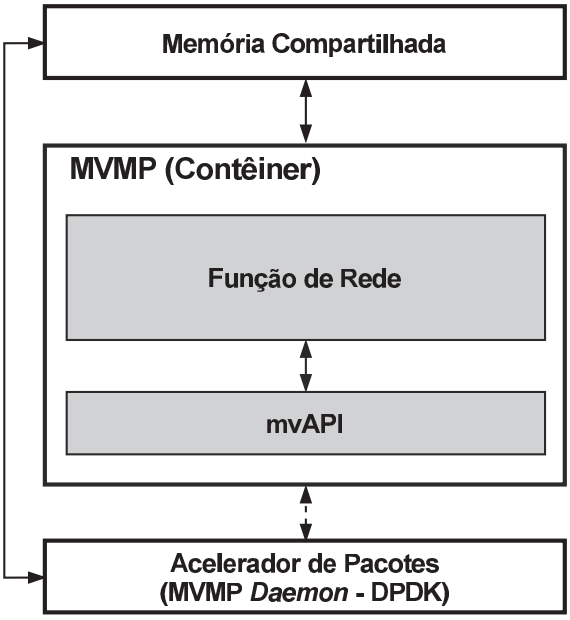


Figura 3.9: Arquitetura Geral do MVMP

pacotes entre instâncias de VNF de um serviço de rede. Assim como o OpenNetVM, o MVMP implementa toda a lógica de encaminhamento de tráfego e gerenciamento de instâncias virtuais através de uma aplicação executando no contexto do DPDK. A principal contribuição do MVMP consiste no suporte a serviços de rede com topologias ramificadas, permitindo a divisão ou cópia do tráfego em caminhos distintos em um mesmo serviço. Para que esse processo seja possível, o MVMP emprega uma estratégia de múltiplos dispositivos virtuais de rede. Esses dispositivos observam políticas de encaminhamento que determinam o(s) próximo(s) salto(s) de um pacote dentro de uma cadeia de serviço. Uma instância de VNF abriga múltiplos dispositivos virtuais que podem encaminhar pacotes por dois ou mais caminhos simultâneos. Nesse caso, operações de escrita em pacotes não devem ser executadas arbitrariamente visto que estes estão sendo usados por diversas funções de rede. Como solução, o MVMP emprega uma memória compartilhada de três regiões: memória de descritores, memória de pacotes da plataforma, memória de pacotes de NF. A memória de descritores é pública, acessível com funções da biblioteca mvAPI e tem a mesma finalidade da memória de descritores do OpenNetVM. Pacotes armazenados na memória de pacotes da plataforma não passam por operações de escrita, mantidos disponíveis até que nenhuma função de rede recorra aos mesmos. Já pacotes inseridos na memória de pacotes de NF são cópias feitas sob demanda de fragmentos da memória de pacotes da plataforma, acessíveis para escrita por funções de rede. Após alterados, pacotes na memória de NF podem ser gravados na memória de plataforma ou diretamente encaminhados na memória de NF para outra função de rede. Essa estratégia permite que as instâncias virtualizadas dispensem pilhas de rede (como a NetVM UIO), utilizando assim espaços de memória dinamicamente alocados em regiões compartilhadas. Finalmente, uma terceira particularidade do MVMP é a invocação ativa da mvAPI pelas funções de rede para recebimento de pacotes (*i.e.*, não utiliza a estratégia de *callbacks* do OpenNetVM). Dessa forma, a plataforma consegue prover recursos para processamento de tráfego em lote, visando o aprimoramento da vazão do sistema.

Um terceiro modelo aplicável ao desenvolvimento de plataformas de execução de VNF, chamado de orientado a domínios específicos, contempla implementações que objetivam atender requisitos particulares relacionados a recursos computacionais, desempenho, implantação e protocolos de determinadas aplicações e tecnologias, como *Internet of Things* (IoT), segurança,

big data e redes 5G. Entre as plataformas de execução de VNF orientadas a domínios específicos existe a **CliMBOS** (Gallo et al., 2018). Essa plataforma é uma adaptação do ClickOS destinada ao desenvolvimento de funções de rede virtualizadas para IoT. O CliMBOS consiste em um sistema *unikernel* (MiniOS) aliado ao acelerador de pacotes *netmap* (Rizzo, 2012), virtualizado através do hipervisor Xen e monitorado a partir de uma interface via *socket* (características herdadas do ClickOS). Essa plataforma suporta a execução de funções de rede implementadas em Click Middleboxes (CliMB) (Laufer et al., 2016), uma coletânea estendida de elementos do CMR que permite ao desenvolvedor de funções atuar em camadas superiores à camada três. A partir desses novos elementos, o CliMBOS provê o ferramental necessário para a implementação de protocolos (*e.g.*, *Constrained Application Protocol* – CoAP) e de semântica fim-a-fim (*i.e.*, comunicação, interpretação e tomada de decisões a partir de informações de vários e diferentes sensores) no contexto de IoT. A arquitetura interna do CliMBOS é apresentada na Figura 3.10. Arquiteturalmente, o CliMBOS se diferencia pela utilização de módulos internos relacionados ao controle e verificação nativa de conexões fim-a-fim (Camada TCP e Servidor TCP), provendo recursos de comunicação confiável na plataforma.

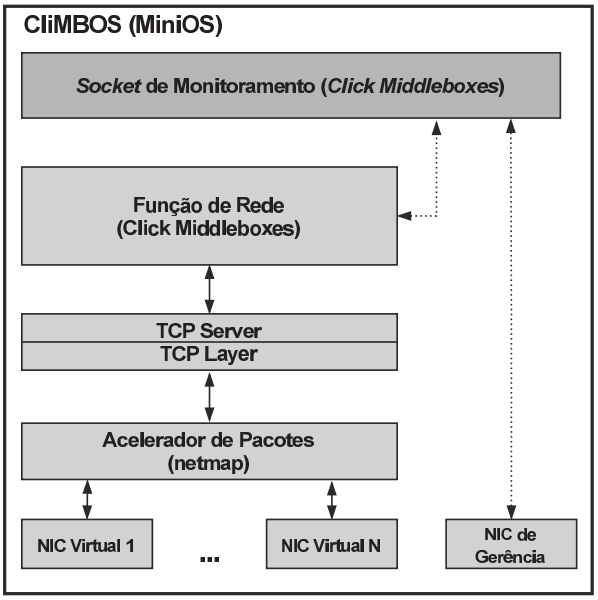


Figura 3.10: Arquitetura Geral do CliMBOS

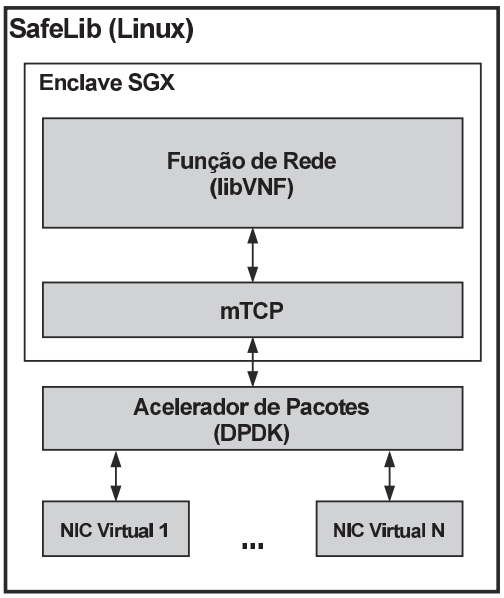


Figura 3.11: Arquitetura Geral do SafeLib

A **SafeLib** (Marku et al., 2019) é uma plataforma de execução de VNF orientada a domínio específico. A plataforma em questão utiliza virtualização completa, criando máquinas virtuais baseadas em Linux. Essas máquinas contam com o acelerador de pacotes DPDK (DPDK, 2021) em conjunto com uma pilha de rede denominada mTCP, ambos utilizados para aprimoramento de vazão. A plataforma suporta a execução de funções de rede desenvolvidas através do ferramental provido pela libVNF (Naik et al., 2018), mas nenhuma discussão é feita em relação ao gerenciamento e monitoramento do ciclo de vida dessas funções. A SafeLib visa disponibilizar um ambiente seguro de execução de funções de rede através do emprego de enclaves *Software Guard Extensions* (SGX) (Costan e Devadas, 2016). Um enclave consiste em uma região de memória criptografada diretamente pelo processador, sendo seu conteúdo inacessível por qualquer processo fora dele (independentemente dos níveis de privilégio desses processos). Além disso, para garantir a integridade dos fluxos de pacotes, o enclave também inclui a pilha de rede (mTCP) acessada pela função, como demonstra a arquitetura interna da plataforma na Figura 3.11. Entre os objetivos de projeto da SafeLib estão a garantia da confidencialidade de dados sensíveis, da capacidade de implementação de funções de rede genérica de camada

dois até camada quatro, do desempenho no processamento de tráfego de rede, do baixo custo de implantação/migração e da facilidade de implementação de funções de rede. Essa plataforma tem especial foco em segurança, isolamento e integridade, sendo assim especialmente adequada quando funções de rede são implantadas em domínios de virtualização não confiáveis ou quando o tráfego processado é demasiadamente sensível e sujeito a ataques.

Tabela 3.1: Sumarização das Plataformas de Execução de VNF

Plataforma de Execução de VNF	**Modelo**			**Virtualização**	**Aceleradores de Pacotes**	**Desenvolvimento de Funções de Rede**	**Gerenciamento e/ou Monitoramento Interno**
	Genérico	Orientado a SFC	Orientado a Aplicação				
ClickOS (Martins et al., 2014)	X			Paravirtualização	*netmap*	Click Modular Router	Socket de Controle
Click-On-OSv (Marcuzzo et al., 2017)	X			Paravirtualização	DPDK	Click Modular Router	RESTful
Leaf (Flauzino et al., 2020)	X			Virtualização completa	*N/A*	*N/A*	RESTful
SampleVNF (OPNFV, 2021)	X			Virtualização completa	DPDK	*N/A*	*N/A*
NFF-Go (Intel, 2021)	X			Conteinerização Micro-serviços	DPDK	Go	*N/A*
NetVM (Hwang et al., 2015)		X		Virtualização completa	DPDK	*N/A*	*N/A*
OpenNetVM (Zhang et al., 2016b)		X		Conteinerização	DPDK	*N/A*	*N/A*
HyperNF (Yasukata et al., 2017)		X		Virtualização completa	*netmap*	*N/A*	*N/A*
MVMP (Zheng et al., 2018)		X		Conteinerização	DPDK	*N/A*	*N/A*
CliMBOS (Gallo et al., 2018)			X	Paravirtualização	*netmap*	Click Middleboxes	Socket de Controle
SafeLib (Marku et al., 2019)			X	Virtualização completa	DPDK	libVNF	*N/A*

A Tabela 3.1 sumariza as principais características das plataformas de execução de VNF apresentadas no decorrer desta seção. A partir dessas características é possível apontar a ampla exploração de tecnologias de virtualização diferentes, sendo cinco plataformas instanciadas como máquinas virtuais completas, três como máquinas paravirtualizadas e três como contêineres ou microsserviços (virtualização leve). Além disso, a plena adoção de aceleradores de pacotes (91% das plataformas) evidencia a necessidade dessas ferramentas para permitir que funções de rede virtualizadas atinjam níveis de vazão competitivos aos equipamentos legados. Entre as plataformas que utilizam aceleradores de pacotes, o DPDK foi o mais adotado (70%), sendo o *netmap* a segunda opção (30%). Menos da metade das plataformas de execução de VNF (45%) definiram as ferramentas, bibliotecas e linguagens de programação suportadas pelas mesmas, para as demais não houve discussão, ficando a responsabilidade de preparação da plataforma para a execução de tais ferramentas transferida aos seus operadores. Finalmente, poucos esforços foram realizados na oferta de agentes de gerenciamento e monitoramento do ciclo de vida interno das instâncias de VNF, sendo que apenas duas plataformas proveem suporte para operações de gerenciamento e monitoramento (18%) e outras duas apenas para operações de monitoramento (18%).

3.2 IMPLEMENTAÇÕES DO EMS

Por definição, o elemento operacional EMS pertence ao domínio de trabalho de VNF da arquitetura de referência NFV. Todavia, a implementação desse elemento está tipicamente associada a projetos de NFV-MANO, como o OpenStack Tacker (OpenStack, 2021), Open Source MANO (ETSI, 2021), Open Baton (Berlin, 2021) e CloudStack Vines (Flauzino et al., 2020). Na prática, esse fato ocorre devido ao motivo descrito a seguir. É comum que sejam empregues operações nativas dos próprios sistemas NFV-MANO para gerenciar o ciclo de vida das funções de rede através de um EMS. Além disso, sistemas NFV-MANO incluem protocolos de comunicação específicos e não padronizados que são usados na comunicação entre EMS e VNFM. Apesar de ser uma prática conveniente, a não utilização do protocolo

padrão da interface Ve-Vnfm-em (NFVISG, 2020a) limita a portabilidade e integração das soluções de EMS, que não podem ser executadas em ambientes NFV heterogêneos em termos do NFV-MANO (VNFM) adotado. Assim o EMS acaba vinculado ao próprio NFV-MANO, em particular ao VNFM, não sendo frequentemente disponibilizado de forma genérica no domínio de VNF propriamente dito. Além disso, os operadores de tais soluções de EMS lidam com falta de uniformidade de suas capacidades operacionais, impedindo, por exemplo, a programação de *scripts* genéricos e abrangentes utilizáveis independentemente da implementação do domínio de VNF e de NFV-MANO do ambiente NFV acessado.

O OpenStack Tacker (OpenStack, 2021) é um sistema NFV-MANO que conta com um VNFM e NFVO nativos, além de utilizar a nuvem OpenStack como seu VIM/VI. O OpenStack Tacker não disponibiliza um EMS como parte de seu sistema. Entretanto, para possibilitar a execução de operações de configuração interna inicial das funções de rede, o Tacker (em conjunto com o OpenStack Neutron) permite o envio de arquivos executáveis para instâncias de VNF que suportam o serviço chamado Cloud-Init (Canonical, 2021a). O Cloud-Init é instalado nas instâncias virtuais, sendo requisitado através de um conjunto de *scripts* específicos conhecidos como Cloud-Config. O Cloud-Config adota um modelo de *scripts* de alto nível escritos em *YAML Ain't Markup Language* (YAML), permitindo requisitar desde comandos de terminal e *bash scripts*, até utilizar recursos previamente disponíveis no sistema operacional sendo carregado, como interpretadores e compiladores de linguagens de programação, para executar programas variados.

O serviço Cloud-Init não é considerado um EMS completo, mas sim um agente de configuração para execução de *scripts* que, de forma independente, realizam operações internas em uma instância de VNF. Feitas essas observações e considerando apenas as suas características de configuração, o Cloud-Init é categorizado como um EMS **não-MANO, interno e dedicado**. A Figura 3.12 apresenta um fluxo abstraído de troca de mensagens entre VFNM (Tacker) e Cloud-Init durante a configuração de uma instância de VNF. Após o envio do documento de configuração (Cloud-Config) por parte do VNFM (Tacker), o Cloud-Init realiza sua validação e aplica as diretivas de configuração nele descritas na máquina virtual. Por fim, o Cloud-Init confirma o sucesso ou falha do processo de configuração para o VNFM.

O OSM é um projeto gerido pela ETSI que oferece um sistema NFV-MANO através de contêineres (ETSI, 2021). Esse sistema utiliza tanto nuvens OpenStack quanto contêineres Kubernetes como seu VIM/VI. O OSM não dispõe de um EMS nativo, mas oferece um agente de configuração interna através do serviço Day-Like. Esse serviço divide o processo de configuração das instâncias de VNF em três etapas, referidas como três dias. O dia zero diz respeito à criação e operacionalização da instância virtual, essa etapa não faz parte das tarefas de configuração executadas por um EMS, porém viabiliza os recursos para estas serem iniciadas. O dia um objetiva configurar a instância virtual como uma VNF, iniciando a sua função de rede. Entre as ações realizadas durante essa etapa estão a instalação das dependências de VNF (*e.g.*, linguagens de programação, bibliotecas e aplicações específicas), a implantação da função de rede na plataforma e a sua configuração inicial, caso necessária (*e.g.*, definição de regras de *firewall* e construção do conjunto inicial de nomes de um *Network Address Translator* – NAT). Por fim, durante o dia dois, o agente de gerência da plataforma de VNF é instalado e inicializado. Entretanto, quando o agente de gerência é originalmente provido na plataforma, como no ClickOS (Martins et al., 2014), Click-On-OSv (Marcuzzo et al., 2017) ou Leaf (Flauzino et al., 2020), essa etapa é dispensável.

Assim como o Cloud-Init, a execução do serviço Day-Like depende de ferramentas pré-instaladas nas plataformas de execução de VNF. Dessa forma, aplicando as mesmas restrições consideradas para o Cloud-Init do OpenStack, o serviço Day-Like pode ser categorizado como

um EMS **não-MANO, interno e dedicado**. A Figura 3.13 ilustra o fluxo abstraído de troca de mensagens entre o VNFM (OSM) e o serviço Day-Like na efetivação da configuração inicial de uma instância de VNF. Destaca-se que a principal diferença entre o fluxo apresentado para o Cloud-Init (Figura 3.12) e o fluxo apresentado para o Day-Like consiste da possibilidade de, no último, instalar (durante o dia dois) um agente para o gerenciamento do elemento virtualizado.

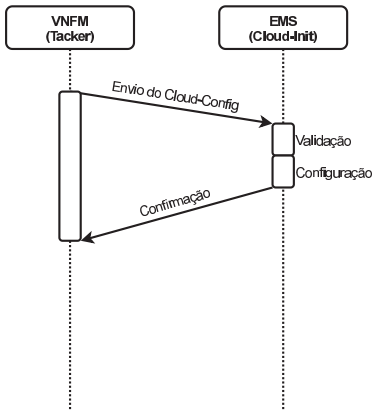


Figura 3.12: Fluxo de Mensagens Abstraído do Cloud-Init

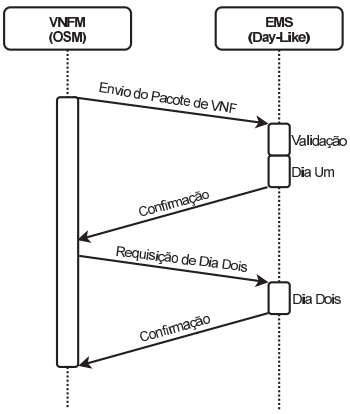


Figura 3.13: Fluxo de Mensagens Abstraído do Day-Like

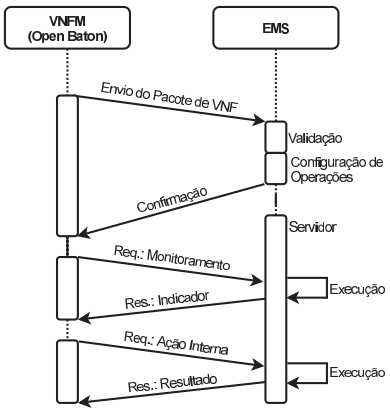


Figura 3.14: Fluxo de Mensagens Abstraído do EMS do Open Baton

O Open Baton (Berlin, 2021) é um sistema NFV-MANO com VNFM, NFVO e suporte ao OpenStack como seu VIM/VI. Além disso, o Open Baton permite a implementação de *drivers* para habilitar o uso de outros provedores de nuvem como VIM/VI. O Open Baton possui um EMS que é inserido internamente às plataformas de execução de VNF. O EMS é programável e permite a utilização de executáveis providos através do campo chamado *scripts-link* do pacote de VNF reconhecido pelo Open Baton. Acessando esses dados, o EMS define operações internas de gerenciamento e disponibiliza as mesmas através de interfaces REST ou AMQP (*Advanced Message Queuing Protocol*). Após ser inicialmente configurado pelo pacote de VNF utilizado na implantação da função de rede, atualizações podem ser realizadas no EMS em tempo de execução. Essas atualizações ocorrem tanto pela substituição completa do pacote de VNF, quanto pela substituição direta de um executável relativo a uma operação de EMS em particular. Diferente do OpenStack Tacker e do OSM, o EMS do Open Baton mantém-se disponível mesmo após a fase de configuração da instância de VNF, sendo considerado uma solução completa. Dessa forma, esse EMS é categorizado como **MANO (independente), interno, restrito e dedicado**. A Figura 3.14 exemplifica o fluxo abstraído de troca de mensagens entre o EMS e o Open Baton para a realização da configuração inicial da instância de VNF e sua posterior manutenção e monitoramento.

O sistema NFV-MANO CloudStack Vines disponibiliza implementações dos elementos VNFM e NFVO, ambos projetados para atuarem em conjunto com nuvens CloudStack (VIM/VI) (Flauzino et al., 2020). O Vines apresenta um EMS desenvolvido como um elemento de rede do CloudStack, sendo nativamente integrado ao ambiente de nuvem e compatível com os elementos do NFV-MANO. Originalmente, o EMS provido pelo Vines foi desenvolvido para operar com a plataforma de execução de VNF Leaf (também proposta no contexto do CloudStack Vines). Entretanto, é possível definir *drivers* estendidos que expandem a atuação do EMS para outras plataformas de execução de VNF. Esse EMS é segmentado em seis módulos operacionais que desempenham funções específicas: (i) interface de acesso, através da qual são feitos o envio e recebimento de requisição de operações de EMS; (ii) controle de acesso, que realiza a autenticação de requisições recebidas tendo em vista suas origens; (iii) base de informações de VNF, que consiste em uma base de dados relacionados a todas as instâncias de VNF gerenciadas pelo

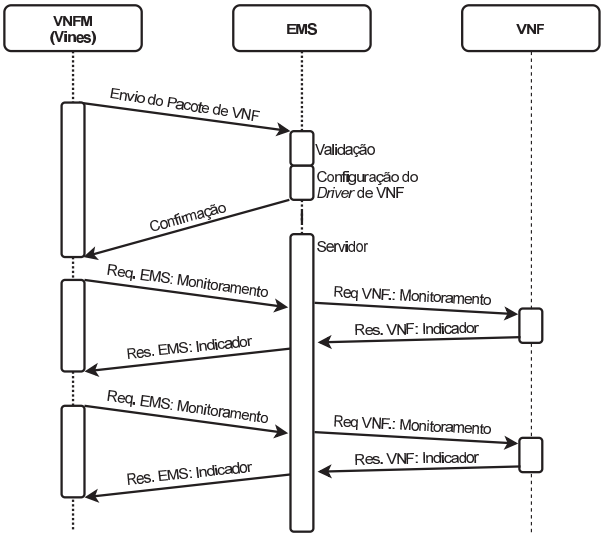


Figura 3.15: Fluxo de Mensagens Abstraído do EMS do CloudStack Vines

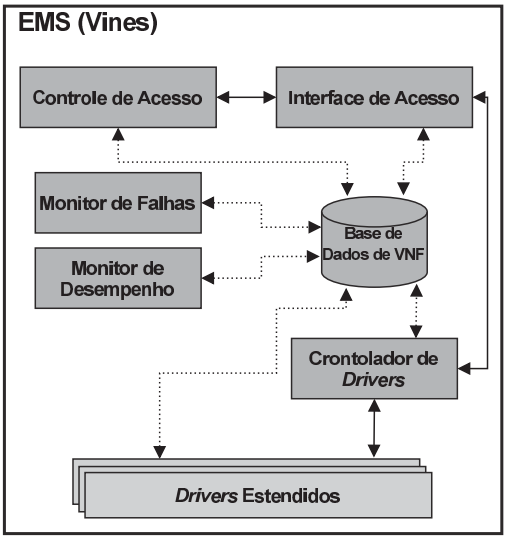


Figura 3.16: Arquitetura Geral do EMS do CloudStack Vines

EMS; (iv) monitor de desempenho, que monitora políticas de desempenho das instâncias de VNF; (v) monitor de falhas, que monitora o correto funcionamento das instâncias de VNF; e (vi) controlador de *drivers*, que gerencia e aloca *drivers* de comunicação entre EMS e plataformas de execução de VNF.

Considerando as características apresentadas, o EMS do CloudStack Vines é categorizado como **MANO (VNF), externo, restrito e guarda-chuva**. A Figura 3.15 apresenta os fluxos abstraídos de troca de mensagens com o EMS do Cloudstack Vines, demonstrando sua configuração inicial (*i.e.*, registro de uma instância de VNF) e posterior requisição de operações. Note que esse diagrama de sequência, dentre aqueles apresentados nesta seção, é o único que registra a linha de vida da VNF. Isso ocorre visto que o EMS do Vines é categorizado como externo, atuando fora do sistema hospedeiro da instância de VNF. Também, esse EMS é o único construído de maneira modular, possibilitando sua representação arquitetural, como exibida na Figura 3.16, com os módulos descritos anteriormente.

Tabela 3.2: Sumarização das Soluções de EMS e Serviços Similares

	Integração		**Localização**	**Gerenciamento**	**Abragência**
	MANO	***não-MANO***			
OpenStack Tacker (Cloud-Init)		X	Interno	*N/A*	Dedicado
Open Source Mano (Day-Like)		X	Interno	*N/A*	Dedicado
Open Baton	X (Independente)		Interno	Restrito	Dedicado
CloudStack Vines	X (VNF)		Externo	Restrito	Guarda-chuva

A Tabela 3.2 resume as categorizações e classificações das soluções de EMS e sistemas de configuração abordados nesta seção. Considerando a integração, a classificação das soluções como MANO e não-MANO é balanceada, sendo 50% das soluções analisadas compreendidas em cada uma dessas categorias. Também, entre as soluções com integração MANO, metade são classificadas como independentes, enquanto a outra metade são exemplares da classe de

VNF. Nenhuma solução de EMS MANO é classificada como VNFC. Em relação à localização, 75% das soluções analisadas são internas à instância de VNF. Tipicamente, a incorporação do elemento EMS à plataforma de execução de VNF simplifica a execução dos processos de gerência, visto que, dessa forma, o EMS acessa diretamente o sistema hospedeiro da função da rede. Entretanto, essa abordagem exige um preparo prévio das plataformas de execução de VNF para disponibilizarem o EMS nativamente, resultando em soluções monolíticas e pouco flexíveis. Características da categoria de gerenciamento não são aplicáveis ao Cloud-Init e Day-Like visto que estes são considerados serviços de configuração (*i.e.*, não são soluções de EMS completas). As demais soluções são categorizadas como restritas por oferecerem operações de gerenciamento individualizadas de plataformas de execução de VNF específicas. Porém, é importante ressaltar que essas soluções de EMS não disponibilizam a totalidade das operações previstas na interface Ve-Vnfm-em (NFVISG, 2020a). Por fim, a categoria de abrangência é majoritariamente definida como dedicada. Esse cenário ocorre principalmente pela predominância de soluções de EMS com localização interna à instância de VNF, sendo consequentemente dedicada à mesma. Por outro lado, a solução de EMS do CloudStack Vines é categorizada como guarda-chuva, podendo atuar em múltiplas instâncias de VNF simultaneamente.

3.3 SOLUÇÕES DE MAPEAMENTO MULTIDOMÍNIO

Diferentes mapeamentos em múltiplos domínios são capazes de prover um serviço de rede com sucesso. Entretanto, considerando um conjunto de políticas, restrições e métricas de interesse, o desempenho verificado em mapeamentos distintos é tipicamente diferente (Herrera e Botero, 2016). Assim, soluções que visam determinar mapeamentos válidos enquanto qualificam os mesmos se tornam necessárias. Em geral, as soluções de mapeamento multidomínio podem ser divididas em duas classes principais: (i) centralizadas e (ii) distribuídas. Soluções centralizadas são executadas em um único ponto de processamento. Essas recuperam todas as informações necessárias relativas ao serviço requisitado e aos domínios disponíveis, as avaliam considerando um conjunto de métricas de interesse e retornam um ou mais mapeamentos adequados a elas. Por outro lado, soluções distribuídas espalham a requisição de integração dos elementos virtualizados entre os domínios disponíveis através de um protocolo de comunicação comum aos mesmos. Neste caso, cada domínio avalia a requisição para determinar segmentos que podem ser hospedados por eles, encaminhando o restante aos domínios vizinhos. Soluções centralizadas são propostas em (Dietrich et al., 2015; Riera et al., 2016; Wang et al., 2017), enquanto que soluções distribuídas são apresentas em (Abujoda e Papadimitriou, 2016; Zhang et al., 2016a).

Em (Dietrich et al., 2015) informações ofuscadas (*i.e.*, dados agregados através de uma peso) sobre a adequabilidade de alocação de funções de rede nos provedores de infraestrutura são utilizadas. Assim, essa solução procede a otimização do mapeamento do serviço considerando quatro critérios fixos: minimização dos (i) custos financeiros, (ii) do número de provedores e domínios diferentes utilizados, (iii) utilização de recursos computacionais e maximização da (iv) adequabilidade das funções de rede. Já em (Riera et al., 2016), uma plataforma de orquestração de serviços de rede multidomínio, chamada TeNOR, é proposta. A plataforma TeNOR conta com uma solução de mapeamento multidomínio nativa. Essa solução centraliza informações de custos financeiros, atraso de transmissão e utilização de recursos computacionais, que utiliza como entradas para sua política de minimização que decide o mapeamento das funções de rede de uma topologia serviço requisitada. Finalmente, em (Wang et al., 2017), o mapeamento ocorre em um cenário híbrido onde tanto domínios privados quanto de provedores disponibilizam recursos relacionados a redes ópticas. A solução proposta visa determinar mapeamentos que minimizem

custos financeiros (privilegiando assim domínios privados) e a utilização de *slots* de frequência nos canais ópticos.

Na solução proposta em (Zhang et al., 2016a) uma metodologia centrada em vértices é utilizada para realizar o mapeamento multidomínio. Essa solução realiza rodadas de trocas de mensagens entre provedores para determinar mapeamentos possíveis iterativamente. Além disso, uma política que limita a quantidade de alocações de função de rede a cada rodada de troca de mensagens é aplicada, evitando assim monopolização da topologia de serviço por um único provedor. Nessa solução, nenhuma otimização específica é realizada, sendo retornado ao usuário apenas um conjunto de mapeamentos possíveis que atendem aos requisitos de instanciação das funções de rede que compõem o serviço requisitado. Utilizando um processo semelhante ao apresentado em (Zhang et al., 2016a), a solução DistNSE (Abujoda e Papadimitriou, 2016) determina mapeamentos candidatos de forma distribuída e os avalia considerando uma função de otimização que visa a minimização do custo financeiro total e balanceamento de carga interdomínio. Porém, diferente da solução em (Zhang et al., 2016a), não existe limitação no número de alocações de funções de rede feitas por provedores a cada rodada de troca de mensagens.

Em (Li et al., 2020) é proposta uma solução de mapeamento multi-domínio baseada em um algoritmo genético mono-objetivo, chamada GA+LCB. O propósito dessa solução é alocar as funções de rede de um serviço virtualizado em um ambiente multidomínio com base em um único indicador (E). Esse indicador, no que lhe concerne, representa várias métricas de domínio, como disponibilidade de rotas, largura de banda, número de funções de rede que cada domínio pode hospedar, entre outras. Além de mapear o serviço de rede em vários domínios, a solução proposta também fornece um esquema de *backup* para as funções de rede mapeadas. O esquema de *backup* objetiva melhorar a confiabilidade geral do serviço. No entanto, também é possível mapear as funções de rede ignorando a criação de *backups*.

A solução proposta em (Rodis e Papadimitriou, 2021) emprega um algoritmo genético mono-objetivo para mapear serviços de rede virtualizada em nós de processamento no substrato físico. A solução visa otimizar o uso de recursos de computação e rede. Desta forma, dado um conjunto de serviços a serem mapeados, os autores propõem uma função objetivo que minimiza a capacidade residual dos nós para hospedar funções e rotas de comunicação. A função objetivo consiste na minimização de um único indicador denominado C_m. Esse indicador geralmente obtém os melhores resultados quando todas as funções de rede são mapeadas para um único nó de substrato.

Tabela 3.3: Sumarização de Características das Soluções de Mapeamento

	Solução de Mapeamento (Referência)						
	(Dietrich et al., 2015)	(Riera et al., 2016)	(Wang et al., 2017)	(Abujoda e Papadimitriou, 2016)	(Zhang et al., 2016b)	(Li et al., 2020)	(Rodis e Papadimitriou, 2021)
Classe	Centralizado	Centralizado	Centralizado	Distribuído	Distribuído	Centralizado	Centralizado
Heurística	Mapeamento de Subgrafo	N/A	*K-Cut*	*Time to Live*	N/A	Heurística Genética	Heurística Genética
Métricas de Otimização	Custos Financeiros Quantidade de Provedores Recursos Computacionais Adequabilidade (Pesos)	Custos Financeiros Atraso Interdomínio Recursos Computacionais	Slots de Frequência Custos Operacionais	Custos Financeiros Balanceamento de Carga	N/A	Disponibilidade de Rotas Disponibilidade de Banda Confiabilidade de Domínio Capacidade de Suporte a Funções de Rede	Eficiência de Banda Eficiência de Processamento
Suporte a Personalização de Métricas de Avaliação	✗	✗	✗	✗	✗	✗	✗
Suporte a Definição de Dependências de Domínio	✗	✓	✓	✗	✗	✗	✗

A Tabela 3.3 sumariza as principais características das soluções apresentadas. Apesar destas soluções trabalharem com múltiplos objetivos de otimização, elas não permitem aos seus usuários configurar o conjunto de métricas que devem ser avaliadas. Essa ausência de suporte a customização acarreta dificuldades para modelar e avaliar políticas e acordos que são diretamente ligados à tarefa de implantação (*e.g.*, atraso máximo, distancia geográfica máxima). Além disso, limitações quanto ao processamento de restrições são notadas nas soluções (Riera et al., 2016) e (Wang et al., 2017). Nelas não é possível definir dependências de domínio específicas de algumas

funções de rede, o que torna inviável, por exemplo, a utilização das mesmas para distribuir serviços híbridos onde há funções implementadas fisicamente em um nodo da rede coexistem com funções virtualizadas.

Finalmente, a solução GA+LCB (Li et al., 2020) fornece suporte limitado para a definição de dependências de domínio – que podem ser especificadas apenas para a primeira e a última funções de rede de uma cadeia. Além de empregar um algoritmo genético mono-objetivo, essa solução não permite aos usuários ajustar características genéticas típicas, como definir a probabilidade de cruzamento e escolher o algoritmo seletor. Semelhante ao GA+LCB, a solução proposta em (Rodis e Papadimitriou, 2021) emprega um algoritmo genético mono-objetivo que permite aos usuários apenas ajustar características genéticas, como taxas de cruzamento e mutação. No entanto, este algoritmo não permite que o usuário altere sua função objetivo nem defina dependências de domínio de determinadas funções de rede em um serviço. É possível afirmar que nenhuma das soluções apresentadas suporta uma configuração de avaliação customizada ou é baseada em algoritmos genéticos multiobjetivos para mapear serviços em múltiplos domínios. No presente trabalho, argumentamos que algoritmos genéticos multiobjetivos podem suportar configurações de avaliação personalizáveis enquanto encontram bons mapeamentos de candidatos em tempos viáveis para problemas complexos de mapeamento de serviços de rede virtual.

Também são destacadas soluções de implantação que, em outros contextos, utilizam heurísticas genéticas para resolver problemas de otimização. Em (Cao et al., 2016), (Carpio et al., 2017), (Khebbache et al., 2018) e (Tavakoli-Someh et al., 2019) são apresentadas soluções baseadas em algoritmos genéticos para executar a tarefa de alocação de servidores durante a integração de um serviço de rede virtualizado. Em especial, a solução em (Cao et al., 2016) utiliza soluções baseadas em *Multi-Objective Genetic Algorithm* (MOGA) e NSGAII para otimizar a seleção de servidores considerando dois objetivos: redução da sobrecarga das conexões entre servidores e balanceamento de carga entre servidores. Já na solução proposta em (Carpio et al., 2017), uma heurística genética é construída visando a otimização da seleção de servidores em múltiplos centros de processamento. Essa solução tem por objetivo minimizar a sobrecarga das conexões entre os servidores escolhidos. Similarmente, em (Khebbache et al., 2018), o NSGAII é utilizado para minimizar dois objetivos: o número de servidores utilizados e a sobrecarga das conexões entre eles. Finalmente, em (Tavakoli-Someh et al., 2019), uma solução é proposta também sobre NSGAII visando a maximização do uso médio dos servidores disponíveis e, ao mesmo tempo, a minimização do número de servidores necessários para a integração do serviço. Em um segundo contexto, de programação da execução de funções de rede virtualizadas, uma solução baseada em NSGAII é proposta visando otimizar o uso de recursos computacionais de servidores hospedeiros e o tempo de conclusão do processamento das funções de rede neles hospedadas (Ma et al., 2017). Apesar de utilizarem heurísticas genéticas, nenhuma das soluções apresentadas é destinada ao mapeamento multidomínio de serviços de rede, além de não serem capazes de otimizar métricas de interesse customizadas pelo usuário.

3.4 SUMARIZAÇÃO E DISCUSSÃO

Recentemente diferentes plataformas, soluções e sistemas foram propostos para implementar, na prática, o paradigma NFV. Entre essas implementações estão aquelas desenvolvidas para instrumentalizar os elementos operacionais pertencentes ao domínio de trabalho de VNF, ou seja, as próprias instâncias de VNF e os EMS correspondentes. Para garantir o correto funcionamento desses elementos operacionais, assim como o atendimento dos mais diversos fundamentos relacionados ao paradigma NFV, uma série requisitos devem ser satisfeitos, como o processamento de protocolos específicos, provimento de interfaces de comunicação padroni-

zadas e flexibilidade para adoção de tecnologias de virtualização variadas. Atualmente, tanto plataformas de execução de VNF quanto soluções de EMS são implementadas livremente, sem observar de forma integral o rigor técnico detalhado em documentos de referência fornecidos por entidades de padronização de NFV, como ETSI e IETF. Apesar dessas implementações suportarem a execução de ambientes virtualizados de funções de rede, o cenário descrito resulta em *softwares* pouco interoperáveis e até mesmo imprevisíveis quanto às suas funcionalidades, criando barreiras tecnológicas para a evolução do paradigma na totalidade.

Em particular, para as plataformas de execução de VNF, duas funcionalidades com grande potencial de exploração para a composição de funções e serviços de rede complexos são desconsideradas: NSH e VNFC. A ausência de suporte ao processamento nativo de NSH impõe a utilização de funções de rede extras para o desencapsulamento/encapsulamento do cabeçalho de serviço de rede para pacotes que atravessam cadeias de função de serviço (arquitetura SFC). Essa imposição pode ocasionar um maior atraso de processamento e degradação da vazão do serviço (Vu et al., 2016). Já a ausência de suporte a VNFC estabelece que as funções de rede executadas nessas plataformas sejam construídas de forma monolítica, retardando os seus processos de desenvolvimento e minimizando as oportunidades de reuso de código. Em relação às características que estão presentes nas plataformas de execução de VNF apresentadas, além de suas disparidades arquiteturais, destaca-se a fixação do uso de uma ferramenta exclusiva de acesso ao tráfego de rede (*e.g.*, aceleradores de pacotes, como DPDK e netmap, ou soluções tradicionais como *sockets* de camada dois). Essa predeterminação de uma única ferramenta para acesso à rede pode restringir a implantação de tais plataformas conforme o ambiente de virtualização disponível. Por exemplo, *sockets* de camada dois tradicionais podem ser insuficientes para o processamento de grandes quantidades de tráfego, enquanto aceleradores de pacotes com uso intensivo de CPU, como o DPDK, podem ser incompatíveis com ambientes de virtualização limitados em recursos computacionais.

Quanto às soluções de EMS, existem poucas opções de implementações que estão disponíveis para uso prático, sejam elas provenientes da indústria ou da academia. O baixo investimento em soluções de EMS é consequência do foco dado ao desenvolvimento de sistemas de NFV-MANO que, em teoria, conseguiriam satisfazer por completo as necessidades de gerenciamento dos ambientes de rede NFV. Porém, apesar dos esforços realizados para o lançamento de um NFV-MANO autossuficiente no gerenciamento de todos os domínios de trabalho da arquitetura NFV, esse sistema completo ainda não foi alcançado. Em geral, os sistemas NFV-MANO disponíveis apresentam problemas de compatibilidade de comunicação com várias plataformas de execução de VNF, suportando o gerenciamento do ciclo de vida interno apenas de um subconjunto pequeno das mesmas. Para contornar esse obstáculo, as soluções de EMS apresentadas na Seção 3.2 deste capítulo foram propostas. Entretanto, devido ao cenário descrito, os EMS disponíveis são fortemente vinculados aos sistemas NFV-MANO, herdando suas operações e limitações, o que resulta em soluções dependentes e pouco portáveis. Além disso, para suprir superficialmente necessidades de configuração de uma instância de VNF, alguns sistemas NFV-MANO optam por fornecer apenas rotinas específicas de EMS, provendo uma solução parcial ao desafio do gerenciamento interno de instâncias de VNF. Finalmente, por se tratarem de implementações simples e restritas, a maior parte das soluções de EMS não apresentam uma organização arquitetural com módulos operacionais bem definidos, sendo disponibilizadas como um *software* indivisível, pouco personalizável e com baixa ou nenhuma portabilidade entre sistemas NFV-MANO diferentes.

Por fim, em relação às soluções de mapeamento multidomínio apresentadas, todas manifestam profundas limitações relacionadas a flexibilidade de configuração e avaliação de funções objetivo personalizadas. Ou seja, as necessidades dos operadores e gerentes de redes

devem ser adaptadas às capacidades das soluções, e não o contrário. Esse comportamento limita as possibilidades e cenários de aplicação destas soluções, sendo pouco práticas para operarem em ambientes genéricos como aqueles providos por plataformas de *NFV-as-a-Service* (Bondan et al., 2019). Também, limitações quanto a definição de restrições de mapeamento de determinadas funções pertencentes a um serviço inviabilizam a utilização das soluções em cenários híbridos de rede, onde funções implementadas em *hardware* dedicado coexistem com aquelas virtualizadas. Sem a capacidade de fixar uma função em determinado domínio (não modificando a sua posição durante o mapeamento), os resultados de otimização podem requisitar migrações físicas de funções implementas em *hardware*, o que pode ser inviável ou indesejável para as partes interessadas.

4 ARQUITETURA DE PLATAFORMAS DE EXECUÇÃO DE VNF

Este capítulo aborda as características desejáveis e os módulos necessários para a padronização do elemento operacional VNF. Primeiramente, na Seção 4.1, a arquitetura para plataformas de execução de VNF é apresentada, tendo seu escopo de atuação delimitado e seus módulos internos descritos e discutidos. A Seção 4.2 demonstra a aplicabilidade prática da arquitetura genérica proposta através da implementação de uma plataforma de execução de VNF compatível com a mesma. A plataforma implementada foi testada em diferentes estudos de caso, cujas descrições e resultados são relatados e examinados criticamente na Seção 4.3. Finalmente, a Seção 4.4 sumariza e discute de forma abrangente o capítulo.

4.1 DEFINIÇÕES ARQUITETURAIS E OPERACIONAIS

A padronização do elemento operacional VNF é uma necessidade indispensável para satisfazer os múltiplos fundamentos do paradigma NFV (*i.e.*, portabilidade, desempenho, integração, gerenciamento e escalabilidade). Uma das formas de alcançar tal objetivo é através da definição arquitetural desse elemento. Essa definição, essencialmente, serve para orientar o desenvolvimento de plataformas de execução de VNF, conduzindo a soluções que executam funções de rede independentemente da heterogeneidade de *software* que pode constituir o domínio VNF. Além disso, essas implementações devem ser naturalmente cooperativas com outros elementos operacionais da arquitetura NFV, adotando protocolos capazes de abstrair as suas diferentes estratégias de execução. Dentro desse contexto, visto a crescente exploração acadêmica/industrial e o caráter de aplicação prática ainda experimental do paradigma de virtualização de funções de rede, também é importante estabelecer diretivas flexíveis, capazes de suportar expansões, remoções e adaptações de funcionalidades ao longo do tempo. Todas as considerações realizadas foram observadas durante a definição arquitetural do elemento VNF (em específico, das plataformas de execução de VNF) proposta como parte deste trabalho, resultando em uma arquitetura genérica, interoperável e altamente personalizável no que diz respeito a sua implementação.

A arquitetura proposta não inclui em seu escopo diretivas para o desenvolvimento das funções de rede (*i.e.*, NF) que executam nas plataformas de execução de VNF, sendo estas últimas os objetos de estudo. Dessa forma, as funções de rede podem ser projetadas em diferentes linguagens de programação que podem ou não serem suportadas por uma plataforma de execução de VNF em particular. Evidencia-se que o ferramental de suporte à execução de funções de rede é definido em nível de projeto de implementação dessas plataformas, e não em nível arquitetural. Entretanto, é importante ressaltar que, uma vez projetada para suportar uma determinada linguagem de programação, por exemplo, a plataforma deve prover um conjunto de capacidades operacionais padronizadas, estas então definidas em nível arquitetural. Assim, uma função de rede programada com uma ferramenta específica suportada por múltiplas plataformas de execução de VNF deve ser executada nas mesmas de maneira natural, requerendo nenhuma modificação em seu código nuclear (*i.e.*, voltado ao processamento de tráfego) e o mínimo de modificações possível em seu modelo de apresentação para implantação nessas plataformas (*e.g.*, *flags*, metadados e cabeçalhos de funções de *callback*).

A arquitetura de plataforma de execução de VNF proposta neste trabalho, publicada em (Fulber-Garcia et al., 2019b) e (Fulber-Garcia et al., 2019a) e ilustrada na Figura 4.1, é constituída de seis módulos internos, todos executados em um sistema operacional hospedeiro (*VNF Core*):

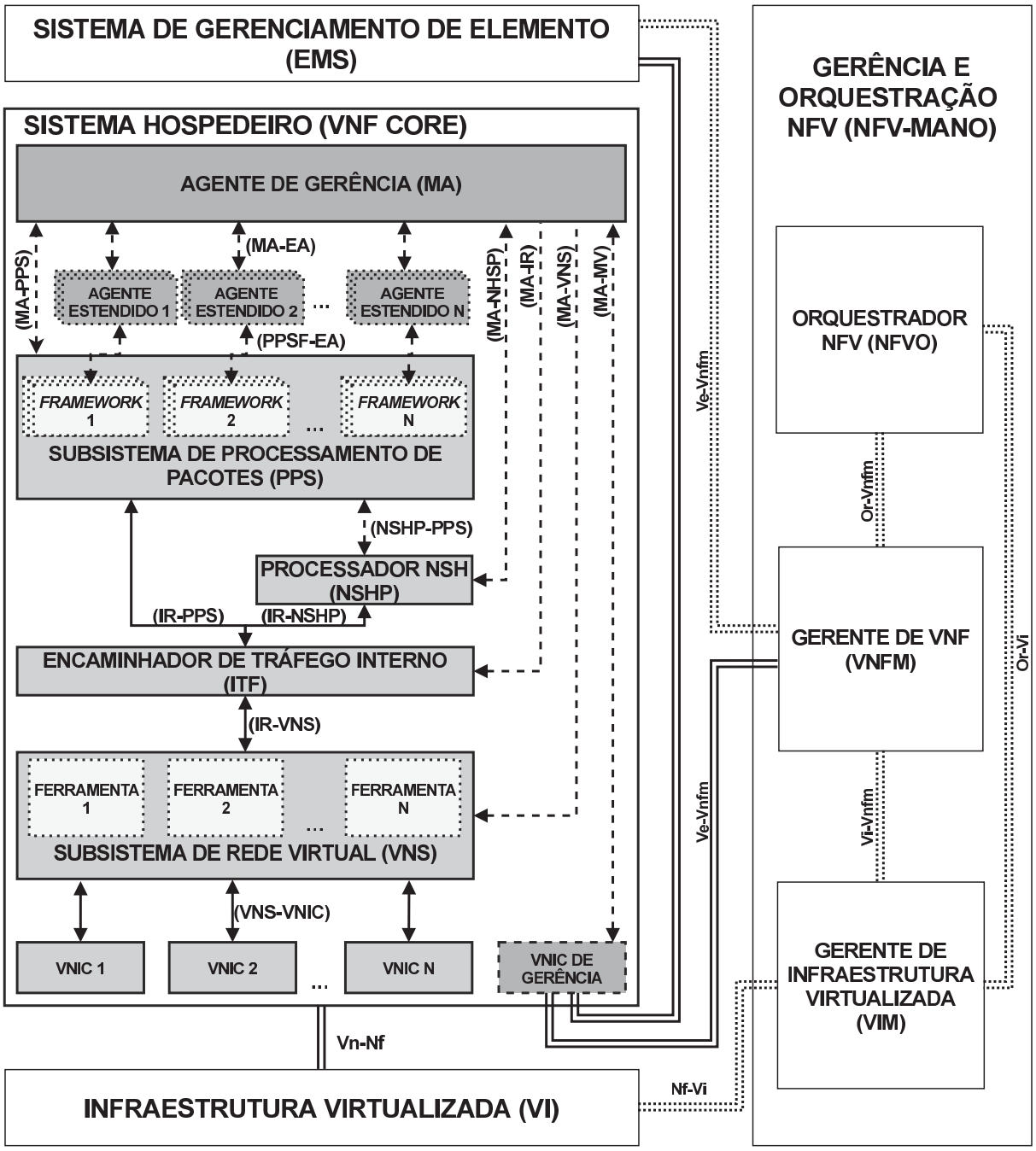


Figura 4.1: Arquitetura de Plataformas de Execução de VNF

(i) Subsistema de Rede Virtual (*Virtual Network Subsystem* - VNS); (ii) Encaminhador de Tráfego Interno (*Internal Traffic Forwarder* - ITF); (iii) Processador NSH (NSH *Processor* - NSHP); (iv) Subsistema de Processamento de Pacotes (*Packet Processing Susbsystem* - PPS); (v) Agente de Gerenciamento (*Management Agent* - MA); e Agente Estendido (*Extended Agent* - EA). Cada um desses módulos executa operações específicas, processando pacotes relacionados ao plano de dados/controle (linhas sólidas) e ao plano de gerência (linhas tracejadas). Interfaces externas ao sistema hospedeiro também são consideradas na arquitetura, permitindo a gestão dos seus recursos computacionais virtualizados (disponibilizados pelo VI) e suportando a execução de operações de gerência e orquestração (através do EMS e VNFM). Os módulos foram projetados para serem independentes entre si, comunicando-se através de interfaces de acesso bem definidas.

Dessa forma, um módulo em particular pode ser reimplementado e substituído de acordo com a evolução das tecnologias relacionadas a ele, sendo esse processo invisível aos demais módulos de uma plataforma. O detalhamento de cada módulo é apresentado a seguir:

- **Subsistema de Rede Virtual** (VNS) – Módulo cuja principal responsabilidade é acessar os Controladores de Interfaces de Rede Virtual (*Virtual Network Interface Controller* - VNIC) providos pelo hipervisor, gerenciando assim o recebimento e envio de pacotes. Observa-se que essas operações de entrada e saída de dados, quando executadas através de uma pilha de rede nativa de sistemas operacionais tradicionais, não são adequadas para atender os requisitos de desempenho relacionados à vazão de redes de alta velocidade (*e.g.*, 40GbE/100GbE). Para suportar esses requisitos, diversas ferramentas de aceleração de pacotes (*e.g.*, netmap (Rizzo, 2012), PacketShader (Han et al., 2010), Intel DPDK (DPDK, 2021), PF_RING/DNA (ntop pf_ring, 2021) e OpenOnload (OpenOnload, 2021)) têm sido propostas e extensivamente avaliadas. Essas ferramentas podem substituir os tradicionais *sockets* de camada dois nas tarefas relacionadas ao encaminhamento de tráfego, sendo soluções adequadas para aplicação em redes baseadas no paradigma NFV. De forma genérica, tanto recursos tradicionais (*e.g.*, *sockets*) quanto aceleradores de pacotes são tratados como ferramentas do VNS. Esse módulo deve disponibilizar ao menos uma ferramenta de acesso à rede, mas pode estender a oferta para múltiplas ferramentas, gerenciando as mesmas e recebendo/enviando os pacotes de forma abstraída aos demais módulos da arquitetura. Em último caso, a escolha pela utilização de uma ferramenta de rede é entendida como um parâmetro interno das plataformas de execução de VNF, sendo este definido pelos operadores de rede.

- **Encaminhador de Tráfego Interno** (ITF) – Uma vez que o tráfego de rede é capturado pelo VNS, este é encaminhado internamente para o ITF. O módulo de ITF é configurado e reconfigurado pelo agente de gerência no momento da implantação de uma nova função de rede ou de um novo componente de VNF (*i.e.*, VNFC). Esse módulo torna-se operacional no momento da inicialização do processamento de tráfego na plataforma. O ITF é responsável por manter a ordem de encaminhamento dos pacotes entre os componentes de VNF que estão sendo executados no PPS. Além disso, quando necessário, ele também coordena o envio de pacotes ao NSHP para remover ou reinserir o cabeçalho de serviço de rede. Durante a configuração do ITF, os canais de comunicação acordados entre os diferentes módulos da arquitetura são estabelecidos, sendo que estes podem assumir a forma de, por exemplo, regiões de memória compartilhada, *pipes* interprocessos ou *sockets* (a opção por uma tecnologia ou outra é feita em nível de projeto de implementação de uma plataforma). Esses canais de comunicação são os meios que viabilizam o trânsito de pacotes entre VNS, PPS e NSHP. É importante ressaltar que a presença do ITF flexibiliza tanto a implementação de componentes de VNF, quanto a composição dos mesmos em funções de rede completas. Isso acontece tendo em vista que o ITF consiste em um ponto único de comunicação entre um VNFC e a plataforma, removendo a necessidade de compatibilidade direta entre componentes heterogêneos e tornando as relações de encaminhamento de tráfego independentes e individualizadas.

- **Processador NSH** (NSHP) – O cabeçalho de função de serviço (NSH), especificado pela IETF, encapsula pacotes (*i.e.*, camada três) permitindo o direcionamento dos mesmos em caminhos de cadeias de função de serviço. Entretanto, apesar de suas múltiplas vantagens no contexto de NFV, o uso do NSH é opcional, podendo ser

substituído por outros mecanismos e protocolos, como SDN/OpenFlow. O NSHP, um módulo com ativação opcional, foi incluído na arquitetura para tratar exclusivamente dos desafios relacionados ao NSH. Esse módulo objetiva (i) desonerar os desenvolvedores de realizarem modificação no código-fonte de funções de rede nativamente agnósticas ao NSH, permitindo assim que executem em um ambiente onde este protocolo é usado; e (ii) evitar a necessidade de instanciação de funções de rede extras para atuarem como *proxies* NSH. O NSHP, essencialmente, abstrai a existência do cabeçalho de serviço, o removendo, atualizando e reinserindo, respectivamente, antes, durante e após o processamento de um pacote por uma função de rede. Além disso, para quando funções agnósticas ao NSH são utilizadas, esse módulo fornece um conjunto de operações de alto nível para manipulação dos campos editáveis do cabeçalho. Especificamente, quando ativado, todo o tráfego de chegada é transmitido do ITF para o NSHP antes de ser encaminhado para o PPS, enquanto todo tráfego de saída é encaminhado do ITF ao NSHP após o retorno do PPS. De maneira geral, o NSHP atua em campos específicos do cabeçalho de serviço, os quais podem ser modificados por instâncias de VNF: o Índice de Serviço (*Service Index* - SI) e o Cabeçalho de Contexto (*Context Header* - CH). O SI determina a posição do pacote na cadeia de serviço e é normalmente modificado de forma automática pelo NSHP, já o CH carrega metadados do pacote durante a sua passagem por uma cadeia de serviço, sendo alterado pelo NSHP sob demanda das funções de rede. Sendo assim, o conjunto mínimo de operações do módulo NSHP inclui "remoção NSH" (automática), "reinserção NSH" (automática), "recuperação CH" (sob demanda da NF) e "atualização CH" (sob demanda da NF).

- **Subsistema de Processamento de Pacotes** (PPS) - O módulo de PPS corresponde aos *frameworks* e a lógica de processamento de pacotes empregados na implementação e execução de funções de rede virtualizadas na arquitetura. Comumente, o conjunto de *frameworks* de desenvolvimento pode incluir aplicações de alto nível, como *Click Modular Router* (Kohler et al., 2000) e *Vector Packet Processing* (The Linux Foundation, 2021), linguagens de programação completas, a exemplo de C (Gurevich e Huggins, 1992), C++ (Stroustrup, 1999) e Python (Sanner et al., 1999), e bibliotecas independentes para processamento de pacotes que se acoplam às linguagens de programação, como Scapy (SecDev, 2021), libtins (Fontanini, 2021) e libnet (The libnet Developer Community, 2021). Ainda, *frameworks* legados que são utilizados apenas por plataformas de execução de VNF específicas, como NetLib (NetVM (Hwang et al., 2015)) e NFLib (OpenNetVM (Zhang et al., 2016b)), podem ser adaptados no contexto do PPS (sintaticamente e semanticamente) para prover compatibilidade interplataforma na execução de funções de redes. Além de indicar o conjunto de *frameworks* de desenvolvimento disponível, o PPS também é responsável pela criação, inicialização e manutenção dos componentes que executam as rotinas de processamento do tráfego de rede, assim como de seus Agentes Estendidos (discutidos a seguir). Esses componentes e agentes podem ser estabelecidos desde como simples *threads* ou processos subordinados diretamente ao sistema operacional da plataforma de execução de VNF, até como contêineres ou máquinas virtuais (através de virtualização aninhada), apresentando maior grau de independência em relação ao mesmo.

- **Agente de Gerência** (MA) - O objetivo primário do Agente de Gerência é configurar, controlar e monitorar a execução das funções de rede. Além disso, ele também é responsável por coordenar a atividade dos demais módulos internos de uma plataforma de execução de VNF. Essencialmente, o MA deve prover cinco operações de alto nível:

requisitar; iniciar; parar; recuperar; e monitorar. A operação "requisitar" recebe um pacote de VNF (*i.e.*, VNFP) e procede com a instalação do mesmo na plataforma, então configurando todos os seus módulos internos. A operação "iniciar" ativa todos os módulos previamente configurados, iniciando o processamento de tráfego de rede. A operação "parar", ao contrário da "iniciar", suspende o processamento do tráfego de rede ao desativar os módulos da plataforma. Uma vez que os módulos estejam configurados, a operação "recuperar" destina-se à obtenção de informações relacionadas à função de rede implantada. Exemplos dessas informações são o identificador e os metadados de criação e distribuição da função. Finalmente, a operação "monitorar" trata da amostragem de métricas de desempenho relacionadas à execução de uma função de rede na plataforma. Esses indicadores são tipicamente obtidos através de consultas aos Agentes Estendidos (métricas particulares da função de rede) ou ao sistema operacional hospedeiro (consumo de CPU, memória, rede, entre outros). Além das operações imprescindíveis, outras podem ser providas pelas plataformas de execução de VNF considerando suas características e capacidades particulares.

- **Agente Estendido** (EA) - Esse módulo é iniciado pelo PPS e posteriormente manipulado pelo MA para monitorar e configurar, em tempo de execução, a função de rede sendo executada, assim como seus componentes em particular. De maneira geral, é esperado que o agente estendido esteja acoplado ao código da função de rede ou seja fornecido como um *script* incluído no VNFP, visto que este módulo atua como um agente de monitoramento dedicado a uma função de rede específica. Exemplos de operações que podem ser disponibilizadas pelos EA são: inclusão de regras em um *firewall*, verificação de quantidade de pacotes descartados por uma regra de *firewall*, definição de rotas em um roteador, entre outras. Este módulo deve prover, essencialmente, uma operação padrão chamada "listar". Essa operação é usada pelo MA para descobrir todas as operações e indicadores que podem ser acessados pelos operadores de redes e demais elementos de gerência da arquitetura NFV. De forma abrangente, o módulo de EA é o primeiro esforço concreto em direção da disponibilização de operações de gerência personalizadas para VNFC/NF em NFV.

Por norma, todos os módulos da arquitetura proposta estão sob controle do MA. Inicialmente, na ocorrência de uma operação de "requisitar", o MA recebe um VNFP através da interface MA-MV, realiza a validação do mesmo, extrai as informações relevantes para a instalação da função de rede e configura os módulos internos de acordo com elas. Um exemplo fundamental das informações contidas no VNFP consiste do conjunto de componentes de VNF e seus ordenamentos no processamento de tráfego de rede, compondo juntos uma função de rede completa. Esse é o material básico utilizado para configurar apropriadamente o encaminhamento de tráfego realizado pelo ITF. O pacote de instalação também deve prover informações específicas da implementação da função de rede inclusa no mesmo, como aquelas relativas aos agentes estendidos que são instanciados em conjunto aos seus respectivos componentes.

Além de informações de configuração em geral, o MA também recupera do VNFP todos os arquivos de código-fonte da função de rede que devem ser executados pelo PPS. Com isso, o MA requisita ao PPS a instalação da função de rede e a criação dos canais de comunicação associados a ela, além da disponibilização das interfaces de gerência dos componentes que contenham agentes estendidos (PPSF-EA e MA-EA). Após a conclusão dessas tarefas, o PPS retorna uma confirmação de sucesso ou falha para o MA (MA-PPS). No caso de uma falha de instalação, um mecanismo de *rollback* pode ser utilizado para abortar o processo apropriadamente, retornando os módulos para um estado pré-operação "requisitar". Por outro lado, no recebimento

de uma mensagem de sucesso, o processo de configuração da plataforma progride. Nesse caso, o MA dispara a inicialização do módulo de NSHP (MA-NSHP), quando requisitado. O NSHP, então, cria o canal de comunicação com o PPS (NSHP-PPS), permitindo que a função de rede acesse o cabeçalho de contexto do NSH em tempo de execução. Após, a configuração do VNS é iniciada conforme as ferramentas de comunicação requisitadas no VNFP (MA-VNS), sendo estas conectadas às VNICs (VNS-VNIC) para habilitar a entrada e saída de pacotes na plataforma de execução de VNF.

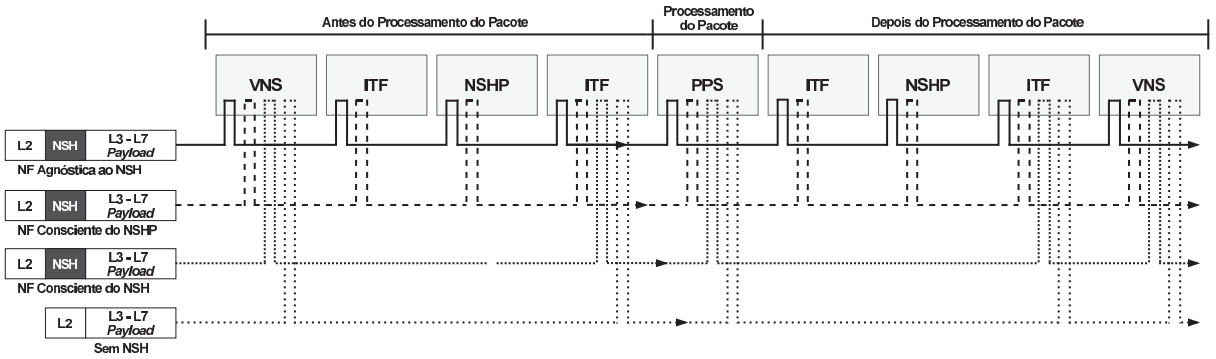


Figura 4.2: Modelos de Tráfego de Rede e Funções de Rede Suportados pela Arquitetura

Finalmente, o ITF é configurado com, no mínimo, dois canais de comunicação: (i) ITF-VNS e (ii) ITF-PPS. O primeiro é destinado ao recebimento e envio de tráfego de rede entre VNS e ITF, enquanto o segundo interconecta o ITF e o PPS com a mesma finalidade. Um terceiro canal, chamado de ITF-NSHP, é estabelecido quando o módulo NSHP é requisitado. Nesse canal o tráfego de rede transita entre o ITF e o NSHP, sendo o NSH removido dos pacotes antes de estes serem encaminhados ao PPS e, posteriormente, reinserido seguido ao término do processamento dos mesmos pelo PPS. A disponibilidade do módulo NSHP permite que arquitetura execute em quatro modelos diferentes segundo as características da função de rede e do tráfego recebido: (i) tráfego com NSH com função de rede agnóstica ao NSH; (ii) tráfego NSH com função de rede consciente do NSHP (instrumentalizada com a biblioteca do NSHP); (iii) tráfego NSH com função de rede consciente do NSH; e (iv) tráfego sem NSH. A Figura 4.2 ilustra o caminho interno percorrido pelo tráfego de rede em cada um dos cenários apresentados. Após o término dos processos de implantação da função de rede e configuração de módulos operacionais, disparado pela operação "requisitar", a plataforma de execução de VNF deve estar preparada para processar o tráfego de rede, mas ainda não operacional. Em termos de implementação, isso representa que os módulos da arquitetura estão instanciados em memória (como estruturas ou objetos, por exemplo), mas que suas rotinas de processamento (*i.e.*, processos de servidor ou *looping* principal) ainda não foram inicializados. Já em termos abstratos, a plataforma de execução de VNF encontra-se em um estado ocioso, aguardando por uma operação de "iniciar". O resumo do processo de configuração descrito é feito no diagrama de sequência na Figura 4.3.

A Tabela 4.1 exibe os estados da plataforma e suas transições possíveis considerando o conjunto de operações essenciais. Quando uma plataforma de execução de VNF está em estado **neutro**, apenas o módulo de MA encontra-se operacional e respondendo exclusivamente à operação de "requisitar". Qualquer outra operação recebida durante esse estado (considerando aquelas operações essenciais) será sumariamente rejeitada sem que modificações sejam realizadas na plataforma. Quando uma operação "requisitar" que provenha um VNFP válido é recebida e processada, o estado da plataforma é transicionado para ocioso. Quando em estado **ocioso**, a plataforma encontra-se com seus módulos operacionais configurados e uma função de rede instalada no PPS. Entretanto, esses módulos ainda não estão ativos, portanto, nenhum tráfego de

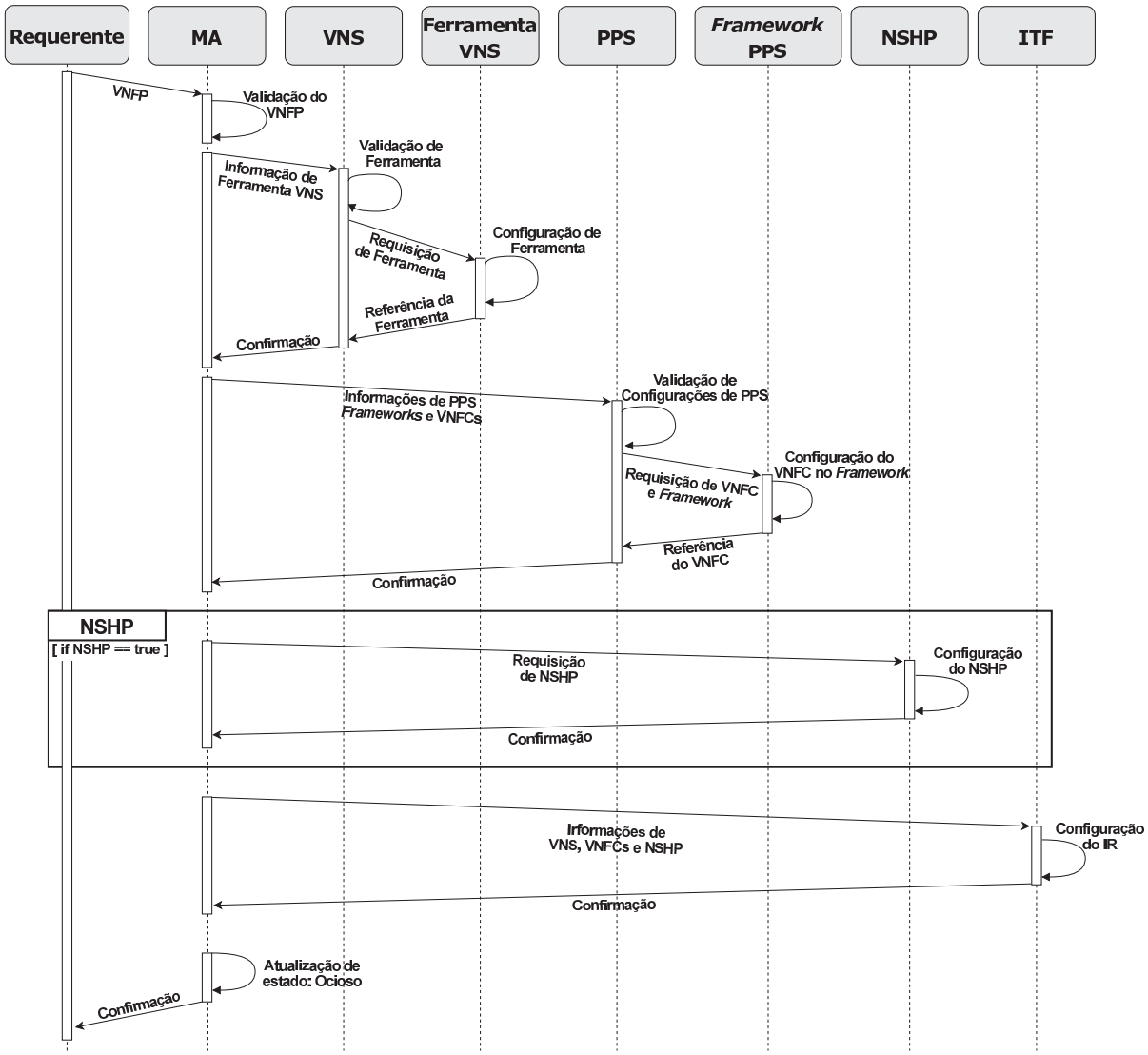


Figura 4.3: Processo de Configuração de Plataformas de Execução de VNF

rede está sendo processado. Nesse estado, três operações essenciais são permitidas: “requisitar”, “recuperar” e “iniciar”, descritas a seguir. A operação “requisitar” modifica a função implantada e mantém a plataforma no estado ocioso. A operação “recuperar” envia informações relativas ao VNFP e, em especial, da função de rede implantada que configura a plataforma no momento da requisição, essa operação não resulta em alterações de estado. A operação “iniciar”, em caso de sucesso, ativa as rotinas de processamento dos módulos configurados e habilita o processamento de tráfego de rede pela função implantada, transicionando a plataforma para o estado operacional. Falhas na operação “iniciar” podem ocorrer devido a incompatibilidades entre a plataforma e o sistema operacional hospedeiro/elemento virtual utilizado, como falta de privilégios para criação de processos e manipulação de aceleradores de pacotes ou uso de *drivers* incompatíveis nas interfaces de rede disponíveis. Nesse caso, um sinal de erro é emitido e a plataforma é mantida no estado ocioso. No estado **operacional**, três operações essenciais são permitidas: “recuperar”, “monitorar” e “parar”. A operação “recuperar” acontece da mesma forma que no estado ocioso. A operação “monitorar” acessa as métricas de monitoramento, sejam da plataforma ou da função, não alterando o estado da plataforma. A operação “parar” paralisa a execução dos módulos operacionais (exceto o MA) e cessa o processamento de tráfego de rede da função instalada, transicionando a plataforma para o estado ocioso. As transições descritas

consideram o funcionamento normal de uma plataforma de execução de VNF, assumindo a inexistência de erros de programação ou de escassez de recursos computacionais, por exemplo.

Tabela 4.1: Transições de Estados de Plataformas de Execução de VNF

	"requisitar"	**"iniciar"**	**"parar"**
Neutro	*Ocioso (sucesso)* *Neutro (falha)*	*Neutro (falha)*	*Neutro (falha)*
Ocioso	*Ocioso (sucesso)* *Ocioso (falha)*	*Operacional (sucesso)* *Ocioso (falha)*	*Ocioso (falha)*
Operacional	*Operacional (falha)*	*Operacional (falha)*	*Ocioso (sucesso)*

A arquitetura de plataformas de execução de VNF detalhada nesta seção concretiza meios de execução de funcionalidades do estado da arte do paradigma NFV, como processamento NSH e fragmentação de funções de rede em componentes. Tais funcionalidades, atualmente, não são tratadas em nenhum nível por outras plataformas de execução de VNF existentes, sendo consideradas desafios independentes a elas. Quando abordados, esses desafios são compreendidos como parte do desenvolvimento de funções de rede, sendo o NSH tratado no próprio código-fonte desta ou através de sistemas terceirizados (*proxies*), e uma função fragmentada em componentes referida como um serviço de rede leve orquestrado pelo NFV-MANO. A arquitetura proposta apresenta uma nova perspectiva para suportar essas funcionalidades de maneira prática e abrangente, evitando mudanças profundas e disruptivas no domínio de trabalho NFV-MANO ao estabelecer padrões e competências para um elemento operacional (*i.e.*, VNF) até então pouco explorado. Assim, além da operacionalização das funcionalidades citadas, também foi possível alargar o escopo de atuação das plataformas de execução de VNF, flexibilizando a utilização de ferramentas de rede (como aceleradores de pacotes) e *frameworks* de desenvolvimento (como linguagens de programação). Finalmente, a arquitetura proposta organiza-se através de módulos operacionais fracamente acoplados, facilitando a implementação e atualização de plataformas baseadas na mesma, além de prover maior interoperabilidade entre plataformas de desenvolvedores diferentes devido à padronização de seu funcionamento interno.

4.2 IMPLEMENTAÇÃO DO PROTÓTIPO COVEN

Como prova de conceito da arquitetura proposta, uma nova plataforma de execução de VNF, chamada *COmprehensive VirtualizEd Nf* (COVEN)[1], foi implementada seguindo as diretrizes apresentadas na Seção 4.1. A plataforma COVEN emprega o sistema operacional Ubuntu Cloud como seu sistema hospedeiro. O Ubuntu Cloud é um sistema operacional genérico desenvolvido para a execução de uma variedade de tarefas. Dessa forma, esse sistema permite a execução do protótipo e a disponibilização de um grande conjunto de *frameworks* de desenvolvimento e ferramentas de rede virtual sem enfrentar problemas de compatibilidade de *software*. Entretanto, para plataformas em nível de produção, a utilização de virtualização leve deve ser observada, priorizando a adoção de contêineres ou máquinas virtuais com sistemas operacionais hospedeiros simplificados, como OSv (Kivity et al., 2014), CoreOS (Red Hat, 2021), Alpine (The Alpine Project, 2021) e TinyCore (The Tiny Core Project, 2021), que satisfazem apropriadamente os requisitos de virtualização NFV (*e.g.*, desempenho, portabilidade e integração). Os módulos internos da plataforma foram desenvolvidos utilizando a linguagem de programação Python 3.

[1]Disponível em `https://github.com/ViniGarcia/COVEN`

O Subsistema de Rede Virtual (VNS) foi implementado de forma genérica, mas provendo exclusivamente a ferramenta de *sockets* de camada dois para a realização do acesso ao tráfego de rede. Apesar disso, aceleradores de pacotes podem ser incluídos como opções na plataforma COVEN através de sua instalação no sistema hospedeiro e, posteriormente, inserção no VNS através do modelo padrão para programação de ferramentas de rede virtual, provido pela mesma. O Encaminhador de Tráfego Interno (ITF) comunica-se com módulos operacionais de VNS e NSHP utilizando regiões de memória compartilhada. Já a comunicação com o PPS (*i.e.* funções de rede ou componentes) foi estabelecida com *sockets* de camada três. Apesar da adoção de *sockets* de camada três prejudicar o desempenho da plataforma em relação à latência e vazão quando comparada ao uso de memória compartilhada, esta tecnologia foi empregada uma vez que apresenta compatibilidade nativa com um enorme número de *frameworks* de desenvolvimento, flexibilizando a implementação de funções de rede, componentes de VNF e agentes estendidos.

O processador NSH atua como um *proxy* interno e nativo que é executado sob demanda, segundo a requisição do operador de rede ou sistema de gerência que atua sobre uma instância do COVEN. O NSHP recebe tráfego de rede do ITF através de uma região de memória compartilhada, mutuamente controlada entre esses módulos. Assim, o NSH de cada pacote é identificado em seu respectivo fluxo e alocado em um dicionário. Cada fluxo apresenta diversos campos comuns de NSH, os quais são mantidos de maneira comunitária pelo NSHP. Por outro lado, informações que podem ser particulares de cada pacote, como o TTL e o Cabeçalho de Contexto (CH), são individualmente armazenadas durante o processo de remoção do NSH. Cada pacote é identificado com um ID único inserido no campo *IP Options* do cabeçalho IP. Em seguida, os pacotes são devolvidos ao ITF para serem encaminhados ao PPS. Durante o processamento de um pacote, acessos e modificações ao CH podem ser requisitados por parte da função de rede e de seus componentes. Requisições pela recuperação ou modificação do CH, quando necessárias, devem ser programadas diretamente nas rotinas de processamento de pacotes. Com esse objetivo, bibliotecas são providas pelo NSHP para cada *framework* de desenvolvimento disponibilizado no COVEN. Finalmente, após a conclusão do processamento pelo PPS, o ITF encaminha os pacotes ao NSHP, o qual atualiza os campos dos seus respectivos cabeçalhos NSH (TTL e SI) e os reinsere nos pacotes. Ressalta-se que, em relação ao CH, o NSHP do COVEN considera apenas a utilização do tipo denominado "tamanho-fixo" (Quinn et al., 2018).

O Subsistema de Processamento de Pacotes (PPS) construído conta nativamente com cinco *frameworks* de desenvolvimento, sendo estes representados por uma aplicação de alto nível de abstração (Click Modular Router) e quatro linguagens de programação (C, Python 3, Java e JavaScript). Os *frameworks* disponibilizados são previamente instalados no sistema operacional hospedeiro, reconhecidos e interfaceados pelo COVEN. Funções de rede e seus componentes são inicializados pelo PPS como processos subordinados ao processo geral da plataforma de execução de VNF. Sendo assim, apesar de sua independência de memória e escalonamento no sistema hospedeiro, o PPS mantém privilégios para parar e remover os processos a qualquer momento. O código-fonte das funções de rede passadas ao PPS durante uma operação "requisitar" não é validado no processo de implantação. Portanto, erros incluídos pelos desenvolvedores de funções de rede são percebidos apenas durante a inicialização da função de rede (*i.e.*, operação "iniciar"), resultando em uma falha devido a um erro em tempo de execução.

O Agente de Gerência foi desenvolvido através de uma interface HTTP utilizando a biblioteca Bottle[2]. Além do conjunto essencial de operações para controle de ciclo de vida da função de rede e sua plataforma, o MA disponibiliza também duas operações extras: "check", que realiza a verificação de *heartbeats* da função de rede e seus componentes (usando a aplicação `Netcat` (The GNU Netcat Project, 2021)); e "turn off", que desativa os módulos da plataforma

[2]https://github.com/bottlepy/bottle

de execução de VNF, inclusive o MA, tornando-a indisponível sem haver necessidade de desligar seu sistema operacional hospedeiro. A primeira operação extra (“check”) realiza uma verificação de alto nível que pode embasar o disparo de monitoramentos mais específicos na tentativa de, por exemplo, solucionar de gargalos de processamento presentes em algum ponto da função de rede. Objetivando essa verificação, um sinal de vida é requisitado para cada componente da função de rede e mede-se o tempo entre a requisição deste sinal e o recebimento da resposta do componente, informando esses dados ao requerente da operação. A segunda operação extra (“turn off”) desativa sumariamente os módulos da plataforma, essa operação pode ser utilizada, por exemplo, na eminência de um ataque ao serviço de rede, assim desativando a plataforma, mas permitindo que mudanças sejam feitas rapidamente no sistema hospedeiro (como troca de IP das interfaces), tornando a recuperação ou atualização da plataforma e de sua função de rede uma tarefa tratada de forma exclusivamente interna ao elemento virtualizado (não requisitando, por exemplo, a reinicialização da máquina virtual). Com essas operações, o COVEN satisfaz o conjunto básico de operações para controle de ciclo de vida de VNF (Bondan et al., 2014).

4.2.1 Interconexão dos Módulos Operacionais

As interconexões para troca de dados entre os módulos operacionais da plataforma de execução de VNF são preparadas durante uma operação “requisitar” válida, sendo estabelecidas de fato em uma posterior operação “iniciar”. Na plataforma COVEN não existe uma tecnologia de conexão padrão, sendo esta definida de acordo com as características da comunicação e compatibilidade necessária em cada uma das interfaces.

A comunicação VNS-VNIC é implementada exclusivamente com *sockets* de camada dois, sendo o modelo mais genérico para adaptação em diferentes sistemas hospedeiros. As conexões ITF-VNS, ITF-NSHP e ITS-PPS são estabelecidas utilizando memória compartilhada interprocessos. Essa tecnologia é usada nessas conexões já que elas ocorrem apenas no contexto interno da plataforma, não requisitando que desenvolvedores de funções de rede ou operadores da plataforma necessitem se adequar a um sistema próprio de gerenciamento de memória. Em relação às conexões habilitadas por memória compartilhada, as condições de corrida causadas por ações de escrita em recursos presentes nas mesmas são tratadas com o uso de *mutexes* (Mueller, 1993). A conexão NSHP-PPS consiste em uma interface REST que possibilita a execução e troca de dados entre uma função de rede e o NSHP. Entretanto, essa interface REST em particular é abstraída através de bibliotecas fornecidas pelo próprio NSHP. As conexões relacionadas ao plano de gerência da plataforma (*i.e.*, MA-PPS, MA-NSHP, MA-ITF, MA-VNS e MA-MV) são tratadas internamente por operações entre processos. Por fim, a conexão PPSF-EA é naturalmente inserida no contexto da função de rede ou componente. Uma implementação típica dessa conexão é via compartilhamento de memória entre o processo que executa as rotinas de processamento de pacotes e as *threads* que o monitoram ou o modificam (outras abordagens também podem ser utilizadas conforme as necessidades do desenvolvedor da função de rede). Já o MA-EA é implementado internamente com *sockets* de camada três. Todas as operações de gerência são disponibilizadas externamente à plataforma por uma interface REST ou através de um servidor TCP, ambos enquadrados no MA.

4.3 EXPERIMENTAÇÃO E RESULTADOS

Esta seção detalha e discute dois estudos de caso construídos para a avaliar a funcionalidade dos módulos operacionais da arquitetura para plataforma de VNF apresentados na Seção 4.1. As avaliações consideram a implementação da arquitetura na forma do protótipo COVEN,

como especificado na Seção 4.2. Os estudos de caso, além de validarem o funcionamento global do processamento de pacotes no COVEN, avaliam a capacidade deste em aplicar funcionalidades do paradigma NFV que não são suportadas por plataformas de execução de VNF existentes: processamento NSH (Subseção 4.3.1) e execução de funções de rede fragmentadas em componentes (Subseção 4.3.2). A execução dos experimentos foi realizada em um servidor Intel Core i7-4790K@3.60Ghz com 8GB de memória RAM DDR4 e Debian 8. Os experimentos foram repetidos 30 vezes, alcançando um intervalo de confiança dos resultados de 95%.

4.3.1 Estudo de Caso: Mitigação de Ataques DDoS DeMONS

O primeiro estudo de caso utiliza NSH com o objetivo promover aprimoramentos de desempenho na mitigação de Ataques Distribuídos de Negação de Serviço (*Distributed Denial of Service* - DDoS) executada pelo serviço de rede DeMONS (DDoS *MitigatiOn* NFV *Solution*) (Fulber-Garcia et al., 2018), que é baseado em NFV. No DeMONS, múltiplas funções de rede virtualizadas são utilizadas para classificar e tratar fluxos maliciosos, suspeitos e benignos. Os fluxos são encaminhados através de canais de alta ou baixa prioridade de acordo com as suas classificações. Esses canais apresentam políticas e larguras de banda diferentes entre si. As funções de rede que operam no serviço DeMONS são: Gerente DeMONS (DeMONS *Manager* - DM), Classificador de Prioridades (*Priority Classifier* - PC), Filtro por Reputação (*Firewall* - FW), Alocador de Fluxos (*Flow Allocator* - FA), Modelador de Tráfego (*Traffic Policer* - TP) e Roteadores de Canal (*Channel Router* - CR). Particularmente, o DM destina-se a gerenciar o serviço de mitigação de DDoS, monitorando e requisitando operações como de escala e balanceamento de carga para as instâncias de VNF do serviço. As funções PC, FW, FA e TP atuam diretamente na análise, classificação, modelagem e encaminhamento dos fluxos de pacotes, impactando diretamente na qualidade de serviço prestada a cada um deles. Por fim, os CR garantem a diferenciação das rotas de maior e menor prioridade, forçando os fluxos a seguirem os caminhos como especificados pelo PC e FA. Também, a modelagem de tráfego realizada pelo TP, assim como as rotas assumidas pelos CR, é definida por políticas de usuário em tempo de configuração do serviço. A Figura 4.4 ilustra em alto nível a topologia de serviço DeMONS.

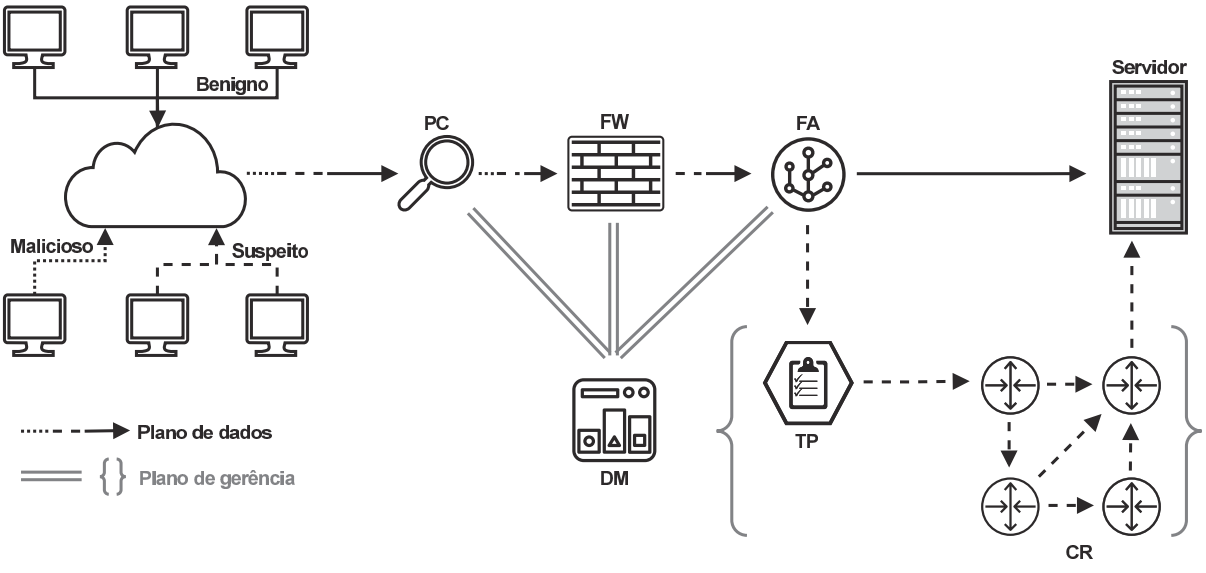


Figura 4.4: Topologia de Serviço DeMONS

Na versão original do DeMONS (Fulber-Garcia et al., 2018), as primeiras três funções que processam o tráfego de rede (*i.e.*, PC, FW e FA) são conectadas por caminhos estáticos e

compartilham a reputação dos fluxos através da função de gerência DM, a qual atua como um ponto centralizador de comunicação. O PC emprega técnicas de detecção de intrusão (*Intrusion Detection Systems* - IDS) para gerar os valores de reputação do tráfego de rede, classificando cada fluxo de entrada no intervalo real [0;1]. Assim, um fluxo com reputação 0 é considerado malicioso, enquanto um fluxo com reputação 1 é dito necessariamente benigno. Reputações entre os valores de extremidade significam que o fluxo é suspeito, seja em menor (próximo a 1) ou maior (próximo a 0) grau. Tanto na ocorrência de um novo fluxo, quanto na modificação da reputação de um fluxo já conhecido, o PC requisita ao DM o salvamento/atualização de suas respectivas informações. Após a classificação, o tráfego de rede é encaminhado ao FW, que consulta a reputação dos fluxos recebidos junto ao DM. O FW então descarta os fluxos considerados maliciosos (*i.e.*, com reputação 0) e encaminha os demais para o FA. O FA, no que lhe concerne, aloca os fluxos prioritários (com as maiores reputações) em um canal de alta prioridade, garantindo aos mesmos o cumprimento de determinadas políticas de qualidade de serviço previamente estabelecidas pelos operadores de rede, como, por exemplo, a não sobrecarga do canal. Por outro lado, fluxos não prioritários são alocados em canais de baixa prioridade, os quais são tipicamente configurados no modelo *best-effort* (sem garantias de qualidade de serviço). Ainda, quando direcionados para os canais de baixa prioridade, os fluxos podem passar por descartes de parte de seu tráfego de rede total baseado em sua reputação, estes executados pelo TP.

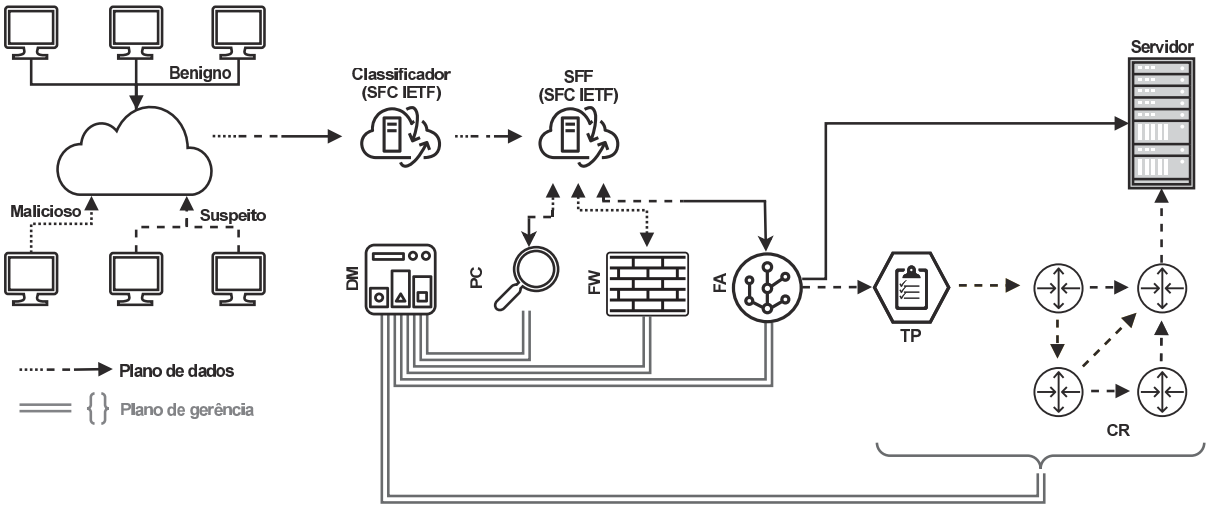


Figura 4.5: Topologia de Serviço do DeMONS + NSH

Para este estudo de caso, o serviço DeMONS foi adaptado no contexto da arquitetura de referência SFC definida pela IETF (Halpern e Pignataro, 2015). O NSH, além de ser empregado no encaminhamento do tráfego de rede na topologia de serviço, é utilizado para realizar controle *in-band* do compartilhamento da reputação dos fluxos de rede, descentralizando o processo de consulta. Desse modo, a tabela de reputações (originalmente mantida, atualizada e acessada no DM) é inserida no PC, sendo a reputação dos fluxos compartilhados através do cabeçalho de contexto (CH) de cada pacote. Além disso, pacotes pertencentes a fluxos benignos e suspeitos têm seu índice de serviço (SI) decrementado em duas unidades após o processamento dos mesmos pelo PC, evitando assim a sobrecarga do FW com a verificação de fluxos que não serão descartados por ele. Finalmente, o FA (assim como o FW) acessa o valor de reputação dos fluxos diretamente no pacote, sem a necessidade de realizar uma comunicação remota com o DM. Ressalta-se que, apesar de o NSHP da plataforma COVEN ser capaz de abstrair o tratamento do NSH das funções de rede, em específico neste estudo de caso, modificações no código-fonte das

mesmas ainda foram necessárias de modo a alterar a forma de gravação/recuperação da reputação dos fluxos (antes centralizado no DM, agora *in-band* no NSH). A Figura 4.5 exibe a topologia do DeMONS modificado para operar com NSH na arquitetura de referência SFC.

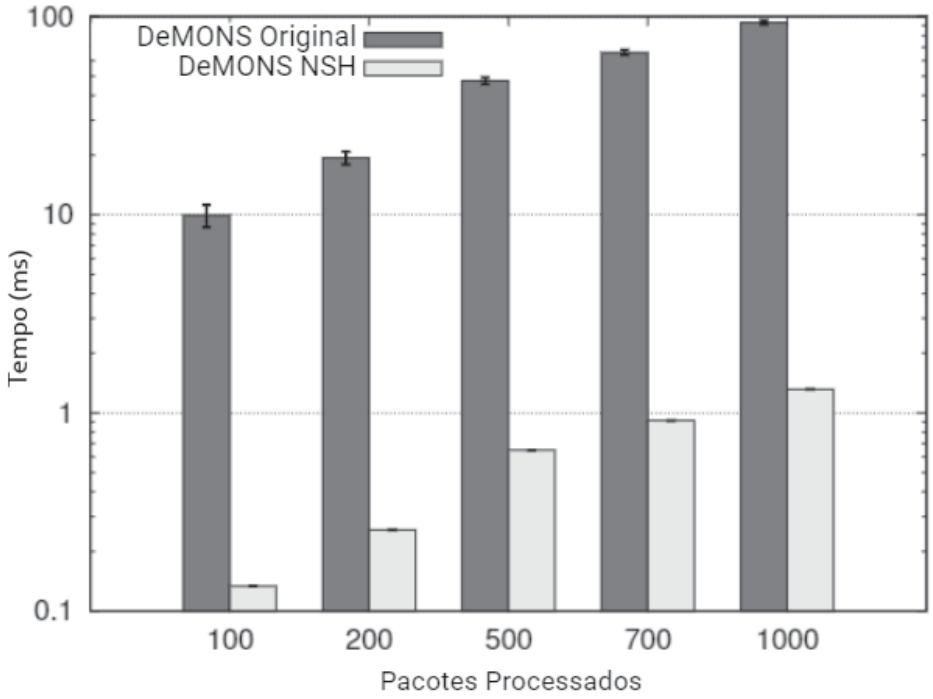


Figura 4.6: Tempo para Consulta de Reputação DeMONS (FW)

O experimento realizado consiste na avaliação da sobrecarga de tempo introduzida pela recuperação e processamento das reputações dos fluxos nos dois modelos do serviço DeMONS apresentados: DeMONS Original (Figura 4.4) e DeMONS NSH (Figura 4.5). Todas as funções de rede foram executadas na plataforma COVEN e, por simplicidade de operação, as medições foram realizadas pela óptica do FW. No FW, dois componentes de função de rede foram desenvolvidos para executar exclusivamente a recuperação das reputações, sendo estes instrumentados para efetivar a medição do tempo total do processo. O componente utilizado no DeMONS original obtém a informação de reputação comunicando-se remotamente com a instância de VNF que executa a função DM (instanciada na mesma máquina física). Essa comunicação é realizada através de *socket* UDP e um protocolo simples, particularmente estabelecido para ela. O segundo componente requisita o CH para o NSHP localmente na plataforma COVEN, assim obtendo a informação da reputação do fluxo ao qual um pacote pertence sem a necessidade de consultas remotas. Durante os testes, a ferramenta Iperf (The Iperf Project, 2021) foi empregada para gerar o tráfego de rede, o qual consiste em pacotes UDP com 1470 bytes. Os resultados de tempo acumulado para consulta da reputação por quantidade de pacotes processados pela função de rede são exibidos na Figura 4.6.

A utilização de controle *in-band* no DeMONS NSH demonstrou resultados com significativas diferenças quando comparados aos seus equivalentes no DeMONS Original, sucedendo uma redução média de 72 a 74 vezes do tempo de execução necessário para a recuperação e processamento da reputação do fluxo ao qual um pacote pertence. Essa redução potencialmente impacta de forma positiva outros indicadores típicos relacionados a funções de rede, por exemplo, aumentando a vazão (em cenários genéricos) e diminuindo a perda de pacotes (em cenários de sobrecarga do serviço). Também, é importante observar que o ganho

acumulado decorrente da adoção da estratégia de controle *in-band* está diretamente associado à taxa de transferência de pacotes no serviço, mas não necessariamente ao volume de dados por ele processado. Esse fenômeno ocorre devido ao tratamento da reputação ser realizado pacote a pacote, independentemente da quantidade de dados presente nos mesmos. Com isso, é possível estimar que, em síntese, cenários onde existem fluxos do tipo guepardo (alta taxa de transferência contínua) e alfa (rajadas com altas taxas de transferência) são beneficiados em maior escala pela versão DeMONS NSH, esta possibilitada, no presente experimento, devido à atuação do NSHP da plataforma COVEN.

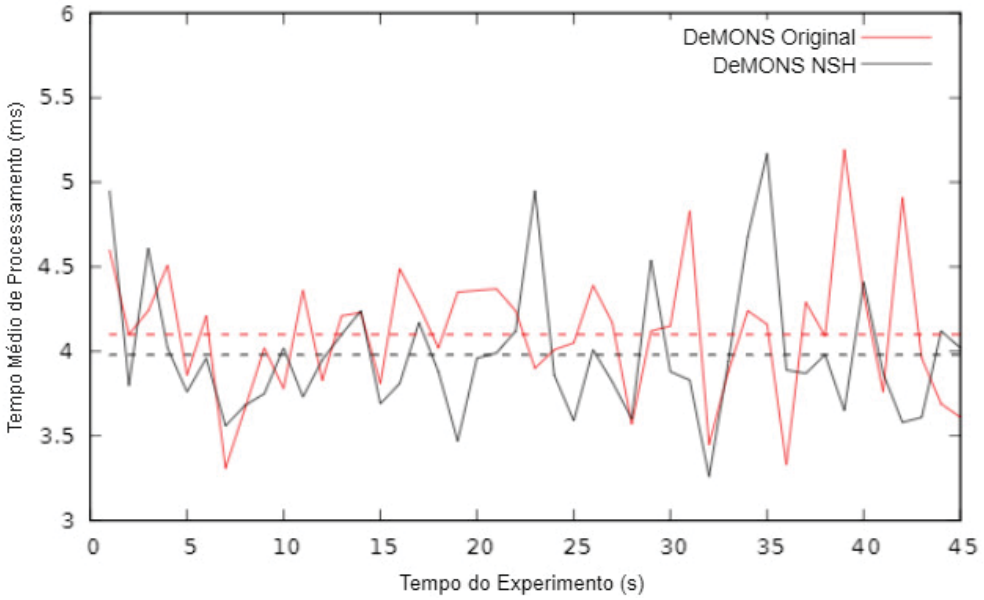


Figura 4.7: Tempo de Processamento por Pacote DeMONS (FW)

Em complemento ao teste de caixa branca do componente de recuperação de reputação usado na função de rede FW, um teste de caixa preta foi executado para determinar a diferença média no tempo total de operação desta função executando sob cada pacote recebido nos dois modelos do serviço DeMONS. Nesse experimento, um fluxo de baixa taxa (1MBps - 1024 pacotes por segundo) de pacotes UDP com reputação 0 (*i.e.*, maliciosos) foi submetido ao serviço. A medição do tempo de execução da função é iniciada no momento do recebimento de um pacote pelo FW, sendo concluída após o descarte do mesmo. A Figura 4.7 apresenta os resultados obtidos, as médias parciais calculadas segundo a segundo são representadas pelas linhas contínuas, e as médias gerais por modelo do serviço são exibidas em linhas tracejadas. Os resultados obtidos evidenciam que, em média, o tempo de processamento de um pacote pela função FW é maior quando a recuperação da reputação do seu respectivo fluxo é realizada através da estratégia centralizada e remota (DeMONS Original), em comparação à estratégia de controle *in-band* e local (DeMONS NSH). O atraso extra é de 0,2 milissegundos para cada pacote processado, ou 5,2% do tempo total de execução. Nesse teste, a função DM (DeMONS original) encontra-se instanciada na mesma máquina física que a função FW. Sendo a assim, o tempo de transmissão é bastante reduzido quando comparado a uma transmissão realizada entre máquinas diferentes em uma mesma rede ou na Internet, reduzindo o impacto deste fator no tempo total de processamento de um pacote pela função de rede. Porém, visto que o atraso calculado é relativo ao processamento individual de um pacote, o atraso acumulado em valores absolutos pode crescer rapidamente quando fluxos de alta taxa com grandes volumes de pacotes são submetidos ao serviço.

O uso do SFF no cenário de execução do DeMONS NSH se apresentou como uma oportunidade para aprimoramento do encaminhamento do tráfego de rede. Esse elemento operacional da arquitetura SFC, o qual pode ser desenvolvido como, por exemplo, uma VNF ou um dispositivo P4, encaminha o tráfego de acordo com o valor de SI presente no NSH de um pacote, determinando o caminho a ser percorrido por este em uma cadeia de serviço. Assim, manipulando o campo de SI do NSH, torna-se possível modificar as características de encaminhamento de um pacote segundo as decisões tomadas por funções de rede durante o processamento do mesmo. No modelo de serviço DeMONS NSH do estudo de caso apresentado, a função FW processa apenas o tráfego malicioso para coletar estatísticas (*e.g.*, quantidade de pacotes recebidos) e, em seguida, descartar o pacote. Quando não há presença de tráfego reconhecidamente malicioso no serviço, além de não processar o tráfego benigno que seria simplesmente encaminhado, a manipulação do SI também permite a desativação do FW, economizando recursos computacionais.

O primeiro estudo de caso visa demonstrar que o uso da plataforma COVEN e de suas funcionalidades exclusivas, como o processamento nativo de NSH, podem ser explorados para aprimorar serviços baseados em NFV. Especificamente para este estudo de caso, a utilização do módulo NSHP permitiu que, com pequenas modificações no código-fonte das funções de rede, o serviço DeMONS passasse de um modelo de controle de reputação centralizado e remoto, para um modelo *in-band* e local. Para isso, as funções foram modificadas para se tornarem conscientes do NSHP, não requisitando das mesmas a realização do processamento do NSH, mas sim a utilização da biblioteca de manipulação do cabeçalho disponibilizado pelo NSHP. Essa biblioteca permite tanto a recuperação, quanto a modificação da informação de reputação que é armazenada no CH de cada pacote. Assim, o NSHP torna-se mais do que um simples processador do NSH, assumindo um papel essencial na dinâmica de troca de informações durante a execução do serviço de mitigação de DDoS. Os resultados obtidos demonstram que a plataforma COVEN (i) foi capaz de executar o serviço DeMONS de maneira correta; e (ii) efetivou o modelo de controle de reputação *in-band* através da manipulação do NSH interfaceada pelo NSHP.

4.3.2 Estudo de Caso: Implantação de Função de Rede com Múltiplos Componentes

O segundo estudo de caso valida a capacidade dos módulos operacionais ITF e PPS de implantarem e executarem funções de rede compostas de múltiplos componentes. Além disso, os potenciais impactos do uso de determinados *frameworks* de desenvolvimento para a implementação de um mesmo componente são avaliados. Nesse contexto, os desenvolvedores de funções de rede podem se beneficiar ao optarem por diferentes *frameworks* considerando os requisitos e desafios específicos dos componentes a serem construídos. Por exemplo, o conjunto padrão de elementos do *framework* CMR não suporta o processamento de *paylods* sofisticados (*e.g.*, inspeção de conteúdos cifrados). Esse tipo de operação é tipicamente fornecido através de componentes desenvolvidos com linguagens de programação mais flexíveis e genéricas. Além disso, permitir que múltiplos *frameworks* cooperem na plataforma de execução de VNF também flexibiliza a composição de funções de rede e aumenta as possibilidades de reuso de componentes. Como efeito secundário, é esperado que a popularização de funções de rede baseadas em componentes facilite a negociação destas em ambientes de *marketplace* NFV, como T-NOVA (Xilouris et al., 2014) e FENDE (Bondan et al., 2019), permitindo a criação de funções e serviços personalizados e completamente adequados às expectativas dos clientes.

O estudo de caso, ilustrado na Figura 4.8, consiste da criação de um *Firewall* de camada sete explorando a utilização de componentes e processamento NSH. O objetivo dessa função de rede é bloquear tráfego de Skype através de uma abordagem de verificação de rastros de registro (Ehlert et al., 2006). O processo de detecção do tráfego de rede Skype é feito em três etapas: (i) detecção de porta (80 e 443); (ii) checagem de padrões presentes no *payload* (primeiros 72 bytes);

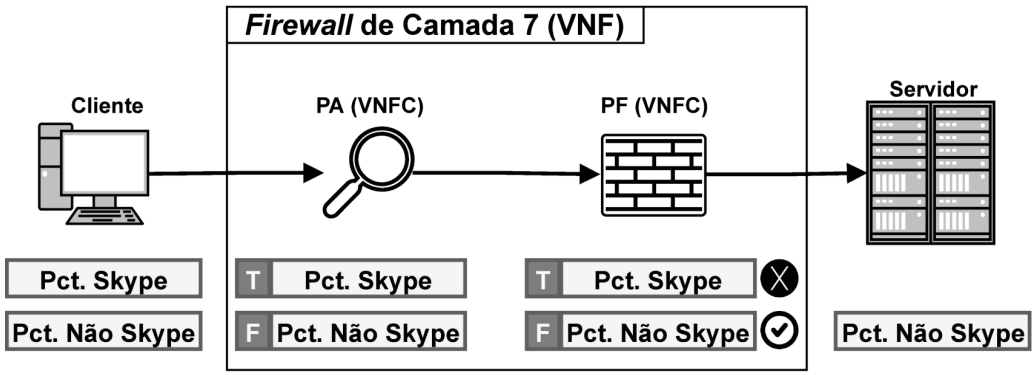


Figura 4.8: *Firewall* de Camada Sete Baseado em Componentes

e (iii) verificação de bytes específicos em diferentes partes do *payload*. As três etapas descritas são implementadas como um componente isolado, chamado de *Packet Analyzer* (PA). Dada a sua complexidade, o PA é desenvolvido com a linguagem de programação Python 3 e auxílio da biblioteca de processamento de pacotes Scapy (SecDev, 2021). Um segundo componente utilizado, chamado de *Packet Filter* (PF), tem diversas implementações, cada uma feita utilizando um *framework* diferente: CMR, C, Java e Python 3. Esse componente dedica-se exclusivamente ao descarte de pacotes compreendidos como tráfego Skype e ao encaminhamento dos demais para o servidor. Os componentes são executados na plataforma COVEN e são conscientes do NSHP. O tráfego de rede consiste em pacotes encapsulados com NSH. De forma geral, quando um pacote ingressa na função de rede, este é analisado à procura de evidências para sua categorização como "Skype" ou "Não Skype" pelo PA. Ao final do processamento desse componente, o pacote é marcado com sua respectiva categoria no CH do NSH (através do NSHP). Em seguida, o PF recupera a marcação realizada pelo PA e decide a ação a ser realizada para o pacote (descarte ou encaminhamento). Ressalta-se que os dois componentes citados são encadeados para a criação da função de rede *firewall* de camada sete, porém ambos são independentes e podem ser incorporados como parte de outras funções de rede complexas.

O *firewall* de camada sete é implantado na plataforma COVEN em todas as suas combinações possíveis de componentes, resultando em quatro modelos diferentes: Python + C, Python + Click, Python + Java e Python + Python. O experimento consiste da medição do *Round Trip Time* (RTT) entre o envio e o recebimento de uma resposta para requisições de cliente/servidor, ambos executados como máquinas virtuais Ubuntu 16.04. O *firewall* de camada sete processa todas as requisições (sentido cliente para servidor), mas realiza o encaminhamento incondicional das respostas (sentido servidor para cliente). Um fluxo de baixa taxa (1MBps - 1024 pacotes por segundo) de pacotes UDP foi submetido ao sistema por 45 segundos para cada combinação de componentes. O fluxo ingressa na porta 80 e seus pacotes contêm os primeiros 72 bytes de *payload* com padrão Skype. Entretanto, os bytes específicos buscados na terceira etapa pelo PA não são correspondentes a aqueles do Skype, sendo todos os pacotes classificados como "Não Skype". Assim, o PF realiza unicamente a operação de encaminhamento após concluída a análise da marcação dos pacotes recebidos. O fluxo de rede foi gerado através da ferramenta Iperf (The Iperf Project, 2021). O objetivo do experimento é verificar o impacto no desempenho causado pela adoção de diferentes *frameworks* de desenvolvimento na execução de componentes análogos (no caso, o PF). Os resultados são apresentados na Figura 4.9.

Os resultados demonstram que, para este estudo de caso, a combinação Python+Python apresentou o pior desempenho relacionado ao RTT, enquanto a combinação Python+C originou as melhores medições. As demais combinações de componentes, Python+CMR e Python+Java, alcançaram resultados próximos entre si, e intermediários em relação às demais. Esses resultados

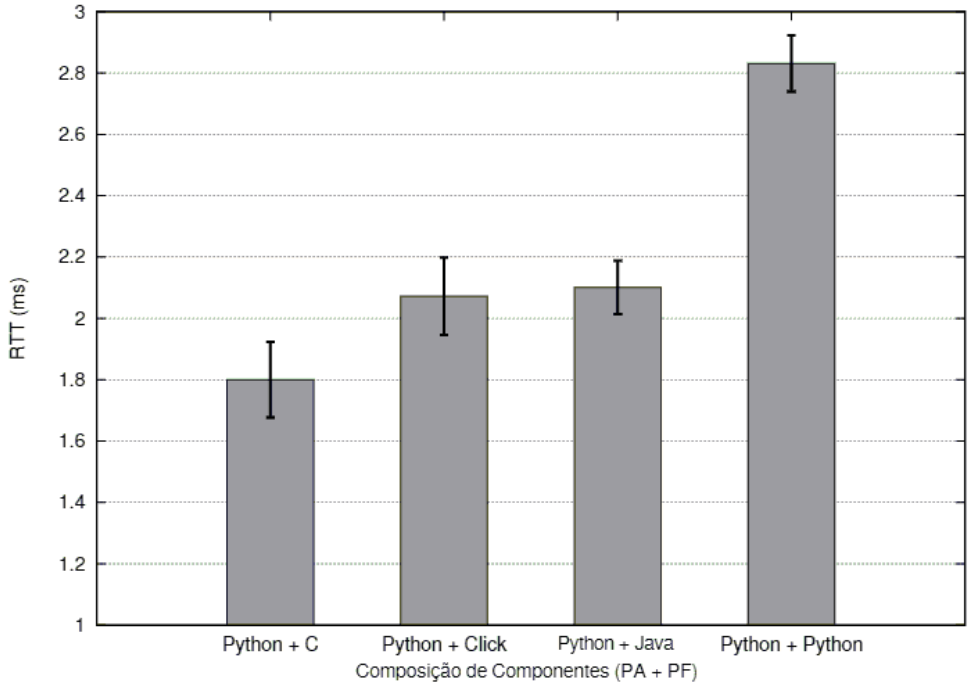


Figura 4.9: RTT das Possíveis Combinações de Componentes PA e PF

são explicados pelas próprias características dos *frameworks* de desenvolvimento empregados no componente PF: Python é uma linguagem que realiza uma pré-compilação de código, gerando um *bytecode* que é posteriormente interpretado pela *Python Virtual Machine*; similarmente, Java também executa uma pré-compilação de código para *bytecode*, sendo este interpretado pela *Java Virtual Machine*. Porém, a pré-compilação Java executa etapas extras de otimização (relacionadas aos blocos de código de uso frequente) desconsideras no contexto da linguagem Python; o CMR realiza uma tradução das suas especificações em alto nível para código em C++, posteriormente compilando o mesmo; e, finalmente, C é uma linguagem de baixo nível e compilada.

De maneira geral, para o componente PF, quanto menor é a abstração de código e a interação com o mesmo em tempo de execução, melhor em desempenho é a sua execução. Entretanto, essa característica é bastante evidente devido ao funcionamento com caráter simplificado desse componente. Componentes mais sofisticados, como o PA, requerem capacidades específicas que se beneficiam do uso de linguagens de alto nível e bibliotecas para tratamento de pacotes (*e.g.*, Python 3 e Scapy ou Java e Pcap4J) mesmo que a utilização destas impacte, até determinado grau de aceitação, no desempenho da função de rede. Conclusões similares podem ser observadas em outros cenários de desenvolvimento, por exemplo, ao incorporar código *assembly* em linguagens de mais alto nível (*e.g.*, C, C++), ou ao utilizar o *Java Native Interface* (JNI) por questões de portabilidade.

Assim como no primeiro estudo de caso, o segundo objetiva explorar as capacidades únicas da arquitetura de plataformas de execução de VNF, esta representada pelo protótipo COVEN. Neste estudo de caso, a possibilidade de composição de uma função de rede através de múltiplos componentes (VNFC) é validada. Essa validação é realizada através da composição e implantação de uma função de *firewall* de camada sete, a qual é formado por dois componentes: PA e PF. A utilização dessa função de rede está diretamente ligada a configuração dos módulos operacionais ITF e PPS da arquitetura proposta. O primeiro módulo (ITF) resolve o encaminhamento interno entre os demais módulos da arquitetura e os componentes da função, preparando todo o caminho a ser percorrido pelos pacotes desde sua obtenção no VNS até o retorno ao mesmo. Essa preparação

inclui a criação das conexões entre o ITF e o NSHP (para encapsulamento e desencapsulamento NSH) e com ambos os componentes que executam a função no PPS (PA e PF). Já o PPS delimita os *frameworks* de desenvolvimento suportados pela plataforma, provendo todo o ferramental para a execução dos componentes implementados a partir deles. Também, esse módulo operacional gerencia os processos do COVEN que executam cada um dos componentes de função. O experimento demonstrou que o modelo arquitetural proposto, assim como sua implementação, permitem com sucesso a composição e execução de funções de rede com múltiplos componentes. Além disso, como resultado imediato dos testes efetuados, demonstrou-se que a escolha dos *frameworks* de desenvolvimento para implementação de componentes é uma tarefa que merece atenção, dado que esta impacta diretamente nos indicadores de desempenho das funções de rede.

4.4 SUMARIZAÇÃO E DISCUSSÃO

A padronização arquitetural do elemento operacional VNF é um passo essencial para satisfazer os múltiplos requisitos e fundamentos do paradigma NFV. Através de uma arquitetura genérica e flexível é possível assegurar que plataformas de execução de VNF se adaptem a ambientes de virtualização com características e disponibilidade de *hardware* e *software* diversas, além de promover um cenário de cooperação natural entre plataformas de diferentes desenvolvedores, mas que seguem padrões comuns. Finalmente, as definições arquiteturais devem ser adaptáveis para permitir a atualização contínua das mesmas, incluindo e satisfazendo novas tecnologias e funcionalidades. Nesse contexto, uma arquitetura para plataformas de execução de VNF foi proposta, sendo discutida no decorrer do presente capítulo. A arquitetura proposta é composta de seis módulos operacionais que executam em um sistema operacional hospedeiro. O sistema hospedeiro é a base virtualizada que provê recursos para a execução e gerenciamento dos módulos operacionais, o qual pode assumir a forma de, por exemplo, uma máquina virtual, um contêiner, ou mesmo um processo isolado e independente. Já os módulos da arquitetura consistem de: (i) VNS - faz o controle da entrada e saída do tráfego de rede na plataforma; (ii) ITF - realiza o gerenciamento do encaminhamento do tráfego de rede entre os módulos operacionais internamente à plataforma; (iii) NSHP - responsável pelo processamento do cabeçalho NSH para funções agnósticas ao mesmo; (iv) PPS - provê recursos para a execução de funções de rede, sejam elas monolíticas ou dividas em componentes de VNF (VNFC); (v) MA - gerencia o ciclo de vida de todos os módulos da arquitetura, além de prover a interface de comunicação com agentes externos; e (vi) EA - são agentes de monitoramento e configuração específicos de uma função de rede ou VNFC.

Para validar o modelo arquitetural proposto, uma plataforma de execução de VNF, chamada *COmprehensive VirtualizEd Nf* (COVEN), foi construída como um protótipo de prova de conceito. A plataforma COVEN adota o sistema operacional Ubuntu Cloud como seu sistema hospedeiro, e implementa o conjunto de módulos operacionais internos através da linguagem de programação Python 3. O VNS disponibiliza uma interface de acesso à rede via *sockets* de camada dois, sendo a única ferramenta de rede virtual nativa. O módulo ITF mantém canais de comunicação com o VNS e NSHP através de regiões de memória compartilhada. Ademais, o ITF encaminha o tráfego de rede à função implantada no PPS utilizando *sockets* de camada três. O NSHP é de ativação opcional, este se destina a desencapsular e reencapsular o cabeçalho NSH de pacotes recebidos e processados, também disponibilizando uma interface para manipulação do CH por parte das funções de rede. O PPS, adicionalmente à manutenção da função de rede e seus componentes, também fornece suporte a cinco *frameworks* de desenvolvimento: CMR, C, Python 3, Java e JavaScript. O MA é o único módulo que deve se manter ativo durante todo o ciclo de vida da plataforma, independentemente de seu estado, sendo ele responsável pelo

recebimento de VNFP, suas validações, configurações e execuções. Toda a comunicação do MA com agentes externos é realizada via uma interface HTTP ou, alternativamente, um servidor TCP. Por fim, os agentes estendidos (EA) devem ser diretamente programados no código-fonte da função de rede ou VNFC, sendo suas operações definidas no descritor da VNF provido como parte de seu pacote de implantação.

Experimentos foram conduzidos em dois estudos de caso objetivando a validação do funcionamento dos módulos da arquitetura proposta através da plataforma COVEN. O primeiro experimento investigou a utilização de processamento NSH no contexto do DeMONS, um serviço de mitigação de DDoS baseado em NFV. Essa funcionalidade foi usada para modificar a forma de acesso à reputação dos fluxos no serviço, de uma dinâmica centralizada em uma função de rede gerente (DM), caso chamado de DeMONS Original, para um controle *in-band* focado no uso do cabeçalho de contexto (CH) NSH, caso chamado de DeMONS NSH. Dois testes foram executados, ambos fazendo medições exclusivamente na função FW. O primeiro teste analisou a sobrecarga do tempo de consulta ao valor de reputação nos dois modelos da solução DeMONS (teste de caixa branca). O segundo teste recuperou o tempo médio de processamento de pacotes nos mesmos modelos de serviço (teste de caixa preta). Em ambos os testes, o DeMONS NSH obteve melhores resultados em relação ao DeMONS Original. O segundo experimento investigou a capacidade da arquitetura proposta de implantar funções de rede com múltiplos componentes de VNF. Com isso, um *firewall* de camada sete dedicado ao reconhecimento e bloqueio de tráfego de rede Skype foi desenvolvido com dois componentes: um PA, exclusivamente implementado em Python 3, e um PF implementado em C, CMR, Java e Python 3. A função de rede foi implantada com todas as combinações de componentes possíveis, sendo o RTT do processamento de pacotes medido para cada uma dessas combinações. Os resultados obtidos no decorrer do segundo experimento evidenciaram que os *frameworks* de desenvolvimento usados influenciam significativamente no desempenho da função de rede.

De maneira geral, os experimentos conduzidos, mais do que seus resultados imediatos, demonstraram que os módulos propostos na arquitetura de plataformas de execução de VNF são viáveis, tendo funcionado corretamente e executado suas operações de forma adequada na plataforma COVEN. Além das funcionalidades tipicamente encontradas em plataformas de execução de VNF existentes (*e.g.*, acesso à rede e processamento de tráfego por funções monolíticas), os experimentos também consideraram estudos de caso onde recursos inovadores da arquitetura proposta foram explicitamente utilizados. Esses recursos consistem, por exemplo, no processamento nativo de NSH pela própria plataforma de execução de VNF (módulo NSHP) e na implantação de funções de rede compostas por múltiplos componentes de VNF (módulo PPS), sendo a dinâmica de encaminhamento interno do tráfego de rede personalizável de acordo com os recursos utilizados (módulo ITF).

5 ARQUITETURA DO ELEMENTO OPERACIONAL EMS

Este capítulo propõe uma arquitetura interna para o EMS, elemento operacional de gerência de VNF. Inicialmente, na Seção 5.1, características e competências gerais, módulos operacionais internos e interfaces de comunicação da arquitetura de EMS são apresentados e discutidos. Também na primeira seção, sugestões de desenvolvimento para os módulos operacionais da arquitetura de EMS são descritas, evidenciando a flexibilidade desta em implementar diferentes categorias de EMS (estabelecidas no Capítulo 2.2). A Seção 5.2 detalha o processo de desenvolvimento de um protótipo de EMS compatível com a arquitetura proposta, este atuando como prova de conceito da mesma. O protótipo é testado através da sobrecarga imposta à execução de operações típicas de gerenciamento de VNF, sendo os resultados exibidos e discutidos no decorrer da Seção 5.3. Por fim, a Seção 5.4 sumariza e discute globalmente o capítulo.

5.1 DEFINIÇÕES ARQUITETURAIS E OPERACIONAIS

Atualmente, não existe nenhuma arquitetura de referência para a estruturação e desenvolvimento de soluções de EMS. Desse modo, características como as de integração, localização, gerenciamento e abrangência são definidas para os EMS existentes como uma consequência e não como finalidade, determinadas através da utilização conjunta e fortemente acoplada das soluções de EMS existentes em sistemas NFV-MANO específicos. Porém, considerando a crescente oferta de sistemas heterogêneos de gerenciamento, orquestração e execução de funções no núcleo da rede, a implementação de soluções de EMS adaptáveis a múltiplos cenários de execução é uma necessidade fundamental para atender aos diversos fundamentos do paradigma NFV, como integração e portabilidade, assim como tratar dos desafios relacionados ao interfaceamento e interoperabilidade em redes modernas baseadas em *software*. Grande parte da importância do EMS no paradigma NFV decorre do seu papel de intermediar a comunicação entre os elementos operacionais de VNF, VNFM, OSS e BSS. A ausência de um EMS acarreta duas dificuldades principais: (i) incapacidade dos elementos de gerência (VNFM e OSS/BSS) acessarem internamente uma instância de VNF genérica através de sua plataforma de execução; e para quando isso é possível, (ii) necessidade de preparação prévia das plataformas de execução de VNF através da instalação de aplicações e configurações extras em seus respectivos sistemas hospedeiros, adaptando estas aos protocolos e tecnologias (tipicamente não padronizados) utilizados pelos elementos de gerência. Essas dificuldades originam-se, principalmente, pela falta de esforços de pesquisa dedicados ao EMS, padronizando o elemento através da criação de uma arquitetura interna e da adoção de protocolos de comunicação estabelecidos por organizações de especificação e recomendação reconhecidas, como ETSI e IETF.

A arquitetura do elemento operacional EMS proposta nesse trabalho é delineada na Figura 5.1. Essa arquitetura é constituída por seis módulos internos principais: (i) Base de Informações de Gerência de EMS (*EMS Management Information Base* - EMIB), (ii) Subsistema de Acesso (*Access Subsystem* - AS), (iii) Roteador Interno (*Internal Router* - IR), (iv) Subsistema de Monitoramento (*Monitoring Subsystem* - MS), (v) Subsistema de VNF (VNF *Subsystem* - VS) e (vi) Módulo de Configuração (*Configuration Module* - CM). Cada módulo fica responsável pela execução de operações específicas de tratamento, encaminhamento e/ou aplicação de requisições de gerência, sejam estas para as instâncias de VNF sob controle do EMS (plano de dados, indicado por setas com linhas contínuas) ou para a administração das configurações de módulos

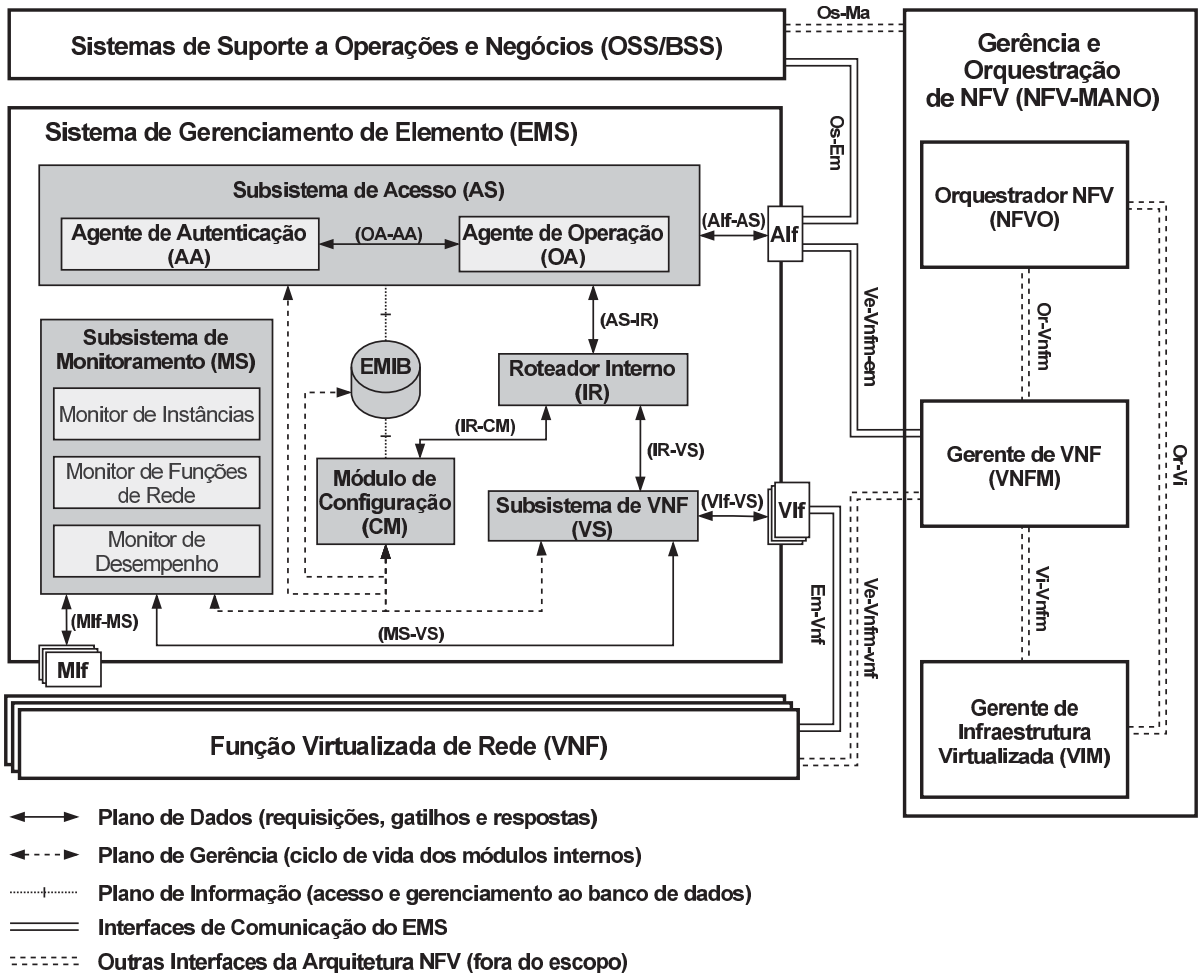


Figura 5.1: Arquitetura do Elemento Operacional de EMS

da própria solução de EMS (plano de gerência, dado pelas setas com linhas tracejadas). Um terceiro plano, o de informações, trata do acesso e manutenção de configurações que definem o modo e o alcance de operação de uma solução de EMS instanciada. Essas configurações são armazenadas na EMIB, manipuladas unicamente através de módulos específicos da arquitetura de EMS (linhas pontilhadas). As interfaces de comunicação estipuladas para o EMS na arquitetura de referência do paradigma NFV são demonstradas por linhas duplas contínuas, enquanto outras interfaces fora do escopo do EMS consistem de linhas duplas tracejadas.

Além dos módulos principais, a arquitetura também conta com submódulos operacionais: Agente de Autenticação (*Authentication Agent* - AA) e Agente de Operação (*Operations Agent* - OA), ambos inseridos no contexto do AS; e agentes de monitoramento de instâncias virtuais, funções de rede, qualidade de serviço e qualidade de experiência, estes enquadrados no MS. Ainda, interfaces externas são usadas para habilitar a comunicação de módulos operacionais particulares com elementos fora do escopo da arquitetura: Interface de Acesso (*Access Interface* - AIf), conectada ao AS; Interface de Monitoramento (*Monitoring Interface* - MIf), conectada ao MS; e Interface de VNF (*VNF Interface* - VIf), conectada ao VS. Todos os módulos da arquitetura de EMS foram projetados para serem pouco acoplados entre si, sendo o funcionamento de uma solução completa e a cooperação entre os módulos individuais garantidos pela programação de pontos de acesso bem definidos entre eles. Dessa forma, módulos operacionais podem ser naturalmente substituídos conforme a evolução e surgimento de novas tecnologias e funcionalidades. Os

relacionamentos e competências dos módulos da arquitetura de EMS proposta são descritos a seguir:

- **Base de Informações de Gerência de EMS (EMIB)** – Módulo que consiste em um ponto de armazenamento de informações relacionadas à configuração e administração de um EMS. A EMIB pode ser programada como um simples sistema de pastas e arquivos, ou mesmo como um banco de dados complexo. A escolha do modelo de implementação deste módulo deve considerar a complexidade e quantidade de informações mantidas pelo EMS pretendido. Exemplos de informações presentes na EMIB são: dados e credenciais de usuários do EMS; descrição, localização e modelos de acesso das instâncias de VNF gerenciadas; agentes e políticas de monitoramento executados continuamente; além de informações de suporte a comunicação e compatibilidade com plataformas de outros elementos da arquitetura NFV (*e.g.*, VNF, VNFM e OSS/BSS). Também, a EMIB deve fornecer um meio de acesso organizado às suas informações, este utilizado pelos módulos operacionais conectados ao plano de informação do EMS. Esse meio de acesso pode ser habilitado de diferentes formas, por exemplo, ao usar uma biblioteca padrão de acesso local, uma interface de comunicação (*e.g.*, HTTP, *socket*) ligada à rede "*localhost*" ou um sistema externo, tal qual um gerente de banco de dados (*e.g.*, SQLite (The SQLite Project, 2021), PostgreSQL (The PostgreSQL Project, 2021), MongoDB (The MongoDB Project, 2021)).

- **Subsistema de Acesso (AS)** – Uma vez enviadas requisições para um EMS, o AS é o módulo responsável pelo recebimento, validação, autenticação e encaminhamento das mesmas para o Roteador Interno. Do mesmo modo, após processada uma requisição, o AS deve encaminhar o seu resultado para o requerente. Este módulo contempla dois agentes: Agente de Operação (OA) e Agente de Autenticação (AA). O OA define e hospeda a Interface de Acesso (AIf) do EMS. É no OA que estão disponibilizadas todas as operações de acesso externo, sejam estas relacionadas ao gerenciamento de instâncias de VNF ou à administração do próprio EMS. Sendo assim, minimamente, o OA deve prover acesso às operações protocolares da interface Ve-Vnfm-em (NFVISG, 2020a). O AA, no que lhe concerne, verifica se as requisições recebidas são oriundas de usuários válidos e com privilégios suficientes para as realizarem. Esse agente, em particular, é de utilização opcional, porém é constantemente recomendado em documentos de especificação que abordam o elemento EMS (NFVISG, 2019, 2020d). Quando utilizada, a autenticação do AA deve ocorrer após a validação de uma requisição pelo OA e antes do encaminhamento desta para o Roteador Interno.

- **Roteador Interno (IR)** – O módulo de IR direciona todas as requisições (recebidas do AS) internamente ao EMS. Ou seja, requisições por operações de gerenciamento de instâncias de VNF são enviadas para o Subsistema de VNF. Ainda, requisições de manutenção do próprio EMS são repassadas ao Módulo de Configuração. O IR deve conseguir rastrear a totalidade do fluxo de encaminhamento de uma requisição no EMS, ligando-lhe aos seus resultados e eventuais exceções e erros. É normal que mais de uma estrutura de mensagem seja usada no IR, por exemplo, na definição de um corpo de mensagem específico para requisitar o Subsistema de VNF ou para requisitar o Módulo de Configuração. Ao final, informações do resultado do processamento de uma requisição são retornadas ao AS para envio ao seu respectivo requerente.

- **Subsistema de Monitoramento (MS)** – Este módulo oferece um ambiente de execução para monitores de instâncias de VNF. Em outras palavras, usuários do EMS podem

estabelecer, a partir das operações disponíveis no Subsistema de VNF, agentes de monitoramento que são executados continuamente. Esses agentes devem prover suporte ao desenvolvimento de gatilhos, possibilitando assim o envio de notificações, alertas e relatórios para destinatários específicos quando eventos de interesse são identificados. O envio dessas mensagens é realizado por Interfaces de Monitoramento (MIf) próprias dos agentes, evitando assim a sobrecarga do AS. A implementação dos agentes de monitoramento pode ser realizada a partir de processos e *threads* sob o controle direto do EMS/MS, ou a partir de virtualização (tipicamente aninhada e leve) sob o controle do sistema operacional hospedeiro do EMS. No último caso, o sistema hospedeiro deve prover ao EMS/MS todos os privilégios necessários para que este opere os agentes de monitoramento virtualizados. Apesar de, *a priori*, um agente de monitoramento ter acesso a qualquer operação disponibilizada no Subsistema de VNF, aquelas mais comumente utilizadas estão relacionadas ao monitoramento de uma instância virtualizada (*e.g.*, envio de *heartbeats*) e da função de rede sendo executada nela (*e.g.*, filtragens e inspeções), além de métricas genéricas de desempenho (*e.g.*, atraso e *jitter*).

- **Subsistema de VNF (VS)** – O VS é um dos principais módulos da arquitetura proposta. É a partir dele que se estabelece a comunicação entre o EMS e uma ou mais plataformas de execução de VNF. Este módulo deve reconhecer previamente o conjunto de operações de VNF possíveis, além dos procedimentos de execução e modelos de resultado de cada uma delas. Assim, mais do que aplicar uma operação em uma instância de VNF, o EMS pode interpretar o seu resultado e responder adequadamente ao requerente. Toda a comunicação descrita ocorre através de Interfaces de VNF (VIf). A gerência dessas interfaces varia de acordo com o projeto e implementação do EMS, mas a existência de múltiplos VIf é natural. Exemplos de cenários onde vários VIf podem ser usados são: definição de um VIf para cada plataforma de execução de VNF suportada pelo EMS e criação de um VIf exclusivo para atender um agente de monitoramento do MS (evitando a sobrecarga das interfaces de uso comum). No caso de soluções de EMS categorizados como restritas, em que operações extras além das previstas no protocolo Ve-Vnfm-em são ofertadas, o AS deve configurar o AIf em concordância com as operações não protocolares suportadas e oferecidas pelo VS. Também, nesse caso em particular, múltiplas operações não protocolares oferecidas pelo VS podem ser executadas de forma conjunta e organizada, gerando assim uma única operação de alto nível disponibilizada no AS/AIf.

- **Módulo de Configuração (CM)** – O objetivo primário do CM é estabelecer as configurações de uma instância de EMS (armazenadas na EMIB) em seus respectivos módulos operacionais. Assim, após a inicialização do EMS, os módulos se encontrarão completamente operacionais e responsivos a requisições. Após a execução das tarefas de inicialização, o CM se torna um módulo administrativo disponível, assim se mantendo até o final do ciclo de vida do EMS. Dessa forma, operações como mudanças de configuração do EMS, ativação e desativação de agentes de monitoramento, inclusão e exclusão de instâncias de VNF gerenciadas e registro de novos usuários são conduzidas através dele. Essas operações administrativas são também disponibilizadas para os usuários através do AS, e suas efetivações não devem comprometer o desempenho das tarefas de gerenciamento de instâncias de VNF sendo executadas. É importante ressaltar que o CM mantém o controle generalizado do EMS, sendo capaz de acessar e modificar qualquer um de seus módulos configuráveis. Sendo assim, é importante garantir que o

acesso a esse módulo seja realizado apenas por usuários com um nível de privilégio adequado.

De maneira geral, quando uma requisição é recebida pelo AS através do AIf (AIf-AS), o OA procede com a sua validação considerando seu conjunto de operações disponíveis. Caso a requisição esteja conforme, esta é encaminhada ao AA (OA-AA) que autentica o requerente e verifica se o mesmo possui privilégios para a execução da operação pretendida. Caso a requisição seja válida e autorizada, ela é então encaminhada do AS para o IR (AS-IR). O IR realiza o registro interno para rastreio da requisição e a encaminha para o VS (no caso de uma operação de gerenciamento de VNF – IR-VS) ou ao CM (no caso de uma operação de (re)configuração do EMS – IR-CM). Considerando que a requisição é de gerenciamento de VNF, o VS realiza a identificação da plataforma de execução de VNF que está sendo requisitada e o endereço de sua respectiva instância virtualizada (essas informações são incluídas pelo AS durante o processo de validação da requisição). Atentando a essas informações, a requisição é remodelada e encaminhada através da VIf adequada (VS-VIf). Por outro lado, se a requisição for de uma operação de configuração do EMS, o CM identifica o módulo alvo da mesma e atua com intermediário para sua execução através do plano de gerência/informação. Os resultados das operações seguem o caminho inverso no EMS, sendo finalmente encaminhados ao requerente pelo AS. Os demais pontos de acesso interno que devem ser destacados são o MS-VS, que é utilizado pelo MS para requisitar periodicamente ao VS operações de monitoramento de VNF de seus agentes; e o MS-MIf, sendo usado no envio de mensagens geradas pelos agentes de monitoramento do MS para seus respectivos assinantes.

A Figura 5.2 apresenta, através de um diagrama de sequência, o processo específico de entrada de requisições e saída de resultados na arquitetura de EMS proposta. O diagrama ilustra desde a entrada até a saída de uma requisição que foi processada com sucesso. Esse processo é referido no decorrer do trabalho como "E/S de Requisições". Inicialmente, uma entidade requerente envia uma requisição ao EMS, esta é recebida e validada pelos submódulos do AS. O AIf, então, analisa a requisição quanto ao seu formato de pacote. Por exemplo, sendo o AIf um servidor HTTP, qualquer requisição que não siga o seu respectivo protocolo é descartada. Após validada neste nível, a requisição é encaminhada ao OA, o qual verifica a disponibilidade da operação requisitada. Nesse caso, se a requisição for por uma operação inexistente, esta é descartada e um código de erro é enviado como resposta ao requerente (por exemplo, 404 quando o protocolo HTTP é usado). O mesmo acontece quando a operação está disponível, porém os argumentos providos são inválidos ou insuficientes (para o caso HTTP, um código de erro 400 pode ser retornado). Em seguida, no caso de um método de autenticação ser utilizado, a requisição é enviada ao AA, que valida o requerente e seu nível de privilégio no EMS. Se o requerente não é autenticado, um código de erro é retornado ao mesmo (em HTTP, os códigos 401 e 403 podem ser aplicados). É importante ressaltar que, para autenticar o usuário, o AA acessa diretamente a EMIB para verificar a sua existência, a conformidade entre o autenticador recebido e o registrado (*e.g.*, senha, certificado) e o seu nível de privilégio (o conjunto de informações é denominado "Dados de Usuário" no diagrama de sequência). Sendo a autenticação bem-sucedida, uma confirmação é enviada ao OA, que encaminha a requisição para os módulos operacionais que efetivamente a aplicam/executam no EMS. Por fim, após o retorno desses módulos operacionais, o OA recebe o resultado da requisição e o encaminha para o seu respectivo requerente através do AIf.

A Figura 5.3 mostra um diagrama de sequência detalhando uma operação de configuração no EMS. Neste trabalho esse processo é denominado como "Operação de CM". No diagrama, a linha de vida denominada "E/S de Requisições" opera como a entidade requerente da operação e representa todos os processos de entrada e saída de requisições e resultados exibidos na Figura

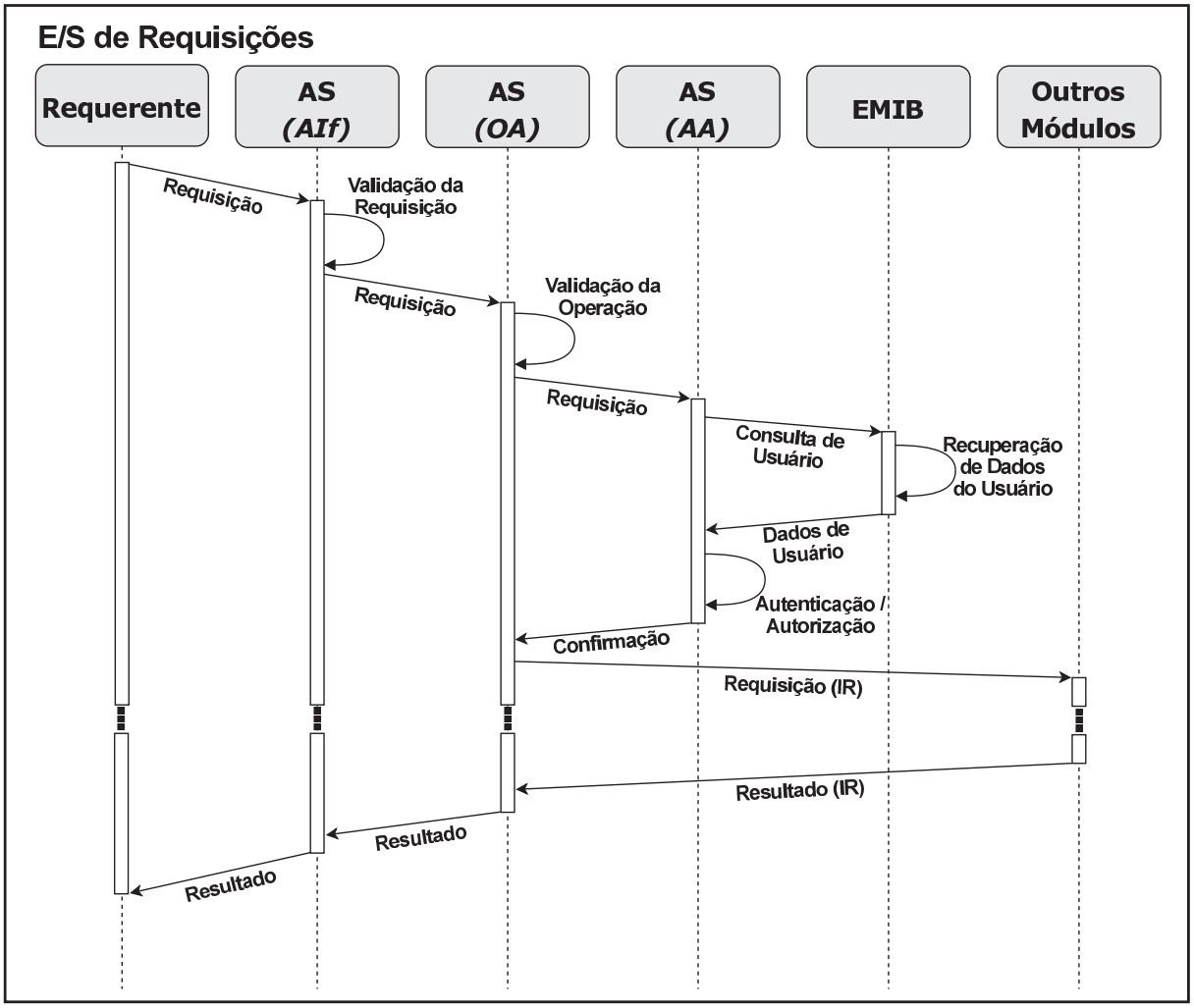


Figura 5.2: Entrada e Saída de Dados no EMS

5.2. Por sua vez, as demais linhas de vida da Figura 5.3 representam a linha de vida "Outros Módulos" presente na Figura 5.2. Dessa forma, após todas as validações e encaminhamentos do AS, o IR recebe a requisição, procedendo com o seu registro de rastreio. Esse registro atribui uma identificação única para a requisição, assegurando que todos os resultados posteriores decorrentes de sua aplicação/execução sejam devidamente informados ao requerente correto. Em seguida a requisição é enviada ao módulo CM, o qual identifica a operação a ser executada e verifica se a mesma é consistente com o estado do EMS. Nessa etapa, erros semânticos da requisição podem ser detectados, por exemplo, requisitar a inicialização de um agente de monitoramento que já está sendo executado, ou ainda requisitar a modificação de um usuário que não existe na EMIB. Nessas situações, apesar de a requisição estar correta, sua aplicação é inviável e um código de erro é retornado (409 no caso de um servidor HTTP). Tanto as validações, quanto a aplicação/execução da operação requisitada podem demandar múltiplos acessos a EMIB, sejam eles para consultas ou para inserção, modificação e remoção de dados. Operações como a remoção de um usuário são exclusivamente tratadas com a atualização da EMIB, porém, outras operações, como a inicialização de um agente de monitoramento, necessitam ser aplicadas em módulos específicos, o que é realizado após a atualização da EMIB. Por fim, o resultado das operações é retornado ao IR, que identifica o requerente e encaminha as informações ao AS.

A Figura 5.4 exibe o diagrama de sequência referente a uma operação de VNF sendo executada através do EMS. Da mesma forma que no diagrama anterior (Figura 5.3), a linha de vida

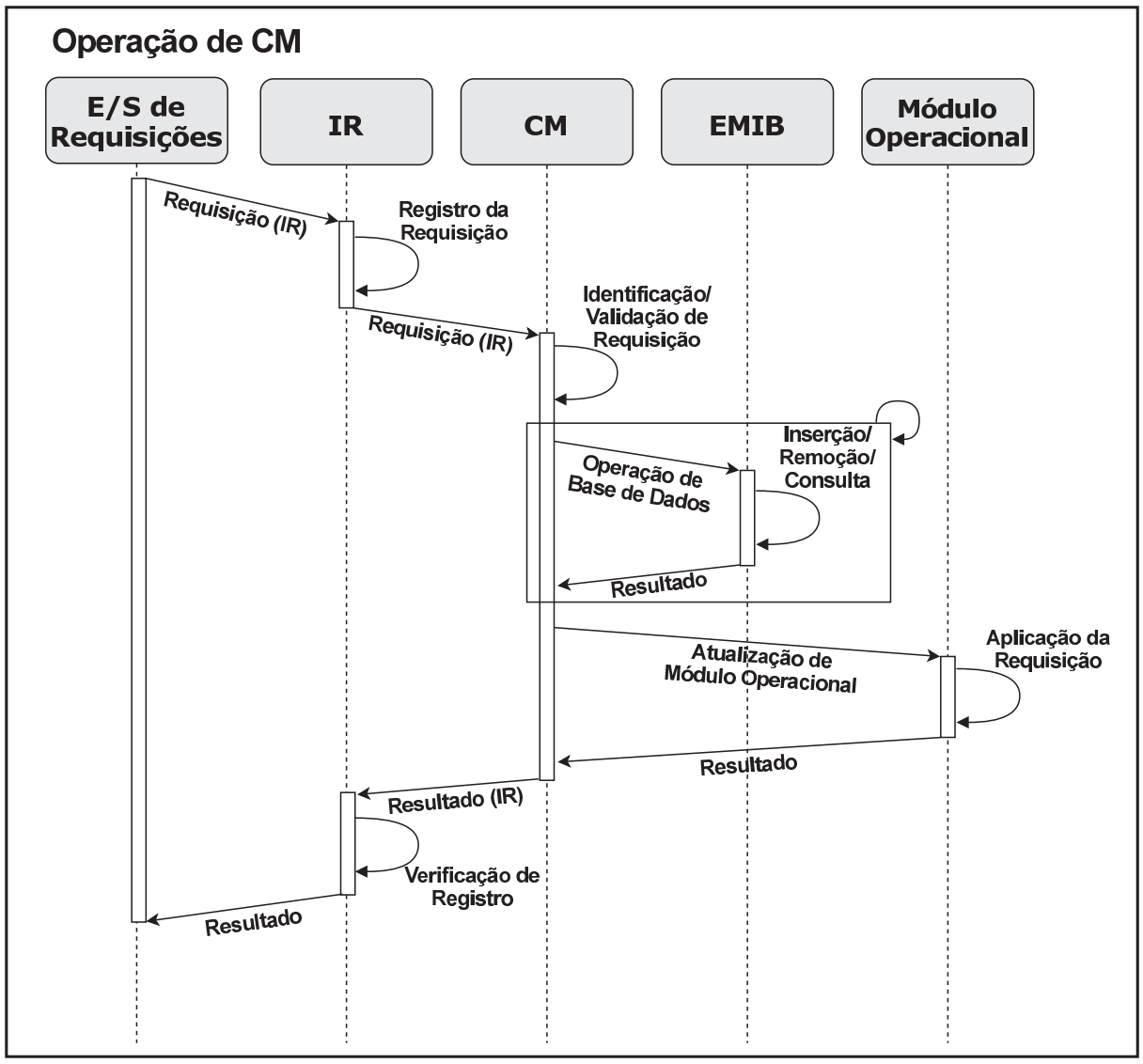


Figura 5.3: Operação de Configuração no EMS

denominada “E/S de Requisições” indica o processo da Figura 5.2. Por sua vez, as outras linhas de vida da Figura 5.4 correspondem à linha de vida “Outros Módulos” da Figura 5.2. Neste trabalho, o processo apresentado a seguir é intitulado de “Operação de VS”. Inicialmente, um processo de registro de requisição idêntico ao feito para operações de configuração do EMS é realizado. Assim, a requisição é identificada unicamente pelo IR e repassada de forma direta ao módulo de VS. O VS valida a operação em relação à plataforma de execução de VNF sendo requisitada, determinando sua adequação ao MA daquela plataforma. Se a operação é dita inválida nesse nível, um código de erro é retornado como resultado (por exemplo, considerando um AS HTTP, 409). Caso contrário, o VS solicita ao CM uma consulta à EMIB para recuperar os dados da VNF alvo da requisição. Esses dados incluem, por exemplo, o IP/Porta de acesso para comunicação com a instância virtual da VNF através de seu MA. Com isso, o VS prepara a requisição para a instância de VNF considerando as tecnologias e protocolos de comunicação específicos da sua respectiva plataforma de execução. Essa requisição é encaminhada à instância de VNF através do VIf, interface que também recebe, posteriormente, o resultado da operação. O resultado é então formatado pelo VS em uma mensagem interna que é repassada ao IR. Finalmente, o IR

recupera o registro da requisição e identifica o requerente, remetendo os resultados ao mesmo através do AS.

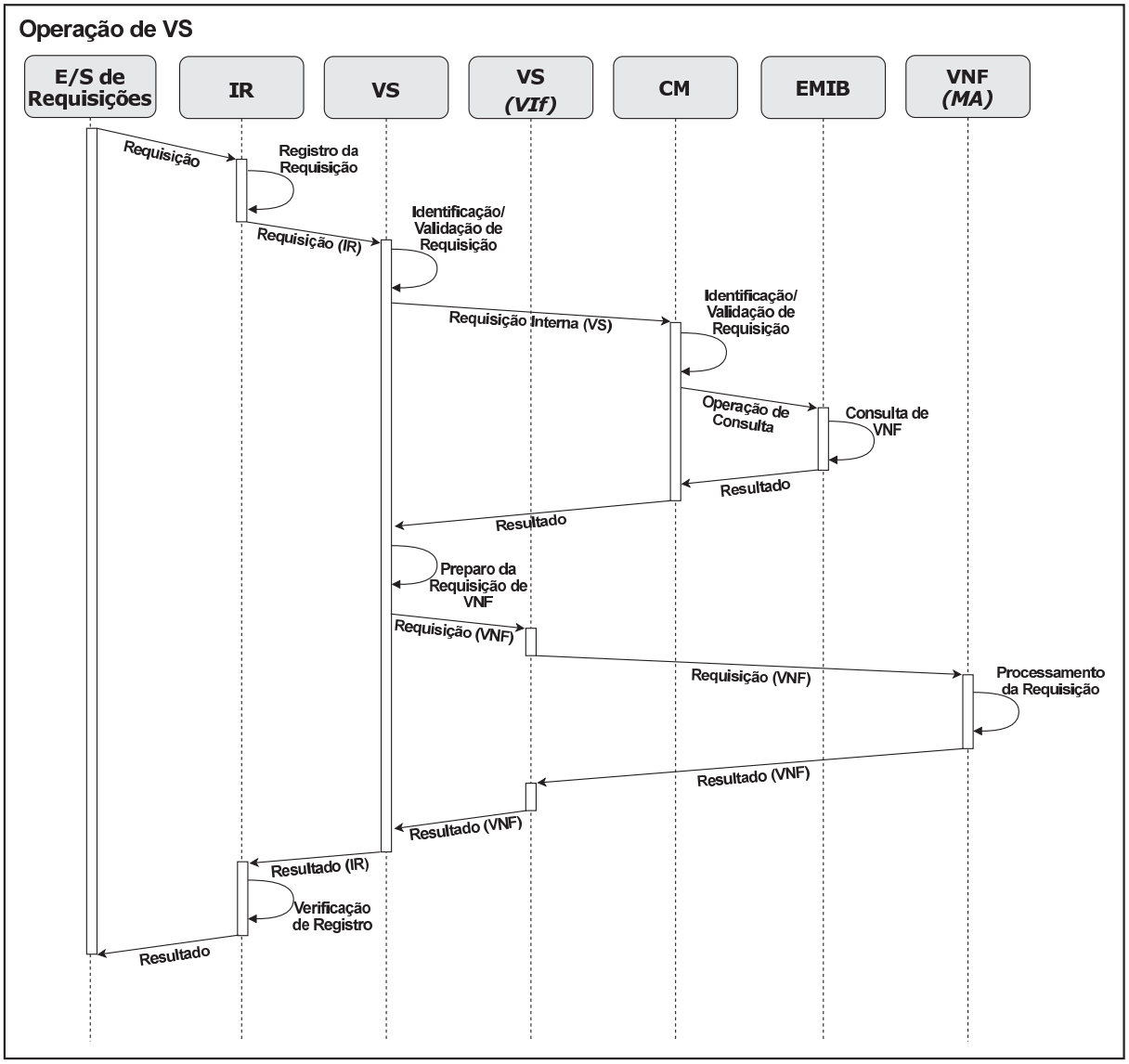


Figura 5.4: Operação de VNF no EMS

A arquitetura de EMS apresentada nesta seção conta com módulos operacionais internos suficientes para a efetivação de operações FCAPS em instâncias de VNF variadas, considerando as interfaces disponibilizadas pelos MA de suas respectivas plataformas de execução. Através da implementação desses módulos internos, é possível desenvolver desde soluções de EMS simples, como aquelas dedicadas e genéricas, até sistemas complexos, guarda-chuva e restritos. Além disso, funcionalidades exclusivas e estratégias de multi-compatibilidade com diferentes formas de implementação de outros elementos operacionais do paradigma NFV (*e.g.*, VNFM, OSS, BSS, VNF) podem ser incluídas nos subsistemas da arquitetura. É importante ressaltar que, apesar de modelos de implementação e troca de mensagens serem sugeridos no decorrer da presente seção, a arquitetura de EMS é genérica e flexível, não sendo limitada por nenhuma tecnologia, linguagem de programação ou interface de comunicação específica. Por fim, a arquitetura proposta conta com módulos internos fracamente acoplados, *i.e.*, não dependem diretamente de outros módulos desde que suas interfaces e pontos de comunicação sejam compatíveis, evitando que soluções de EMS assumam características monolíticas de desenvolvimento. Assim,

modelos de desenvolvimento e atualização independentes e distintos podem ser adotados para a implementação e manutenção de cada módulo operacional em particular.

5.2 IMPLEMENTAÇÃO DO PROTÓTIPO HOLMES

Um protótipo foi desenvolvido como prova de conceito da arquitetura de EMS apresentada na Seção 5.1. O protótipo, chamado *Holistic, Lightweight and Malleable EMS Solution* (HoLMES)[1], implementa todos os módulos operacionais internos e interfaces previstas na arquitetura de EMS, sendo completamente compatível com a arquitetura de referência NFV da ETSI (NFVISG, 2014). O HoLMES foi projetado para ser genérico e comunicar-se com diferentes plataformas de execução de VNF e de NFV-MANO. Além disso, sua implementação permite que várias instâncias de VNF sejam gerenciadas ao mesmo tempo por uma única instância de EMS. Assim, o HoLMES é classificado como **MANO (independente), externo, restrito e guarda-chuva**. O protótipo é oferecido como um pacote de instalação isolado ou como uma imagem pré-configurada no sistema operacional Ubuntu Cloud (Canonical, 2021b). Esse sistema operacional provê um núcleo de operações e um ambiente para a instalação de aplicações, sendo relativamente econômico em termos da demanda de recursos computacionais e executando os módulos da plataforma sem qualquer necessidade de modificação de código.

O HoLMES foi implementado através da linguagem de programação Python 3. Em particular, o Subsistema de Acesso (AS) utiliza a biblioteca Flask para operacionalizar a Interface de Acesso. Ainda, nesse subsistema, o Agente de Operação (OA) disponibiliza nativamente as funções do protocolo Ve-Vnfm-em (NFVISG, 2020c) cujo fluxo de requisição acontece do VNFM para o EMS. Como destaque, o Agente de Operação suporta a execução de *drivers* de comunicação entre EMS e plataformas de NFV-MANO (em específico, com o VNFM dessas plataformas). Esses *drivers* providenciam as funções do Ve-Vnfm-em que o fluxo de requisição parte do EMS para o VNFM. Para isso, um *template* é disponibilizado pelo protótipo (*VnfmDriverTemplate*), habilitando seus operadores a programar *drivers* de VNFM sob demanda. O Agente de Autenticação (AA) oferece métodos com diferentes níveis de segurança para verificar a autenticidade das entidades que geraram as requisições que chegam ao EMS, exemplos desses métodos vão desde a aceitação incondicional (ausência de autenticação) até autenticação baseada em usuário e senha. O Agente de Autenticação é programado de maneira modular e pode ser facilmente estendido com poucas alterações em seu código-fonte. O Roteador Interno (IR) consiste em um ponto central de troca de mensagens entre os módulos do EMS. O Roteador Interno é responsável por criar rótulos de identificação de requisições e resultados, encaminhando os mesmos para os módulos necessários através de chamadas de função.

O Subsistema de Monitoramento (MS) foi implementado utilizando a biblioteca de processamento paralelo do Python *multiprocessing*. Esse subsistema gerencia múltiplos agentes de monitoramento executados como processos. Os agentes comunicam-se com instâncias de VNF através do Subsistema de VNF, executando rotinas de monitoramento a partir das operações disponibilizadas em suas respectivas plataformas de execução. Um agente de monitoramento é assinado por usuários autorizados de uma instância do HoLMES, os quais passam a receber notificações de monitoramento disparadas na ocorrência de eventos preestabelecidos. Essas notificações são enviadas por Interfaces de Monitoramento (MIf) definidas pelos próprios agentes. A programação de agentes de monitoramento deve ser realizada considerando um modelo importável próprio do HoLMES, este fornecido como parte do protótipo (*MonitoringAgentTemplate*). O último subsistema, o Subsistema de VNF (VS), realiza a comunicação entre o HoLMES e os agentes de gerência das plataformas de execução de VNF. Como plataformas de execução de

[1]Disponível em `https://github.com/ViniGarcia/HoLMES`

VNF heterogêneas podem ser utilizadas, o Subsistema de VNF suporta a criação e aplicação de *drivers* de forma similar àquela empregada para a comunicação com o VNFM no Agente de Operações. Assim, cada plataforma de execução de VNF apresenta um *driver* correspondente que especifica suas operações de gerenciamento, além de definir uma Interface de VNF (VIf) conforme a sua tecnologia e protocolo de comunicação em particular (*e.g.*, REST, *Socket* e RMI). O *template* usado para a criação dos *drivers* relacionados ao Subsistema de VNF também é provido junto ao protótipo (*VnfDriverTemplate*).

A Base de Informações de Gerência de EMS (EMIB) do HoLMES é construída com o banco de dados relacional SQLite (The SQLite Project, 2021). A escolha do SQlite foi feita devido ao seu bom desempenho, simplicidade e disponibilidade ampla de bibliotecas. Nessa base de dados são armazenadas todas as informações necessárias para a execução e manutenção do sistema, contendo tabelas como de usuário, plataformas de execução de VNF, instâncias de VNF, agentes de monitoramento, entre outras. A EMIB, assim como os demais módulos do protótipo, é gerenciada pelo Módulo de Configuração (CM), o qual funciona como um eixo de execução de operações de configuração e atualização do sistema. Assim, todas as requisições não relacionadas a uma instância de VNF em particular passam pelo Módulo de Configuração, sendo então aplicadas de fato dentro do HoLMES. Essas operações de configuração e atualização do protótipo também são disponibilizadas (utilizando um protocolo próprio) pelo Agente de Operação, sendo acessíveis por usuários com privilégios de administradores do sistema. Em relação à comunicação interna, o HoLMES utiliza três tipos principais de mensagens: (i) MensagemRI, modelo padrão de troca de mensagens com o RI, mantém o rastreio de origem e destino do conteúdo (utilizado nas conexões Sa-Ri, Ri-Mc e Ri-Vs); (ii) MensagemGI, conteúdo enviado em uma MensagemRI quando o destinatário é o MC (Ri-Mc), contém informações necessárias para operações de configuração/atualização do EMS; e (iii) MensagemVS, conteúdo de uma MensagemRI quando o destinatário é o VS (Ri-Vs), contém informações para a execução de requisições em instâncias de VNF.

5.2.1 Interconexão dos Módulos Operacionais

As conexões entre módulos e submódulos internos à arquitetura, tanto no plano de dados (*i.e.*, AIf-AS, OA-AA, AS-IR, IR-CM, IR-VS, VIf-VS, MS-VS, MS-MIf), quanto no plano de gerência (conexões entre CM e outros módulos) e de informações (conexão entre CM/AS e EMIB) são majoritariamente desenvolvidas através de chamadas de funções e compartilhamento de memória, utilizando argumentos e estruturas padronizadas. Essas conexões ocorrem no contexto de relações intraprocessuais (troca de mensagens entre (sub)módulos distintos executados por um mesmo processo) e interprocessuais (troca de mensagens entre processos). As interfaces (AIf, VIf e MIf), entretanto, tipicamente consideram tecnologias e protocolos de comunicação específicos na implementação da comunicação entre os módulos operacionais da arquitetura e elementos externos à mesma. Em particular, o AIf é implementado como um servidor HTTP. Para isso, as orientações ETSI para implementação HTTP do protocolo de operações da interface Ve-Vnfm-em (NFVISG, 2020c) foram atendidas. Já as operações relacionadas à (re)configuração dos módulos de EMS (operações de CM) seguem um padrão próprio da arquitetura. No AIf, o sistema de *drivers* de compatibilidade com plataformas NFV-MANO e OSS/BSS é utilizado para intermediar a comunicação de ambos os sentidos. Um *driver* faz a tradução do protocolo do EMS para o do elemento destino e vice-versa (ou seja, disponibilizando operações de EMS – via HTTP – ou requisitando as de outros elementos operacionais da arquitetura ETSI NFV). O VIf pode assumir diferentes tecnologias de comunicação suportadas pela linguagem de programação Python 3 e suas bibliotecas. Nesse cenário, o sistema de *drivers* é utilizado exclusivamente para requisitar operações às diferentes plataformas de execução que hospedam instâncias de VNF,

sendo o protocolo e tecnologia de comunicação implementado caso a caso. Por fim, a interface de comunicação externa do MIf, similarmente ao VIf, é implementada pelo desenvolvedor do agente de monitoramento ao qual esta se vincula, visto que a comunicação parte exclusivamente do EMS para os assinantes dos agentes disponíveis. Sendo assim, as limitações para o MIf também derivam do suporte às tecnologias de comunicação provido pela linguagem de programação adotada e suas bibliotecas.

5.3 EXPERIMENTAÇÃO E RESULTADOS

Esta seção detalha e discute a experimentação realizada para avaliar a funcionalidade dos módulos operacionais da arquitetura de EMS apresentados na Seção 5.1. As avaliações consideram a implementação da arquitetura na solução protótipo HoLMES, como especificada na Seção 5.2. A experimentação realizada, além de validar a execução de operações de FCAPS em plataformas de execução de VNF heterogêneas, avalia também a sobrecarga imposta pelo uso de um EMS (no caso, o HoLMES) intermediando a comunicação entre um OSS e estas mesmas plataformas. Os indicadores de impacto dessa sobrecarga, além de suas causas, são evidenciados na Subseção 5.3.1.

5.3.1 Estudo de Caso: Sobrecarga de Tempo de Operação

No presente estudo de caso, a arquitetura proposta foi avaliada através do protótipo HoLMES na execução de operações típicas de gerenciamento de funções de rede virtualizadas. Com isso objetiva-se determinar a sobrecarga imposta no tempo de operação devido ao uso do EMS intermediando a comunicação entre um OSS e diferentes plataformas de execução de VNF. Para tanto, dois cenários são comparados: (i) OSS implementando uma interface de comunicação para cada uma das plataformas de execução de VNF testadas, requisitando operações de gerência diretamente aos seus MA; (ii) OSS implementando uma interface de comunicação única com o EMS, o qual faz a interface para as requisições por operações de gerência com as plataformas de execução de VNF. Três plataformas de execução de VNF foram testadas: Click-On-OSv (COO – interface HTTP) (Marcuzzo et al., 2017), COVEN (interface HTTP e via Socket TCP) (Fulber-Garcia et al., 2019a) e Leaf (interface HTTP) (Flauzino et al., 2020). Três operações providas de forma similar nessas plataformas foram executadas: (i) obter o estado de execução da plataforma, que é uma operação leve (*Get Status*); (ii) enviar uma função de rede para a plataforma, operação de transferência de dados (*Post NF*); e (iii) configurar e iniciar uma função de rede na plataforma, operação de computação intensiva (*Post Configure and Start*). O tempo de operação é medido a partir da ótica do OSS, sendo a contagem iniciada no momento do envio de uma requisição e cessada no momento do recebimento de sua resposta.

Duas máquinas foram utilizadas na realização dos experimentos, uma para a instanciação dos elementos operacionais de gerência e outra para a instanciação das funções de rede virtualizadas. A máquina de gerência hospeda tanto o OSS que emite requisições, quanto uma instância do HoLMES. Essa máquina conta com um processador Intel Core I3 4010-U 1.9GHz, 8GB de memória RAM DDR3 e Ubuntu 16.04. A segunda máquina é dedicada à instanciação e execução das funções de rede virtualizadas, sendo a virtualização de cada plataforma de execução de VNF mantida somente durante a execução de seus respectivos experimentos. Essa máquina conta com processador Intel Core I5-3330 3.0GHz, 8GB de memória RAM DDR3 e Ubuntu 16.04. A virtualização das funções de rede foi feita através do hipervisor KVM. Todos os sistemas utilizados estavam conectados a uma rede local GbE (*Gigabit Ethernet*) e contavam com um IP dedicado. O ambiente de experimentação é ilustrado de forma simplificada na Figura 5.5. Cada

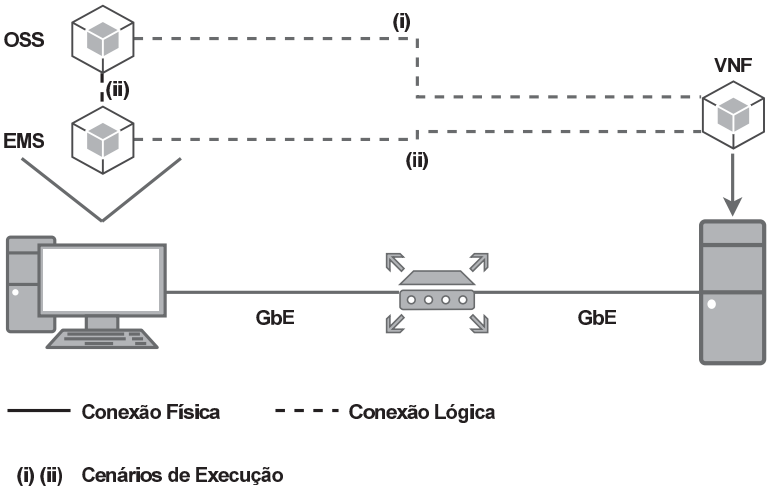


Figura 5.5: Ambiente de Execução para Testes de EMS

experimento foi executado 150 vezes, sendo 10% dos resultados extremos (*i.e.*, melhores e piores) descartados, totalizando o processamento útil de 120 execuções e alcançando um intervalo de confiança dos resultados de 95%.

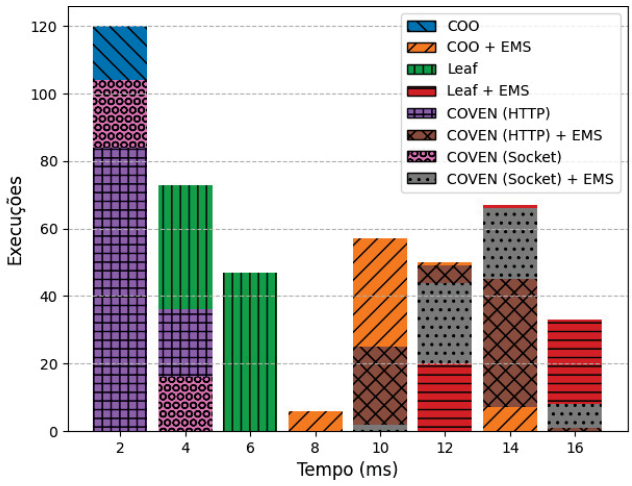


Figura 5.6: Distribuição - *Get Status*

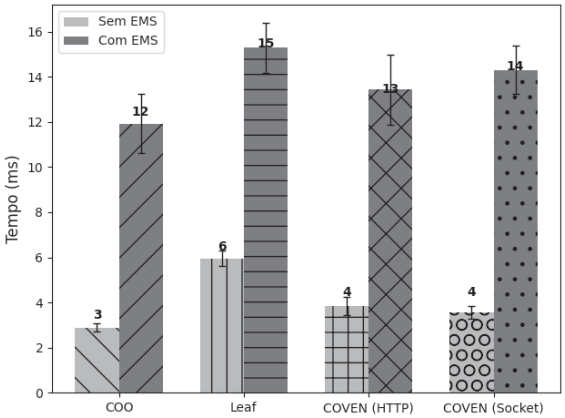


Figura 5.7: Média - *Get Status*

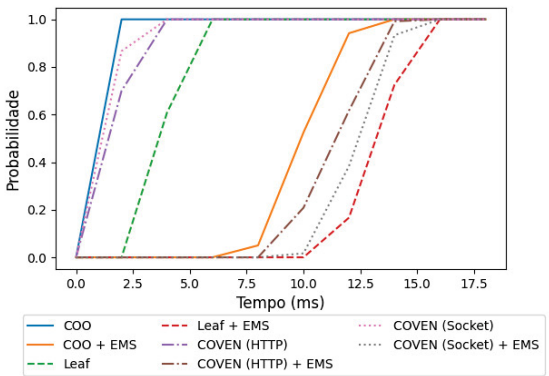


Figura 5.8: CPDF - *Get Status*

O primeiro experimento recupera o estado das plataformas de execução de VNF como uma cadeia de caracteres. Essa é uma operação de monitoramento considerada leve, pois não envolve transferência de arquivos e o tempo de processamento para sua efetivação é desprezível. A distribuição por execução, a média e o Diagrama de Fluxo de Probabilidade Acumulada (*Cummulative Probability Flow Diagram* - CPDF) do tempo dessa operação nas plataformas testadas são apresentados, respectivamente, nas Figuras 5.6, 5.7 e 5.8. Nesse experimento, a sobrecarga no tempo de execução originada pela utilização do HoLMES variou entre 150% (Leaf) e 300% (COO), sendo as sobrecargas intermediárias iguais a 225% e 250% para, respectivamente, COVEN-HTTP e COVEN-Socket. A maior discrepância bruta de resultados, contudo, entre o cenário (i) - sem EMS - em relação ao cenário (ii) - com EMS – ocorreu durante a testagem do COVEN-Socket (10ms). Nesse caso, a maior sobrecarga é atribuída, além da presença da indireção de requisições para o HoLMES, à necessidade do processamento de um protocolo de camada sete (HTTP) no EMS utilizado, o que é dispensado no primeiro cenário devido à comunicação ser realizada exclusivamente via *sockets* TCP. A sobrecarga no tempo de operação nas demais plataformas deve-se exclusivamente à indireção decorrente do EMS. Assim, a diferença no percentual médio do tempo de operação é atribuída às características operacionais das plataformas testadas, refletindo também na complexidade dos *drivers* de interfaceamento usados pelo HoLMES.

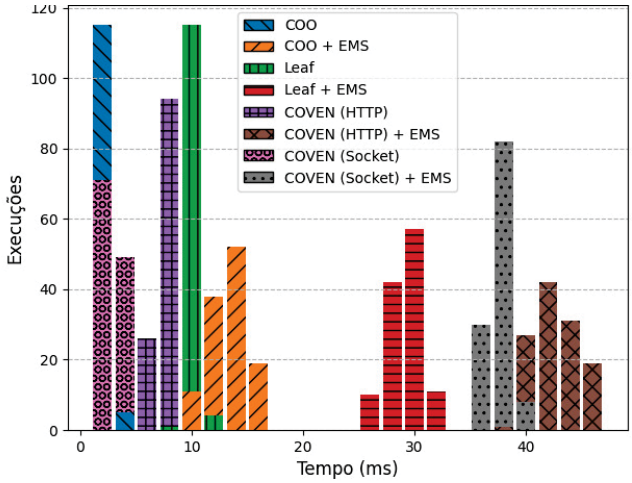


Figura 5.9: Distribuição - *Post NF*

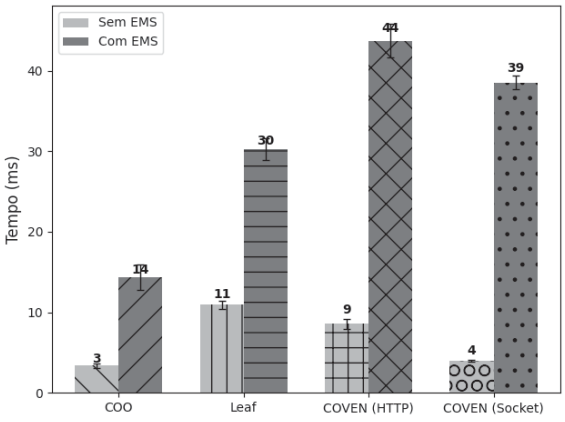


Figura 5.10: Média - *Post NF*

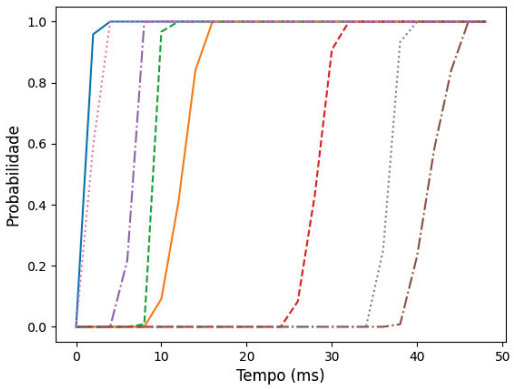


Figura 5.11: CPDF - *Post NF*

No segundo experimento são enviadas funções de rede para as plataformas de execução de VNF. Essa operação de configuração é dedicada à transferência de dados e arquivos. Os resultados relacionados à distribuição por execução, média e CPDF do tempo de operação são exibidos, em ordem, nas Figuras 5.9, 5.10 e 5.11. A média da sobrecarga do tempo de operação decorrente do HoLMES variou entre 367% (COO) e 875% (COVEN-Socket), sendo os resultados intermediários iguais a 173% para o Leaf e 389% para o COVEN-HTTP. A maior variação do tempo médio de operação em relação ao primeiro experimento (*i.e.*, *Get Status*) acontece devido à necessidade da transferência de arquivos das funções de rede para um intermediário (*i.e.*, o EMS), que realiza então o envio para a plataforma destino. As funções de rede enviadas para cada plataforma testada diferem entre si, dado que estas plataformas apresentam requisitos, capacidades e características específicas. Para o COO, a função de rede é um simples encaminhador de tráfego com 352 octetos; para o Leaf, a função de rede encontra-se pré-instalada na plataforma (servidor Apache2) e o pacote de configuração enviado contém 2797 octetos; para o COVEN, a função de rede enviada consiste em um encaminhador de tráfego com dois componentes, sendo o pacote enviado contando com 9287 octetos. A diferença na quantidade de dados transmitida impacta ativamente na variação da sobrecarga produzida pela utilização do EMS. Por fim, no caso do COVEN-Socket, a maior sobrecarga é gerada por motivos análogos aos do primeiro experimento, estes agravados pela necessidade de transferência de dados. Destaca-se que, mesmo no pior caso de sobrecarga (875%), o tempo extra absoluto não superou 35 milissegundos.

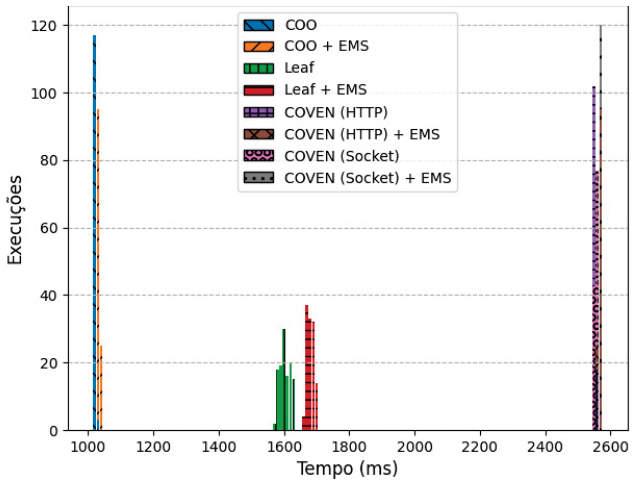


Figura 5.12: Distribuição - *Post Start*

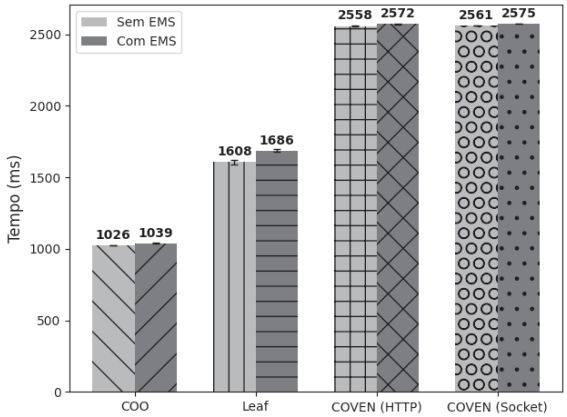


Figura 5.13: Média - *Post Start*

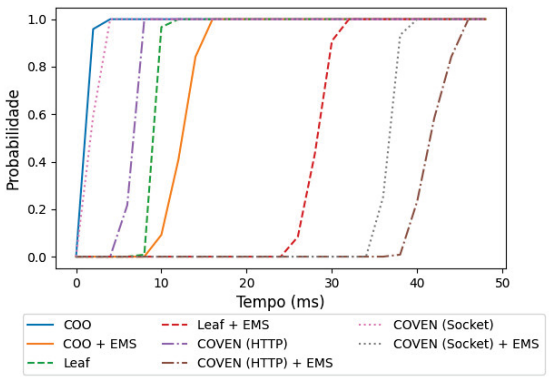


Figura 5.14: CPDF - *Post Start*

O terceiro experimento realiza a configuração e inicialização das funções de rede previamente enviadas para as plataformas de execução de VNF. Essas operações de configuração resultam em situações de computação intensiva internamente às plataformas. A distribuição por execução, média e CPDF do tempo de operação nos cenários testados são expostas nas Figuras 5.12, 5.13 e 5.14, respectivamente. A média de sobrecarga no tempo de operação proveniente do HoLMES variou entre 0,6% (COVEN-HTTP e COVEN-Socket) e 4,8% (Leaf), sendo o resultado intermediário igual a 1,3% (COO). Esse experimento apresenta a menor sobrecarga relativa entre a utilização ou não de um EMS intermediando a comunicação. Tal fenômeno ocorre devido às características de execução das operações de configuração e inicialização de funções de rede: requisição leve (sem transferência de dados), resultando em baixo tempo de transmissão, e operação custosa, requerendo maior tempo de processamento internamente às plataformas de execução de VNF. A plataforma Leaf representa o teto da variação de sobrecarga. Esse comportamento decorre da necessidade de duas chamadas HTTP para iniciar uma função de rede na plataforma, ocasionando uma sobrecarga acumulada. Em contrapartida, as demais plataformas exigem apenas uma chamada HTTP ou uma conexão via *socket* TCP para executar uma operação de inicialização equivalente, culminando nas menores sobrecargas observadas. É importante ressaltar, por fim, que o tempo absoluto elevado resultante da experimentação com a plataforma COVEN decorre desta ser um sistema mais complexo, com uma arquitetura modular e uma variedade de características que são aplicadas em tempo de inicialização, além do suporte à instanciação de vários componentes formando suas funções de rede.

Os resultados dos experimentos demonstram que a utilização de um EMS (em particular, do HoLMES) acresce o tempo total de operação para a aplicação de operações de gerência em plataformas de execução de VNF. Porém, o HoLMES foi capaz de abstrair a comunicação com diferentes plataformas, tratando a heterogeneidade dos agentes de gerência e suas respectivas tecnologias. Esse tratamento permite que, através uma interface única e padronizada, o OSS consiga comunicar-se com uma variedade de plataformas, facilitando seu desenvolvimento e usabilidade. Em relação à variação da sobrecarga de tempo imposta pelo uso do HoLMES, esta pode representar, percentualmente, maiores (875%) ou menores (0,6%) impactos conforme as características da operação requisitada e da plataforma de execução de VNF destino. Entretanto, mesmo com casos de percentuais elevados, o acréscimo de tempo não superou 78 milissegundos, sendo uma variação bruta aceitável para uma miríade de cenários sem requisitos estritos de tempo.

5.4 SUMARIZAÇÃO E DISCUSSÃO

O gerenciamento de funções de rede virtualizadas é um dos desafios do paradigma NFV. O EMS é o elemento responsável pela requisição de operações de gerência diretamente nas instâncias de funções de rede virtualizadas. Esse elemento, entretanto, é parcamente abordado na literatura recente, sendo frequentemente ignorado ou, quando utilizado, desenvolvido de forma ingênua, sem considerar nenhuma orientação arquitetônica. Nesse contexto, as implementações de EMS existentes são restritivas e com aplicabilidade limitada, contribuindo de forma periférica para a evolução do gerenciamento de NFV. Para aprimorar o desenvolvimento de soluções de EMS e aumentar a participação do mesmo no controle do ciclo de vida das funções de rede virtualizadas, uma arquitetura padronizada que oriente a implementação do mesmo é necessária. Através dessa arquitetura, é possível definir módulos operacionais pouco acoplados, flexíveis e atualizáveis, provendo longevidade às soluções desenvolvidas e previsibilidade em relação às suas capacidades operacionais.

Nesse contexto, este capítulo apresentou uma arquitetura interna para o elemento de EMS totalmente compatível com a arquitetura de referência NFV. A arquitetura proposta consiste em uma série de módulos operacionais independentes que se comunicam através de interfaces e pontos de acesso padronizados, permitindo assim abstrair e intermediar de forma holística a comunicação entre VNFM, OSS/BSS e plataformas de execução de VNF. Essa arquitetura é genérica, podendo ser aplicada no desenvolvimento de diferentes categorias e classes de EMS. Os módulos operacionais previstos na arquitetura de EMS são: (i) EMIB - centraliza as informações de gerência do EMS; (ii) AS - responsável pela comunicação do EMS com elementos e usuários externos, trata o encaminhamento de requisições e resultados, assim como a autenticação dos usuários; (iii) IR - dedicado à condução de dados entre os módulos operacionais da arquitetura; (iv) MS - incumbido de gerenciar monitores de instâncias de VNF, assim como de enviar mensagens e alertas gerados por eles para seus respectivos assinantes; (v) VS - encarregado da comunicação entre o EMS e diferentes instâncias de VNF, requerendo operações de FCAPS às mesmas; e (vi) CM - destinado a inicializar e atualizar os demais módulos operacionais da arquitetura.

A arquitetura de EMS proposta foi implementada como um protótipo categorizado como MANO (independente), externo, restrito e guarda-chuva. O protótipo, chamado *Holistic, Lightweight and Malleable EMS Solution* (HoLMES), é fornecido como um pacote de instalação ou como uma imagem de máquina virtual Ubuntu Cloud. Todos os módulos operacionais foram desenvolvidos com a linguagem de programação Python 3. O AS disponibiliza as operações protocolares da interface Ve-Vnfm-em, além de operações restritas a plataformas de execução de VNF específicas, através de uma interface HTTP (OA e AIf). A comunicação com o NFV-MANO e OSS/BSS variados é executada por um sistema de *drivers* que podem ser alocados sob demanda e em tempo de execução no AIf. Ainda, usuários e seus privilégios são autenticados/conferidos no AA. O IR é implementado como um direcionador de tráfego interno que marca e armazena referências das requisições recebidas. O MS gerencia processos que executam agentes de monitoramento, estes realizam operações nas instâncias de VNF e informam resultados relevantes aos seus assinantes através de um MIf, este implementado de forma particular para cada agente. O VS propicia a comunicação e a requisição de operações de FCAPS para instâncias de VNF que executam sob plataformas variadas. Para isso, o VS assume um sistema de *drivers* similar ao do AS, mas destinado a prover compatibilidade de comunicação entre o EMS e as plataformas de execução de VNF (em particular, com os seus MA). A EMIB foi criada como um banco de dados relacional SQLite que armazena informações essências para a configuração e manutenção de uma instância do HoLMES. Exemplos de tais informações são credenciais de usuários, instâncias de VNF, *drivers* de compatibilidade, entre outros. Por fim, o CM tem acesso irrestrito à EMIB, devendo aplicar as configurações nela contidas em todos os módulos operacionais, além de atualizar os mesmos (quando requisitado) durante o ciclo de vida de uma instância do HoLMES. Para isso, garante-se ao CM acesso direto e com privilégio pleno aos demais módulos operacionais.

Um estudo de caso foi conduzido com três experimentos, cada um com o objetivo de analisar uma operação típica do gerenciamento do ciclo de vida de instâncias de VNF: (i) obtenção do estado da plataforma; (ii) envio/instalação de um VNFP; e (iii) configuração/inicialização de NF. No estudo de caso, três plataformas de execução de VNF com MA heterogêneos foram consideradas: Click-On-OSv (Marcuzzo et al., 2017), Leaf (Flauzino et al., 2020) e COVEN (Fulber-Garcia et al., 2019b). Cada experimento tem dois cenários de execução: (i) um OSS requisitando a operação de gerência diretamente à instância virtualizada através do protocolo particular de sua respectiva plataforma; e (ii) um OSS requisitando a operação de gerência ao EMS por um protocolo padrão, sendo que o último então atua como intermediário na comunicação

com a instância de VNF objetivada de acordo com a sua respectiva plataforma de execução. De forma geral, os resultados obtidos demonstraram que a utilização de um EMS, em particular do HoLMES, acresce o tempo de execução de uma operação de FCAPS nas instâncias de VNF, mas padroniza com sucesso a requisição de operações que apresentam uma mesma finalidade, porém são providas de forma diferente nas plataformas de execução de VNF testadas (em relação aos seus protocolos e tecnologias). A sobrecarga causada pelo HoLMES varia conforme a tecnologia de comunicação utilizada, a operação requisitada e a complexidade do *driver* (VS) empregado. Entretanto, em valores absolutos, a sobrecarga analisada nos experimentos ficou contida entre 9 e 78 milissegundos, sendo esta admissível em diversas circunstâncias no contexto de um ambiente NFV (*e.g.*, inicialização de funções de rede em processos de implantação de serviços, que podem chegar à escala de minutos para sua execução completa).

Em suma, os experimentos validaram a capacidade dos módulos da arquitetura de EMS proposta em gerenciar de forma holística múltiplas e diferentes plataformas de execução de VNF. Dessa forma, é possível criar ambientes NFV em concordância com a arquitetura ETSI que suportam plataformas de execução de VNF independentes e heterogêneas. Além disso, a abrangência e flexibilidade de gerenciamento constatada também aprimora a portabilidade do ambiente NFV, um dos requisitos da ETSI. Com a solução proposta é desnecessário adaptar internamente sistemas de NFV-MANO, OSS/BSS ou plataformas de execução de VNF quando elementos operacionais migram de uma infraestrutura a outra, concentrando as adaptações (quando necessárias) em um único elemento operacional: o EMS.

6 CENÁRIOS DE APLICAÇÃO DAS ARQUITETURAS DE VNF E EMS

Este capítulo apresenta um conjunto de cenários de aplicação das arquiteturas dos elementos de VNF e EMS apresentadas nos Capítulos 4 e 5. Mesmo sendo de naturezas distintas, os cenários de aplicação considerados se beneficiam tanto dos módulos operacionais e interfaces de comunicação identificados nas arquiteturas de EMS e VNF, quanto dos modelos operacionais dos protótipos COVEN e HoLMES. A seguir, a Seção 6.1 discute as capacidades de monitoramento de funções e serviços de rede oportunizadas pelas arquiteturas propostas; a Seção 6.2 discute o compartilhamento de instâncias de VNF entre diferentes clientes; a Seção 6.3 discute o desenvolvimento de soluções para emulação e testes de ambientes e serviços NFV. Por fim, a Seção 6.4 sumariza e discute de forma abrangente o capítulo.

6.1 MONITORAMENTO ABRANGENTE DE VNF

De maneira geral, processos de monitoramento de *software* e *hardware* podem ser implementados em diversos níveis de granularidade, abrangência e computabilidade (Stucki et al., 2021). A granularidade diz respeito a fonte das informações acessadas e recuperadas pelo monitoramento. Em um extremo há o denominado monitoramento de caixa-preta, baseado em métricas genéricas independentes da aplicação ou do sistema específico sendo monitorado. No outro extremo, o monitoramento de caixa-branca é projetado para uma aplicação e/ou sistema em particular. Entre esses extremos, existem monitoramentos chamados caixa-cinza que são intermediários. O monitoramento de caixa-cinza considera alguns aspectos dos sistemas sendo monitorados, mas descartam métricas que englobam informações muito específicas de tais sistemas. Assim, o monitoramento caixa-cinza tipicamente foca no sistema e não nas aplicações sendo executadas neste. Por exemplo, uma estratégia de monitoramento de caixa-cinza obtém informações específicas de uma VNF a partir do sistema hospedeiro de sua plataforma de execução.

A abrangência refere-se à capacidade de monitorar apenas métricas isoladas ou de interpretar correlações entre múltiplas métricas simultaneamente. Tipicamente, o processamento simultâneo de múltiplas métricas permite que diferentes propriedades de um objeto gerenciado sejam levadas em considerações para um diagnóstico mais apurado. Uma correlação de múltiplas métricas é chamada de hiperpropriedade (Finkbeiner et al., 2019). Ressalta-se que uma hiperpropriedade pode ser criada através de métricas provindas tanto de monitoramentos de caixa-preta quanto de caixa-branca. Finalmente, a computabilidade refere-se às capacidades e limitações dos próprios monitores, podendo estes apenas fornecer as métricas monitoradas para serem interpretadas por um agente humano ou processá-las e interpretá-las previamente, retornando conclusões de mais alto nível sobre o estado de um objeto monitorado (*e.g.*, (Mandal et al., 2020)). A Figura 6.1 ilustra granularidade, abrangência e computabilidade em dimensões onde podem ser localizados os monitores. A proximidade de uma estratégia de monitoramento com o ponto A (origem) indica características de caixa-preta, baixa abrangência e baixa capacidade de processamento e interpretação de métricas. Já a proximidade do ponto B indica um monitor com características de caixa-branca, grande abrangência e processamento/interpretação sofisticado das métricas disponíveis.

Atualmente, existem múltiplos contextos de monitoramento de NFV. Um contexto específico onde frequentemente técnicas de monitoramento são empregadas é a detecção de anomalias e de gargalos de execução em serviços de rede. Em tal cenário, ferramentas como

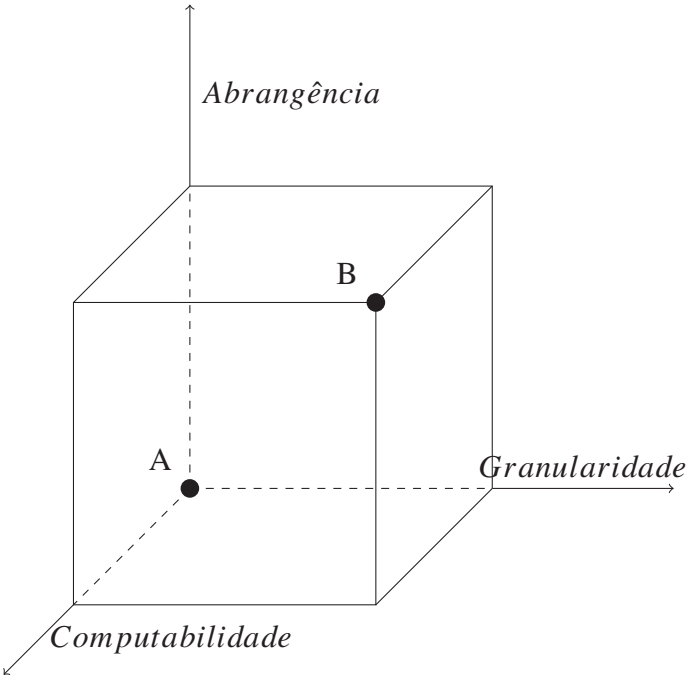


Figura 6.1: Representação em Dimensões das Características de Monitores

NFVPerf (Naik et al., 2016) e o detector de anomalias proposto em (Sauvanaud et al., 2016) usam estratégias de caixa-preta e caixa-cinza para monitorar métricas genéricas de rede, memória e processamento das máquinas virtuais para detectar pontos de sobrecarga em um serviço virtualizado. No caso do NFVPerf o monitoramento é realizado exclusivamente através do hipervisor que suporta o ambiente virtualizado. A principal vantagem dessa estratégia consiste na compatibilidade imediata com qualquer VNF. Porém, apesar de ser eficiente em apontar pontos de sobrecarga isolados no serviço, essa estratégia é pouco eficiente em diagnosticar os motivos da sobrecarga. Por exemplo, é impossível indicar, em uma função de rede específica composta por vários componentes, aquele que está sobrecarregado. O detector de anomalias em (Sauvanaud et al., 2016) aparelha o sistema hospedeiro executado em uma máquina virtual para realizar as medições das métricas necessárias. Com isso, essas medições podem ser classificadas como de caixa-cinza. Diferente do NFVPerf, a estratégia permite averiguar sobrecargas em diferentes componentes de funções de rede. Entretanto, essa estratégia exige a preparação prévia e individual das plataformas de execução de VNF para permitirem a coleta das métricas correspondentes. Nenhuma das duas ferramentas permite a previsão da ocorrência de novos pontos de sobrecarga uma vez solucionado o atual. Por exemplo, essas ferramentas são incapazes de determinar se reconfigurando adequadamente uma VNF sobrecarregada o bom funcionamento das VNF posteriores em uma topologia de serviço será garantido ou não.

Uma terceira solução de monitoramento proposta em (Sanchoyerto et al., 2019) usa sistemas existentes em nuvens OpenStack para monitorar métricas relacionadas ao *hardware* físico e virtualizado (*i.e.*, máquinas virtuais). Essa estratégia se beneficia de operações do OpenStack para disparar ações automáticas quando cenários preestabelecidos são detectados no monitoramento. Um exemplo de ação é a escala automática de instâncias de VNF. Além disso, incluir o *hardware* físico no monitoramento facilita a detecção de gargalos que não necessariamente são diretamente relacionados a um serviço específico, mas ao conjunto de serviços em execução na máquina física. Um exemplo desse cenário é o estouro da pilha de rede da máquina física devido a pequenas sobrecargas de tráfego em todos os serviços executando na mesma, gerando uma grande sobrecarga acumulada. Entretanto, similar aos outros monitores,

este não prevê as consequências da resolução de gargalos e não pode recuperar métricas internas às instâncias de VNF.

Também, normalmente, plataformas de execução de VNF existentes ou não oferecem agentes de gerência nativos (Martins et al., 2014) (OPNFV, 2021) (Intel, 2021) (Hwang et al., 2015) (Zhang et al., 2016b) (Zheng et al., 2018) (Gallo et al., 2018) (Marku et al., 2019), ou o fazem limitadamente quanto ao monitoramento possibilitado por eles, promovendo a recuperação de métricas gerais e relatórios de erros de execução (Marcuzzo et al., 2017) (Flauzino et al., 2020). Da mesma forma, as soluções de EMS existentes, em geral, não proveem suporte (Canonical, 2021a) (ETSI, 2021), ou fornecem suporte limitado (*i.e.*, requer aparelhamento do sistema hospedeiro de uma plataforma de execução de VNF) (Berlin, 2021) para acesso às métricas internas às instâncias de VNF. A exceção é o EMS presente no ClouStack Vines (Flauzino et al., 2020), o qual permite que extensões de protocolos sejam feitas para estabelecer a comunicação com plataformas de execução de VNF, habilitando a obtenção das métricas de monitoramento quando providas pelas mesmas.

Via de regra, o monitoramento de funções de rede virtualizadas permite apenas monitorar o consumo de recursos computacionais e métricas de caixa-preta. Essa limitação decorre da não disponibilização de métricas relacionadas à execução das próprias funções de rede (NF), seja pela não existência de agentes de monitoramento específicos, ou seja pela falta de suporte a estes mesmos agentes por parte do MA das plataformas de execução de VNF. Um fenômeno semelhante acontece em relação às soluções de EMS, que empregam poucos esforços para adequarem-se aos agentes de gerência de plataformas de execução de VNF heterogêneas, assim inviabilizando que as métricas de monitoramento, quando disponíveis, sejam acessadas pelo NFV-MANO ou OSS/BSS através de um protocolo de comunicação padronizado (*i.e.*, Ve-Vnfm-em). As arquiteturas propostas para plataformas de execução de VNF e para o elemento de EMS apresentam soluções para ambos os desafios citados. Em relação às plataformas de execução de VNF, a arquitetura proposta prevê a criação de um agente de gerência (MA) robusto, o qual suporta agentes estendidos de monitoramento (EA) programados pelos desenvolvedores das funções de rede. Desse modo, monitoramentos específicos podem ser programados e executados em conjunto à função de rede, permitindo o acesso a métricas de caixa-branca em tempo de execução da mesma. Adicionalmente, a arquitetura de EMS proposta propicia a criação de soluções restritas, porém maleáveis, estas concretizadas através de um VS configurável sob demanda. Assim, é possível incluir a recuperação de métricas de caixa-branca ao executar operações protocolares da interface Ve-Vnfm-em, como a $GET/vii/indicators$ (NFVISG, 2020c).

6.1.1 Experimentação e Resultados

Para verificar a viabilidade da realização dos processos de monitoramento descritos nesta seção, realizamos uma série de experimentos utilizando tanto a plataforma de execução de VNF COVEN, quanto a solução EMS HoLMES para recuperar e disponibilizar métricas com diferentes granularidades. O cenário de experimentação consiste em uma função de rede de filtragem de pacotes sendo executada por uma instância da plataforma COVEN. O primeiro componente de tal função é responsável pela filtragem de pacotes em camada dois, encaminhando tráfego originário de endereços MAC considerados confiáveis. O segundo componente realiza a filtragem de pacotes em camada três, encaminhando o tráfego proveniente de endereços IP autorizados. Sendo assim, quadros e pacotes com endereços de origem considerados maliciosos ou desconhecidos são imediatamente descartados pela função de rede. O filtro de pacotes adotado contempla uma série de agentes estendidos disponibilizados para ambos componentes. Esses agentes conseguem medir e prover as métricas descritas a seguir:

- ***Recv Packets***: número absoluto de pacotes recebidos pelo componente. Esta métrica pode ser considerada de caixa-branca, baixa abrangência e baixa computabilidade.
- ***Recv Bytes***: número absoluto de bytes recebidos pelo componente. Esta métrica pode ser considerada de caixa-branca, baixa abrangência e baixa computabilidade.
- ***Frwd Packets***: número absoluto de pacotes encaminhados pelo componente. Esta métrica pode ser considerada de caixa-branca, baixa abrangência e baixa computabilidade.
- ***Frwd Bytes***: número absoluto de bytes encaminhados pelo componente. Esta métrica pode ser considerada de caixa-branca, baixa abrangência e baixa computabilidade.
- ***Frwd Packets Rate***: relação percentual entre pacotes encaminhados e pacotes recebidos pelo componente. Esta métrica pode ser considerada de caixa-branca, alta abrangência (correlaciona as métricas de pacotes recebidos e encaminhados) e baixa computabilidade.

Além das métricas de caixa branca intrinsecamente relacionadas à função de rede em execução na instância da plataforma COVEN, também existem as funções de monitoramento e métricas de caixa-preta e caixa-cinza providas pela própria plataforma de execução de VNF. Um subconjunto de tais funções e métricas foi empegado no contexto do experimento realizado, sendo estas descritas a seguir:

- ***Get Status***: métrica que verifica a disponibilidade de uma instância da plataforma COVEN. Esta métrica pode ser considerada de caixa-preta, baixa abrangência e baixa computabilidade.
- ***Get List***: função de monitoramento que retorna uma lista de métricas recuperáveis considerando o estado atual da plataforma. Esta função informa, além das métricas nativas da plataforma, também as métricas disponibilizadas por agentes estendidos das funções em execução. Desta forma, em vista das características dos dados providos por esta função de monitoramento, ela pode ser considerada uma métrica de caixa-cinza, baixa abrangência e baixa computabilidade.

Para o gerenciamento da instância da plataforma COVEN executando a função de rede previamente descrita, utilizamos em nosso cenário de teste uma instância do EMS HoLMES. O *driver* que permite a comunicação entre o HoLMES e COVEN foi estendido para recuperar, além das métricas providas pelo MA da plataforma de execução de VNF, recuperar, calcular e prover uma nova métrica, chamada *Get Dropping Rate*.

A métrica *Get Dropping Rate* informa o percentual de descarte de pacotes de cada componentes da função de rede. Para fazer isso, o EMS requisita duas métricas ao MA da instância da plataforma COVEN, sendo estas requisições efetuadas para cada um dos componentes: *Recv Packets* e *Frwd Packets*. Após a recuperação das métricas, o próprio EMS calcula a taxa de descarte de cada componente, retornando as mesmas como o resultado da função de monitoramento. Sendo assim, esta métrica pode ser considerada de caixa-branca, alta abrangência (correlaciona duas métricas), e alta computabilidade (necessita de comunicação em rede para ser calculada).

Tanto a função de rede, quanto o EMS foram instanciados em uma máquina com Core I3 4010, com 8GB de memória RAM DDR3, executando o sistema operacional Ubuntu 16.04 e utilizando virtualização baseada em processos. O sistema hospedeiro cumpriu o papel de cliente, disparando as requisições pelas métricas descritas ao EMS. Para cada requisição, medimos o tempo de resposta entre o envio da mesma e o recebimento do resultado. Os testes foram

executados 30 vezes para cada métrica, gerando um intervalo de confiança dos resultados de 95%. Finalmente, como o tempo de resposta para a recuperação das métricas de *Recv Packets*, *Recv Bytes*, *Frwd Packets* e *Frwd Bytes* são muito semelhantes, para facilitar a visualização, os gráficos exibem apenas os resultados relacionados à operação *Frwd Packets*.

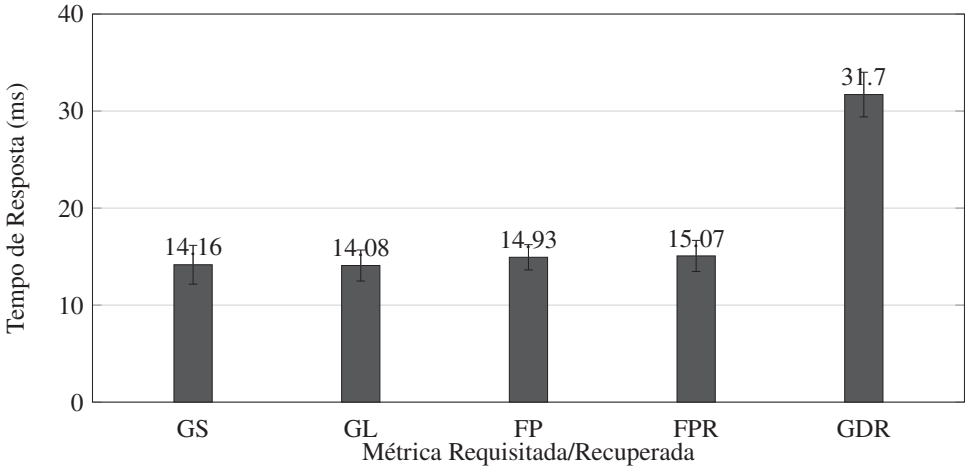


Figura 6.2: Tempo de Resposta para a Obtenção de Diferentes Métricas

A Figura 6.2 exibe os resultados para os testes realizados no cenário previamente descrito. O eixo X do gráfico determina a operação testada, sendo GS o acrônimo para *Get Status*, GL para *Get List*, FP para *Frwd Packets*, FPR para *Frwd Packets Rate* e GDR para *Get Dropping Rate*. As barras, em conjunto com o eixo Y, apresentam a média do tempo de resposta, em milissegundos, para a recuperação das métricas. As barras de erro indicam o desvio padrão correspondente.

Como é possível observar no gráfico, o processo de recuperação das métricas GS, GL, FP e FPR apresentam tempos de resposta muito próximos. Esse fenômeno ocorre pois, mesmo sendo métricas de natureza diferente em relação à granularidade, todas apresentam baixa computabilidade, sendo o tempo de transmissão dos dados dominante em relação ao tempo de computação/cálculo destes na plataforma de execução de VNF. Vale ressaltar, entretanto, que dentre as métricas de baixa computabilidade, apenas FPR é considerada uma hiperpropriedade, uma vez que correlaciona outras duas métricas (*Recv Packets* e *Frwd Packets*). Porém, o cálculo feito para a geração do dado de FPR é trivial, consistindo apenas em um teste condicional e uma expressão aritmética (baixa computabilidade), não impactando significativamente no tempo de resposta.

A única métrica considerada de alta computabilidade, a GDR, apresentou um tempo de resposta consideravelmente maior quando comparada às demais métricas (110% a 125% de acréscimo). Tal diferença decorre, principalmente, do fato desta métrica ser uma hiperpropriedade calculada no contexto do EMS, e não na própria instância virtual executando a função de rede. Sendo assim, o EMS deve realizar um conjunto de processos de recuperação de métricas providas pela VNF para, só então, conseguir calcular o GDR. Com isso, apesar de o cálculo do GDR ser muito semelhante ao realizado para o FPR, a sobrecarga de requisições realizadas gera o aumento verificado para o tempo de resposta.

O experimento realizado objetivou apresentar múltiplas possibilidades de monitoramento a partir da adoção dos módulos das arquiteturas de VNF e EMS propostas, em especial o MA (VNF), EA (VNF) e VNS (EMS). Através desses módulos, é possível fornecer, recuperar e calcular métricas de monitoramento distintas, incluídas em diferentes categorias de granularidade, abrangência e computabilidade. Finalmente, é importante destacar que, a partir das métricas

fornecidas nativamente pela plataforma de execução de VNF, é viável programar novas métricas e funções de monitoramento a serem fornecidas exclusivamente pelo EMS. Para isso, apenas modificações no *driver* da plataforma de execução de VNF em uso são necessárias, não sendo imposta nenhuma alteração da plataforma ou da função de rede nela executada.

6.2 COMPARTILHAMENTO DE INSTÂNCIAS DE VNF ENTRE CLIENTES

O paradigma NFV demonstra grande potencial para ser explorado como uma oportunidade de mercado, provendo funções e serviços de rede sob demanda. Para que esse potencial seja devidamente aproveitado, modelos de venda/aluguel dos elementos virtualizados de rede precisam ser investigados. Em trabalhos recentes, *marketplaces*, como T-NOVA (Xilouris et al., 2015) e FENDE (Bondan et al., 2019), foram criados para oferecer, intermediar a negociação e hospedar/executar funções e serviços de rede em suas infraestruturas, usando modelos de pagamento por produto ou por uso. Outras propostas relativas à negociação em NFV sugerem a oferta de funções, serviços e recursos computacionais virtualizados por leilões (Gu et al., 2016) (Zhang et al., 2017). Todos esses esforços culminaram em um novo modelo de negócios, chamado de *NFV-as-a-Service* (NFVaaS) (Xilouris et al., 2015), que tem como seu principal produto componentes para o núcleo da rede. Tal modelo, como outros *as-a-service* (*e.g.* *Software-as-a-Service*, *Platform-as-a-Service*), aglutinam parceiros de negócio (*e.g.*, provedores de conteúdo, aplicações e infraestrutura) que oferecem serviços para múltiplos clientes (*i.e.*, atores que contratam um serviço). As negociações no modelo *as-a-service* orientam-se por um conjunto de requisitos que visam acompanhar a dinamicidade e heterogeneidade do mercado e dos clientes (Li e Wei, 2014): (i) rápida adequação a mudanças nos serviços prestados; (ii) viabilidade de compartilhamento e reúso de recursos; (iii) garantia de um ambiente seguro que possibilite a colaboração plena entre provedores e clientes.

Apesar da tecnologia NFV ser naturalmente ágil e inovadora na prototipação, lançamento e manutenção de funções e serviços de rede, poucas investigações foram conduzidas abordando o compartilhamento destes entre diferentes clientes. Nesse cenário, trabalhos recentes tratam particularmente da implantação de serviços compartilhados (Hmaity et al., 2016) (Soualah et al., 2018), dos requisitos funcionais do compartilhamento de funções de rede análogas em topologias de serviço (Antonenko et al., 2018) e do compartilhamento de recursos computacionais em ambientes de virtualização (Ordonez-Lucena et al., 2017) (Savi et al., 2019). Além dessas aplicações, topologias de serviço com funções compartilháveis também foram definidas através da teoria de conjuntos (Fulber-Garcia et al., 2020a): seja G uma topologia de serviço com vértices V, representando funções de rede, e arestas E, compondo as conexões lógicas entre funções de rede, considere duas topologias de serviço $G_1 = (V_1, E_1)$ e $G_2 = (V_2, E_2)$ tal que $V_1 \cap V_2 \neq \emptyset$. Assim, diz-se que uma função de rede, isto é, um vértice identificado por v_i, é compartilhável entre G_1 e G_2 se $v_i \in V_1 \cap V_2$. A Figura 6.3 apresenta de forma simplificada a definição formal de topologia de serviço com compartilhamento, sendo a função denominada v_3 o ponto de compartilhável entre as topologias G_1 e G_2. Entretanto, a utilização prática das propostas relativas ao compartilhamento de instâncias de VNF entre clientes distintos são regularmente limitadas pela falta de ferramental adequado que as suportem em um ambiente virtualizado real.

Diversos desafios precisam ser abordados para preparar um ambiente NFV, em especial o domínio de trabalho de VNF, para suportar o compartilhamento de funções de rede e, ao mesmo tempo, garantir o cumprimento de políticas e acordos firmados entre cliente e provedores. O primeiro diz respeito à configuração das funções de rede compartilhadas para suportar necessidades de clientes heterogêneos que as utilizam em seus serviços. Uma possível solução

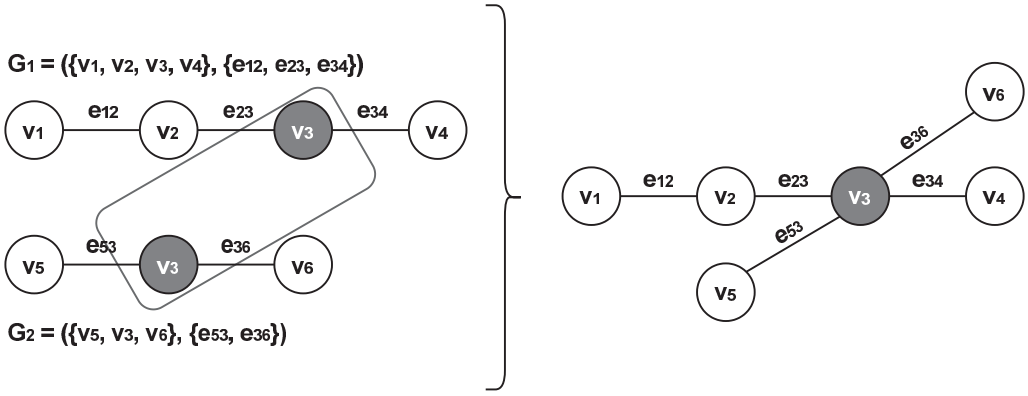


Figura 6.3: Definição Simplificada de Topologia de Serviço com Compartilhamento

consiste na definição de pontos de variação no código-fonte das funções de rede, permitindo a criação de diferentes linhas de produto com um mesmo *software* (da Cruz Marcuzzo et al., 2012). Desse modo, funções podem ser adaptadas para executarem uma linha de produto apropriada para acomodar requisitos funcionais e de desempenho de conjuntos específicos de clientes.

Outro desafio do compartilhamento de VNF é assegurar a privacidade de dados dos clientes. Nesse caso, níveis de isolamento de uma função de rede podem ser criados e o acesso dos clientes limitado apenas aos seus recursos particulares. Nota-se a adoção de tal estratégia no contexto de *Software-as-a-service* (Pearson e Charlesworth, 2009) (Masmoudi et al., 2014), sendo possível naturalmente a adaptar ao paradigma NFV. Nesse caso, a função de rede pode ser específica de cada cliente, mas a pilha de rede compartilhada, assim como a pilha de rede pode ser individualizada e a função compartilhada, ou, ainda, ambos serem compartilhados entre todos os clientes.

Um terceiro desafio consiste na segurança das funções compartilhadas. Além dos tradicionais ataques de negação de serviço, que em funções compartilhadas exigem a identificação do cliente alvejado após a detecção e mitigação do ataque, também existe a possibilidade de um ataque interno advindo de um cliente malicioso. Isso posto, é favorável que o compartilhamento de funções de rede, em especial naquelas que trafegam dados sensíveis, agrupe clientes distintos, porém que guardem determinado grau de confiança entre si, como ao pertencerem a uma mesma federação. Além dos desafios levantados, outros pontos de interesse relacionam-se à interoperabilidade, disponibilidade, escalabilidade, desempenho e confiabilidade das funções de rede compartilhadas.

Delineados os desafios, o próximo passo em direção a um modelo NFVaaS com suporte a funções e serviços compartilhados consiste da construção de um ferramental adequado para executá-los. Como exemplos de ferramentas necessárias estão plataformas de execução de VNF e soluções de EMS conscientes da pluralidade de clientes atendidos pelas instâncias de VNF em um ambiente NFV. Em particular, as plataformas de execução de VNF devem ser capazes de produzir as condições necessárias para a implementação de módulos específicos por cliente, individualizando e provendo privacidade a etapas do seu respectivo processamento de tráfego. Além disso, as plataformas devem contar com capacidades de gerenciamento das funções de rede executadas que considerem os clientes atendidos por elas.

A arquitetura de plataformas de execução de VNF proposta no Capítulo 4 apresenta módulos flexíveis o suficiente que suportam as características de execução do modelo NFVaaS. Por exemplo, o VNS pode ser aparelhado para destinar uma interface de rede virtual para um cliente específico, associando-lhe uma ferramenta de rede virtual particular (se necessário). O PPS, no que lhe concerne, pode executar funções de rede com múltiplos componentes

onde alguns são utilizados apenas por um conjunto dos clientes. Nesse caso, uma lógica de encaminhamento personalizável deve ser introduzida no ITF. O NSHP pode atuar como um módulo de compatibilidade, possibilitando a uma mesma plataforma atender clientes que transmitem/recebem tráfego de rede com e sem NSH. Em conclusão, funções de rede podem ser aparelhadas com agentes estendidos que coletam dados por fluxo ou por cliente, provendo os mesmos pelo MA. Da mesma forma, agentes estendidos podem funcionar como chaves para viabilizar a variação de características da função de rede, tornando as mudanças entre as linhas de produto possíveis realizáveis através do MA.

Soluções de EMS, no contexto de compartilhamento de funções de rede entre vários serviços/clientes, assumem importantes papéis no controle de acesso e na disponibilização de operações, principalmente para sistemas OSS/BSS que podem ser controlados por clientes que contratam serviços no modelo NFVaaS. Considerando a arquitetura de EMS apresentada no Capítulo 5, diversos módulos operacionais podem ser adaptados para suportarem a gerência de funções de rede compartilhadas. Por exemplo, a EMIB pode incluir tabelas de relacionamento entre usuários e instâncias de VNF, estas contendo informações como níveis de privilégio de acesso à operação e IPs ou rotas de comunicação exclusivas de um determinado cliente. Assim, o AS pode ser desenvolvido para considerar tais informações durante a autenticação e validação de uma requisição. Mais, uma das principais atribuições de um EMS, o interfaceamento entre o MA das plataformas de execução de VNF com o VNFM e OSS/BSS, apresenta-se como uma atividade ainda mais crítica. Isso ocorre pois, como destacado anteriormente, novas operações podem se fazer necessárias, sejam elas para monitoramento orientado a clientes ou para a modificação de linhas de produto através de pontos de variação nas funções de rede. Essas operações são identificadas e executas pelo VS e oferecidas para os usuários autorizados através do AS. Por fim, ressalta-se, apesar dos protótipos COVEN e HoLMES apresentarem algumas das funções de suporte a compartilhamento de instâncias de VNF descritas nesta seção, eles não foram nativamente desenvolvidos para operarem em tal cenário de execução, sendo necessárias modificações e novas funcionalidades para se adequarem completamente ao mesmo.

6.2.1 Experimentação e Resultados

De forma a analisar aspectos operacionais e de monitoramento de funções de rede compartilhadas, esta subseção apresenta um estudo de caso de compartilhamento entre múltiplos serviços de uma instância de função de rede executando um sistema de prevenção de intrusão baseado em assinaturas de tráfego.

Para executar o experimento, uma função de rede de análise, encaminhamento e descarte de tráfego de rede foi desenvolvida. Esta função opera buscando assinaturas consideradas maliciosas no conteúdo de pacotes de rede endereçados para serviços específicos; se uma assinatura é encontrada, o pacote é descartado; caso contrário, o pacote é encaminhado ao seu destinatário. Além de realizar o processamento de tráfego, também fornece agentes estendidos para seu monitoramento e configuração. Tais agentes são descritos a seguir:

- ***Post Service IP***: função que permite cadastrar um serviço, através de seus endereços IP, para ter seu tráfego de rede analisado pela função de rede. Sendo assim, após executada com sucesso, para todo o pacote cujo endereço IP de destino estiver contido na lista de serviços cadastrados, a função de rede procederá com o processo de busca por assinaturas maliciosas.

- ***Get NF Statistics***: função que retorna a quantidade total de pacotes e número absoluto de bytes recebidos pela função de rede, assim como o seu tempo total de operação.

- ***Get Services Statistics***: função que retorna a quantidade total de pacotes e número absoluto de bytes processados para cada um dos serviços cadastrados na função de rede, assim como o tempo total dispendido com as análises realizadas para estes.

A função de rede previamente apresentada foi instanciada em um elemento virtualizado do emulador NIEP (Tavares et al., 2018), sendo esta executada através da plataforma COVEN. Além da instância de VNF, o cenário de testes também conta com dois elementos virtualizados NIEP que atuam como clientes (chamados "Cliente #01" e "Cliente #02") de dois serviços distintos, cada um também provido em um elemento virtualizado e referidos como "Serviço #01" e "Serviço #02". Finalmente, um sexto elemento virtualizado é utilizado para hospedar uma instância do EMS HoLMES, empregada na gerência e monitoramento da plataforma COVEN e de sua função de rede em execução. Destaca-se que a função de rede processa o tráfego destinado aos dois serviços, o que a torna uma instância compartilhada. O cenário de testes descrito é ilustrado na Figura 6.4.

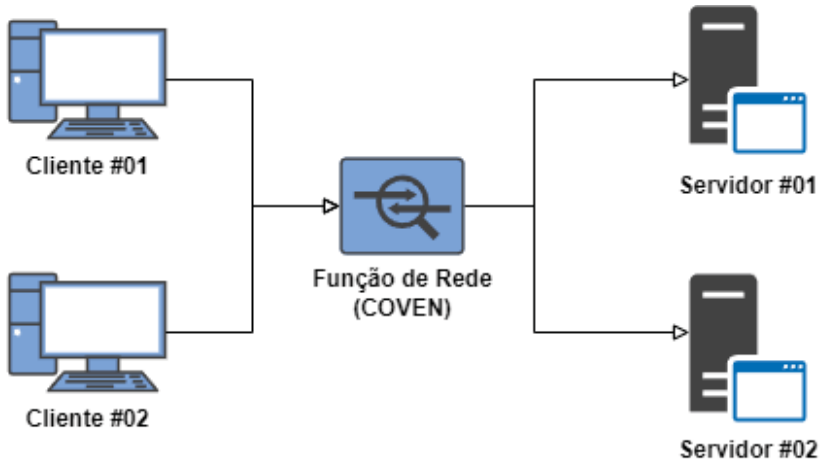


Figura 6.4: Cenário de Testes de Compartilhamento de VNF

Para os testes, o "Cliente #01" dedica-se a enviar requisições ao "Serviço #01", enquanto o "Cliente #02" envia requisições ao "Serviço #02". Existem quatro tipos de requisição: requisições de rápida rejeição (*Fast Reject* – FR), que consistem em pacotes com 50 bytes, sendo os quatro primeiros um padrão malicioso; requisições de rápida aceitação (*Fast Accept* – FA), sendo estas representadas por pacotes com 50 bytes de conteúdo sem padrões maliciosos; requisições de rejeição lenta (*Slow Reject* - SR), consistindo em pacotes com 500 bytes e com um padrão malicioso entre os bytes de número 396 e 400; e requisições de aceitação lenta (*Slow Accept* - SA), contemplando pacotes de 500 bytes de conteúdo sem padrões maliciosos.

O objetivo deste experimento é verificar se, considerando o envio de um mesmo número de requisições, é possível detectar sobrecargas de processamento relacionadas ao conteúdo das mesmas e relacionar tais sobrecargas a um ou mais serviços. Sendo assim, os dois serviços destinatários são cadastrados na função utilizando o agente estendido *Post Service IP*, e diversos cenários de teste são considerados, sempre enviando um total de 60 requisições por cliente/serviço, mas alternando o tipo de requisição enviada. Após a execução de cada cenários de teste, o agente estendido *Get Services Statistics* é requisitado para verificar o tempo total de operação da função de rede dedicado a cada um dos serviços considerados.

O experimento foi realizado em uma máquina com Core I3 4010, 8GB de memória RAM DDR3, executando o sistema operacional Ubuntu 16.04 e utilizando o emulador NIEP e virtualização baseada em processos. Os resultados obtidos são apresentados na Figura 6.5.

No gráfico, o eixo X indica o tipo de requisição originada pelo “Cliente #01” e enviada para o “Serviço #01” e o tipo de requisição originada pelo “Cliente #02” e enviada para o “Serviço #02”, respectivamente, antes e depois do sinal de soma (+). O eixo Y, no que lhe concerne, apresenta o tempo total (em milissegundos) dispendido pela função de rede para o processamento de tais requisições.

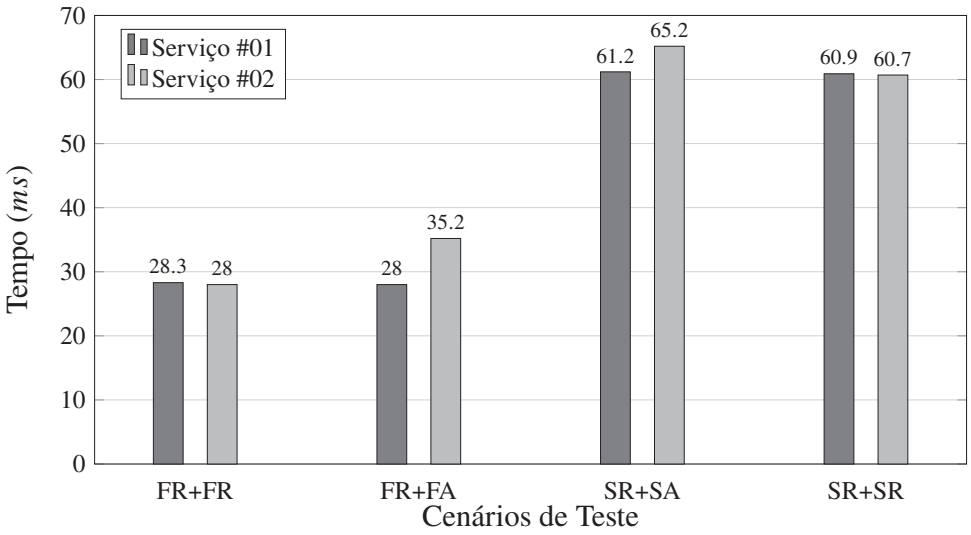


Figura 6.5: Tempo de Processamento da Função de Rede por Serviço Cadastrado

Dentre os cenários de teste do experimento, existem dois cenários de controle: FR+FR e SR+SR. Nesses cenários, além do envio da mesma quantidade de requisições para ambos os serviços, o conteúdo de todas as requisições é idêntico – do tipo FR ou SR. Sendo assim, como apresentado na Figura 6.5, a variação no tempo absoluto dedicado ao processamento das requisições de cada serviço pela função de rede variou marginalmente (entre 0.2 e 0.3 milissegundos). Este fenômeno permite observar que o ambiente de emulação está, de fato, isolado e não existem fatores externos variáveis que apresentem impactos significativos no tempo consumido para o processamento das requisições.

O cenário de testes FR+FA e SR+SA exibem a relação do tempo de processamento entre requisições com mesma quantidade de carga útil, porém estas incluindo padrões maliciosos quando enviadas pelo “Cliente #01” ao “Serviço #01” e não incluindo padrões maliciosos quando enviadas pelo “Cliente #02” ao “Serviço #02”. Naturalmente, o tempo total de processamento da inspeção de tráfego para pacotes com mais bytes em seu conteúdo (requisições do tipo *Slow*) é maior que aquele constatado para o processamento de pacotes com menos bytes em sua carga útil (requisições do tipo *Fast*): 118,6% e 85,2% de aumento para, respectivamente, requisições do tipo *Reject* (FR e SR) e *Accept* (FA e SA).

É importante ressaltar que a diferença percentual relacionada ao aumento do tempo de processamento entre requisições do tipo *Reject* (FR e SR) e *Accept* (FA e SA) decorre da necessidade do processamento completo do conteúdo de um pacote para detectar que o mesmo não carrega assinaturas maliciosas, processando 50 bytes para requisições FA e 500 bytes para requisições SA; já quando existem assinaturas maliciosas presentes na carga útil do pacote, a busca é realizada apenas até que a mesma seja detectada, processando 4 bytes para requisições FR e 400 bytes para requisições SR. Esse fenômeno também é verificado quando comparado o tempo de processamento acumulado entre as diferentes requisições do tipo *Fast* e *Slow*, sendo sempre aquelas relacionadas à aceitação do pacote mais onerosas nas seguintes proporções: 25,7% (FR e FA) e 6,5% (SA e SR).

Com isso, é possível argumentar que funções de rede compartilhadas devem considerar a existência de agentes de monitoramento particularizados para cada um dos serviços atendidos. Esses agentes devem contemplar uma miríade de métricas e indicadores, permitindo detectar a qual serviço (ou se a todos eles) estão relacionados eventos não desejados, a exemplo de sobrecargas e ataques. Como demonstrado nos experimentos acima, as arquiteturas e protótipos propostos no contexto desta Tese apresentam características desejáveis e capacidades técnicas suficientes para implementar tais funcionalidades de monitoramento e gerência.

6.3 EMULAÇÃO DE AMBIENTES NFV

Um dos grandes desafios para desenvolvedores e operadores de tecnologia NFV consiste da avaliação de, por exemplo, desempenho e comportamento das funções de rede antes de seu lançamento e execução em redes de produção. As dificuldades em fazer esses testes envolvem limitações de infraestruturas e a pequena quantidade de ambientes NFV disponíveis atualmente. Com isso, sistemas para a emulação NFV se tornam importantes ferramentas para orientar o processo de desenvolvimento, testagem e avaliação de funções e serviços de rede virtualizados. O uso de emuladores na avaliação de aplicações antes de seus lançamentos é historicamente adotado no contexto de redes de computadores tradicionais (Imran et al., 2010) (Salopek et al., 2014). Da mesma forma que para esses usos históricos, ao se introduzir ferramentas de emulação que suportem infraestruturas NFV (NFVI) e se integrem naturalmente a gerentes e orquestradores (MANO), os desenvolvedores e operadores são melhor instrumentados e ganham maior liberdade para aprimorar suas funções de rede sem correr o risco de comprometer o ambiente de produção. Porém, apesar dos diversos benefícios, os sistemas de emulação NFV atuais são limitados devido à falta de suporte a diversas plataformas de execução de VNF, não são intuitivos e envolvem uma íngreme curva de aprendizagem para poderem ser completamente aproveitados.

Os principais emuladores atuais de ambientes NFV são: *Extensible Service Chain Prototyping Environment* (EsCAPE) (Sonkoly et al., 2015), um emulador de redes Mininet (Lantz et al., 2010) que suporta a execução de funções Click (Kohler et al., 2000) em contêineres; *Multi Datacenter Service Chain Emulator* (MeDICINE) (Peuster et al., 2016), um emulador de ambientes multi-domínio baseados em Mininet com funções de rede virtualizadas conteinerizadas; SONATA (Dräxler et al., 2017), um emulador semelhante ao MeDICINE, baseado em Mininet e funções de rede conteinerizadas, mas que também provê compatibilidade com outros sistemas (*e.g.*, MANO) amparados em especificações da ETSI NFV; e Maxinet (Wette et al., 2014), que não foi desenvolvido especificamente para NFV, mas suporta a execução de funções de rede através de suas máquinas virtualizadas a partir de processos isolados. Além disso, o Maxinet também possui um módulo distribuído que permite a utilização de múltiplas máquinas físicas em uma mesma rede. Apesar de existirem diversas opções de emuladores, nenhuma suporta a virtualização de plataformas de execução de VNF como máquinas virtuais (sejam completamente virtualizadas ou paravirtualizadas). Essa característica inviabiliza a emulação em diversos cenários, uma vez que plataformas como ClickOS (Martins et al., 2014), Click-On-OSv (Marcuzzo et al., 2017) e Leaf (Flauzino et al., 2020) não podem ser testadas. Inclusive, o protótipo COVEN apresentado no Capítulo 4 não é utilizável no contexto dos emuladores descritos. Finalmente, nenhum dos emuladores NFV prevê a existência de soluções de EMS em seus ambientes. Tal fenômeno pode resultar em dificuldades de monitoramento e avaliação das funções e serviços de redes sendo testados.

O sistema *NFV Infrastructure Emulation Platform* (NIEP) (Tavares et al., 2018) (Fulber-Garcia et al., 2020c) é um emulador de ambientes NFV multi-domínio baseado em Mininet (Lantz et al., 2010). O NIEP pode ser executado como um sistema distribuído em várias máquinas físicas

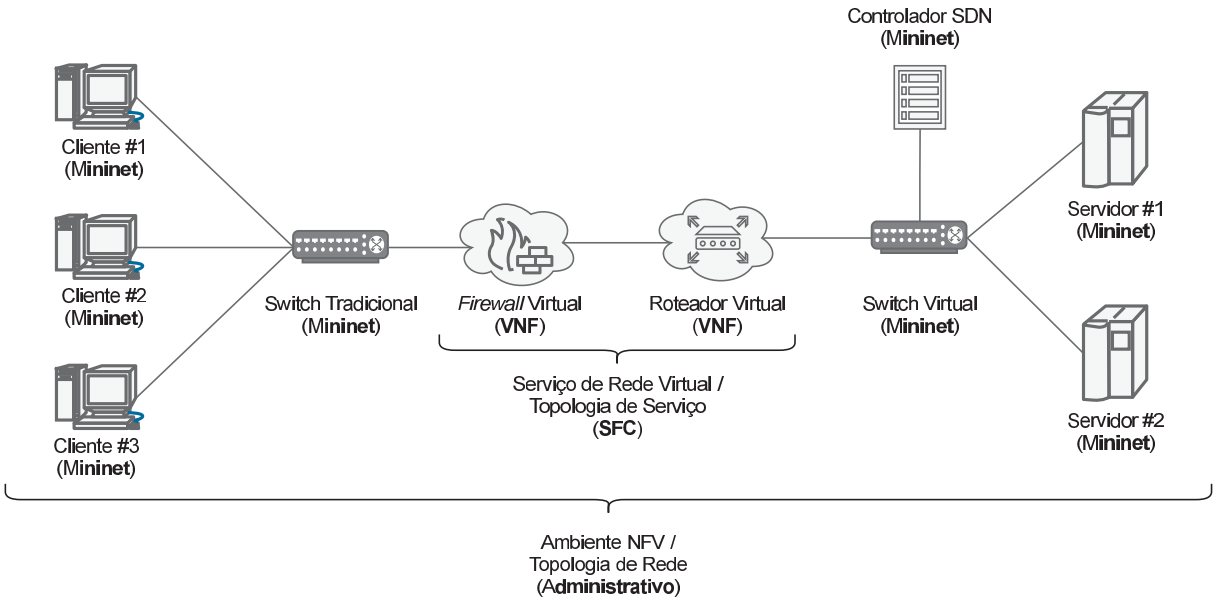


Figura 6.6: Exemplo de Ambiente NFV do NIEP

presentes em uma mesma rede local, suportando a utilização da plataforma de execução de VNF Click-On-OSv (Marcuzzo et al., 2017) na operacionalização de funções de rede desenvolvidas com Click Modular Router (Kohler et al., 2000). Esse emulador é o primeiro a nativamente disponibilizar compatibilidade com uma plataforma de execução de VNF não conteinerizada. A arquitetura do sistema NIEP é dividida em quatro contextos: (i) administrativo; (ii) SFC; (iii) VNF; e (iv) Mininet. Esses contextos são descritos a seguir.

O contexto administrativo (i) é o de mais alto nível e é onde são definidos os ambientes NFV emulados. Esses ambientes podem compreender máquinas de clientes e servidores, *switches* tradicionais ou baseados em SDN (com seus respectivos controladores) e funções e serviços de rede virtualizados. Uma topologia de rede (conexões entre os elementos) deve ser definida no contexto administrativo. O contexto de SFC (ii) é voltado para a criação e gerência de serviços de rede. Nesse contexto são firmadas as conexões lógicas entre instâncias de VNF que compõem um serviço de rede (topologia de serviço). Ressalta-se que qualquer operação de gerência executada no contexto de SFC é aplicada em todas as instâncias de VNF do serviço alvejado. No contexto de VNF (iii) são definidas e gerenciadas as instâncias de VNF individuais. Nesse contexto, o agente de gerência (MA) das instâncias de VNF é acessado, habilitando o monitoramento e a realização de modificações nas funções de rede hospedadas nas mesmas. Nesse cenário, o NIEP atua como um OSS acessado diretamente as instâncias através do protocolo de comunicação da plataforma Click-On-OSv. Instâncias de VNF pertencentes a um serviço de rede também podem ser acessadas de forma particular no contexto de VNF. Todas as instâncias de VNF são virtualizadas pelo hipervisor KVM. O contexto Mininet (iv) provê de acesso a outros elementos de rede gerenciados pelo próprio emulador Mininet. Nesse contexto são realizadas a comunicação e gerenciamento de clientes, servidores, *switches* e controladores. A Figura 6.6 exemplifica um ambiente NFV genérico do NIEP e destaca os contextos aos quais pertencem cada um dos seus elementos.

Atualmente, o sistema de emulação NIEP foi expandido para instanciar e gerenciar tanto a plataforma de execução de VNF Click-On-OSv (nativo), quanto a plataforma COVEN, utilizando para isto o EMS HoLMES. Originalmente, o NIEP gerencia diretamente apenas a plataforma de execução de VNF Click-On-OSv, sendo a plataforma COVEN gerenciada

exclusivamente através do EMS HoLMES (alternativamente, é também possível acessar o MA do COVEN empregando sistemas terceirizados). Esse cenário de gerenciamento direto da plataforma Click-On-OSv decorre de, no momento de lançamento do sistema de emulação, não existirem soluções de EMS que pudessem ser importadas para o emulador, as quais possibilitariam que um gerenciamento holístico fosse realizado para diferentes plataformas de execução de VNF. Assim, para ser possível a inclusão de instâncias de VNF executando em outras plataformas, o código-fonte do emulador deveria ser modificado para reconhecer as interfaces de comunicação e operações de gerência disponibilizadas por cada uma das plataformas almejadas. Agora, essa limitação pode ser tratada pela adoção de soluções de EMS compatíveis com a arquitetura descrita no Capítulo 5. Ou seja, a interface presente no AS disponibiliza operações de gerência de forma padronizada, independentemente da plataforma de execução de VNF sendo gerenciada por uma solução de EMS específica. Também, soluções como o protótipo HoLMES, que possibilitam que um único EMS seja usado de forma abrangente, permitem expandir sob demanda o conjunto de plataformas de execução de VNF suportadas (através do seu sistema de *drivers*). Dessa forma, múltiplas plataformas podem ser facilmente introduzidas no NIEP e, junto delas, funcionalidades inovadoras em NFV são incluídas na emulação, principalmente considerando os módulos operacionais da arquitetura apresentada no Capítulo 4. Dentre essas funcionalidades constam o processamento NSH nativo pelo NSHP e a utilização de agentes estendidos dedicados a colher informações de avaliação das funções de rede sendo emuladas. Ressalta-se que tais funcionalidades já estão implementadas no protótipo COVEN.

6.3.1 Experimentação e Resultados

Visando verificar o impacto da inclusão das plataformas propostas como parte desta Tese (COVEN e HoLMES) no sistema de emulação NIEP, esta subseção apresenta testes de desempenho relacionados a variação do *Round Trip Time* (RTT) verificados para a execução de diferentes operações de gerenciamento requisitadas a uma instância da solução HoLMES e aplicadas em uma instância da plataforma de execução de VNF COVEN, ou requisitadas diretamente ao MA da plataforma COVEN. Cada caso de teste, a serem descritos a seguir, considera a execução das instâncias virtuais tanto diretamente no sistema hospedeiro (sendo processos conectados a interfaces de redes virtuais diferentes), quanto no contexto do sistema de emulação NIEP (utilizando virtualização baseada em processos).

Para a realização dos testes, a instância da plataforma COVEN foi configurada e inicializada com a função de rede de análise, encaminhamento e descarte de tráfego desenvolvida no contexto dos experimentos na Seção 6.2. Dessa forma, todos os agentes estendidos apresentados na Subseção 6.2.1 (*Post Service IP*, *Get NF Statistics* e *Get Services Statistics*), além de as demais operações de gerenciamento protocolares e nativas das plataformas COVEN e HoLMES, também estão disponíveis para execução dos testes relacionados à presente subseção.

O cenário de teste, além de apresentar uma instância da plataforma COVEN sendo gerenciada por uma instância do sistema HoLMES, também conta com uma instância virtual que assume o papel de um cliente, o qual dispara as requisições por operações de gerenciamento para o EMS. Todas as instâncias virtuais estão conectadas na mesma rede local. A Figura 6.7 ilustra, em alto nível, o ambiente considerado para a execução dos experimentos.

O conjunto de operações de gerenciamento considerado para a realização dos testes descritos consiste em:

- *Get Status* (GS): processo para a verificação do estado da plataforma de execução de VNF. Em suma, informa se a plataforma está ligada e se o processamento de tráfego de rede está ativo ou não.

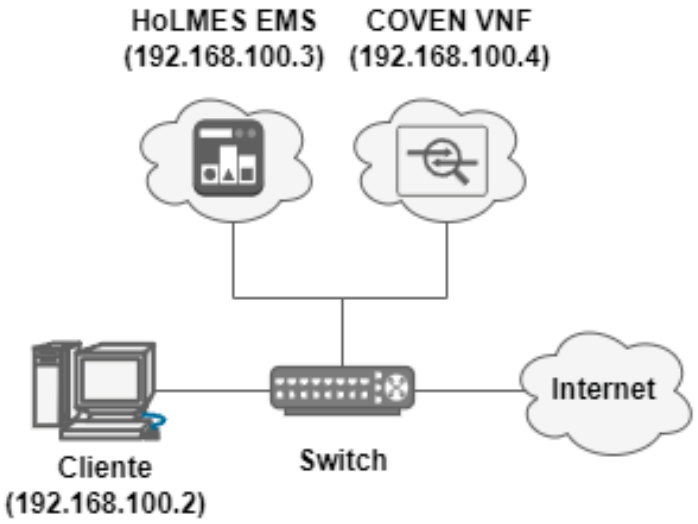


Figura 6.7: Cenário de Testes para Emulação em NFV

- *Post NF* (PN): processo para envio de funções de rede para a plataforma COVEN por um pacote de VNF. O pacote utilizado contém a função de rede de análise, encaminhamento e descarte de tráfego, além do seu respectivo arquivo de configuração, totalizando 1909 bytes.

- *Get NF Statistics* (GNS): processo para verificação da quantidade de pacotes e bytes recebidos pela função de rede. Esta operação de monitoramento é provida como um agente estendido à função de análise, encaminhamento e descarte de tráfego empregada.

O experimento foi realizado em uma máquina com Core I3 4010, 8GB de memória RAM DDR3, executando o sistema operacional Ubuntu 16.04. Todos os testes foram repetidos 30 vezes, resultando em um intervalo de confiança de 95%. A Figura 6.8 apresenta os resultados obtidos. No gráfico, o eixo X representa a operação sendo testada, o eixo Y indica a média do RTT para as operações correspondentes (em milissegundos), sendo as barras de erro o desvio padrão e os padrões impressos nas barras principais os casos de teste considerados: requisição direta ao MA/VNF (MA), requisição direta ao MA/VNF em ambiente emulado (NIEP+MA), requisição intermediada pelo EMS (EMS) e requisição intermediada pelo EMS em ambiente emulado (NIEP+EMS).

A utilização do sistema de emulação NIEP gera sobrecargas que impactam significativamente o tempo de resposta (RTT) das operações de gerenciamento testadas. Tal impacto representa um aumento na média do RTT que varia de 139,7% a 519,5%, quando comparados casos emulados e não-emulados análogos. A intensidade do aumento na média do tempo de resposta constatada depende, principalmente, das características das operações executadas. Porém, a existência permanente desse aumento, independentemente da operação analisada, decorre de uma característica do sistema NIEP: a utilização, mesmo que implicitamente, de *switches* e controladores SDN (e, por consequência, do protocolo OpenFlow) para a transmissão de tráfego internamente. Tal característica é herdada do emulador Mininet, subjacente ao NIEP. A utilização do protocolo OpenFlow resulta em perdas de desempenho quando comparado ao encaminhamento de tráfego local gerenciado pelo *kernel* de sistemas Linux (Gelberger et al., 2013). Essas perdas de desempenho estão relacionadas principalmente a processos internos de identificação de fluxos e manutenção de cabeçalhos extras em quadros e pacotes de rede.

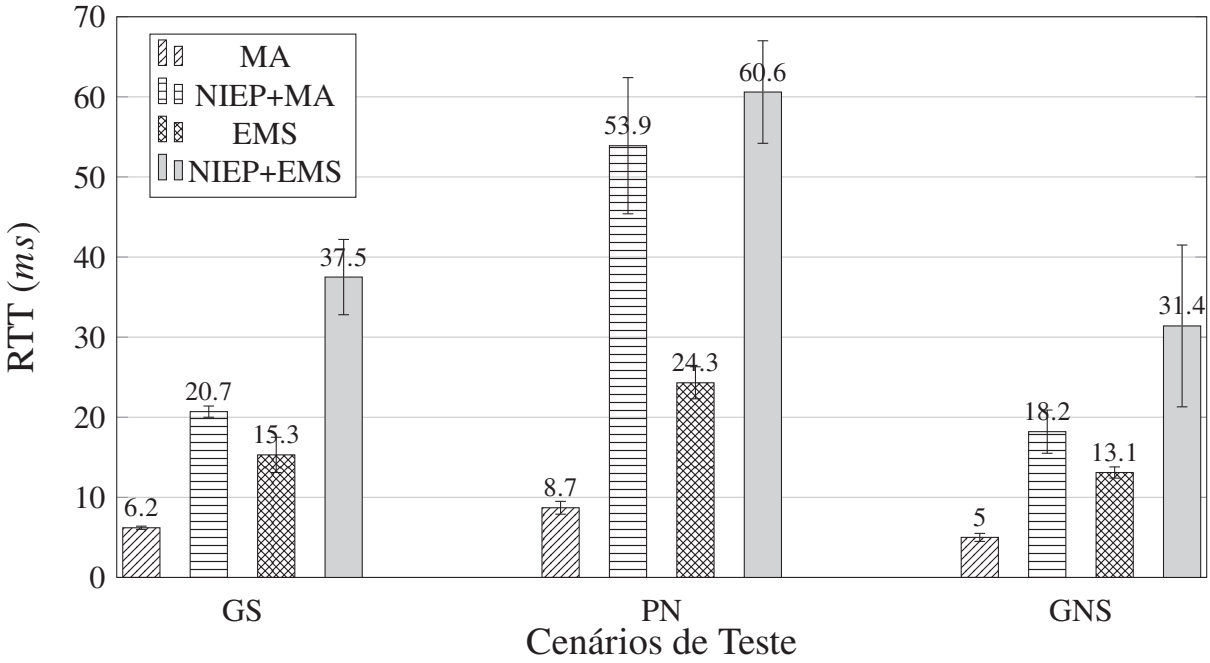


Figura 6.8: Execução de Operações de Gerenciamento em Cenários Emulados e Não-emulados

Particularmente, os cenários de teste GS e GNS são semelhantes quanto as características das operações de monitoramento requisitadas. Em ambos os casos, a operação é considerada leve, realizando uma consulta e retornando o seu resultado. No caso de GS, essa consulta é feita à própria plataforma de execução de VNF. Já no caso de GNS, a consulta é requisitada a um agente estendido da função em execução na plataforma. Sendo assim, essas operações não implicam em processamentos significativamente onerosos (*i.e.*, que predominem no RTT) e nem mesmo realizam transferência de dados. Com isso, é nesses cenários onde são verificados os menores tempos de resposta em todos casos de teste considerados. Além disso, é neste contexto em que a diferença do tempo de resposta entre casos emulados e não-emulados análogos é menor, existindo um aumento para os casos emulados variando de 139,7% a 264%.

O cenário de teste PN, por sua vez, consiste na execução de uma operação de gerenciamento que envolve o envio de um pacote de VNF para ser armazenado na instância da plataforma COVEN. Vale ressaltar, entretanto, que essa operação não realiza nem a configuração, nem a execução da VNF na plataforma, apenas a descompressão, checagem e armazenamento da função de rede em seus diretórios internos. Sendo assim, essa operação não resulta em processos de computação intensiva. Porém, o maior tempo de resposta verificado para este cenário de teste resulta da necessidade de envio dois quadros para transmissão do pacote de VNF, uma vez que este conta com 1909 bytes, sendo a MTU das máquinas utilizadas configuradas com o padrão de 1500 bytes. Essa sobrecarga de transmissão também é refletida nas operações realizadas pelo sistema NIEP, resultando nas maiores aumentos de RTT verificados entre casos de teste análogos não-emulados e emulados: de 149,9% a 519,5%.

A adoção do emulador NIEP representou uma sobrecarga significativa no tempo de resposta para a execução das operações de gerenciamento quando este é comparado com a instanciação do ambiente diretamente no sistema hospedeiro. Porém, o sistema NIEP permite a preparação prévia e instanciação automática de um ambiente de experimentação, além de suportar, caso desejado, a configuração refinada dos modelos de encaminhamento de tráfego através de *switches* e controladores SDN. Tais possibilidades podem ser consideradas atenuantes

das sobrecargas verificadas, uma vez que agregam capacidades gerenciais e de personalização de ambientes de experimentação para desenvolvedores e operadores de rede.

6.4 SUMARIZAÇÃO E DISCUSSÃO

Através da padronização dos elementos operacionais do domínio de trabalho VNF, várias possibilidades de aplicações inovadoras do paradigma NFV, ou de aprimoramento daquelas já existentes, surgem como consequência. A investigação, projeto e implementação de aplicações e aprimoramentos estão previstas no presente trabalho. Particularmente, esta seção delineou três oportunidades de uso das plataformas e soluções derivadas das arquiteturas apresentadas nos Capítulos 4 e 5: (i) monitoramento abrangente e personalizável de funções de rede; (ii) compartilhamento de instâncias de VNF entre clientes distintos; e (iii) emulação de ambientes NFV que suportam funções e serviços operacionalizados em diferentes plataformas de execução de VNF. Apesar serem de diferentes naturezas, as oportunidades identificadas se beneficiam tanto da previsibilidade e padronização de módulos operacionais e interfaces de comunicação propiciadas pelas arquiteturas de EMS e VNF, quanto dos modelos operacionais dos protótipos COVEN e HoLMES desenvolvidos como provas de conceito das mesmas.

É possível classificar o monitoramento de funções de rede virtualizadas através da granularidade, abrangência e computabilidade. O monitoramento de instâncias de VNF, atualmente, é majoritariamente realizado com métricas de caixa-preta (alta granularidade), tipicamente não considera hiperpropriedades (baixa abrangência) e simplesmente disponibilizam métricas (baixa computabilidade). É proposta no presente trabalho uma arquitetura para plataformas de execução de VNF (Capítulo 4) que permite o desenvolvimento de agentes de monitoramento estendidos, capazes de gerar uma miríade de métricas particularmente projetadas para funções de rede específicas. Também, a arquitetura de EMS proposta (Capítulo 5) facilita o monitoramento de instâncias de VNF através do emprego de protocolos padronizados e de amplo conhecimento. Experimentos realizados demonstraram a viabilidade da criação, disponibilização e monitoramento de métricas e hiperpropriedades (sejam elas de caixa branca ou preta, de baixa ou alta computabilidade) através da plataforma COVEN e do EMS HoLMES.

Atualmente, poucos trabalhos se dedicam ao compartilhamento de funções e serviços de rede em NFV. Há diversos desafios como a configuração, interoperabilidade, disponibilidade, segurança, privacidade, escalabilidade, desempenho e confiabilidade das funções de rede compartilhadas entre vários clientes. Nesse cenário, as arquiteturas de plataformas de execução de VNF e de EMS apresentadas, respectivamente, nos Capítulos 4 e 5 proveem uma série de capacidades de interesse para suportar o compartilhamento de funções de rede. Entre essas capacidades destaca-se o desenvolvimento de módulos de monitoramento orientados a clientes como componentes, sendo as informações disponibilizadas através de agentes estendidos. Similarmente, agentes estendidos podem funcionar como chaves de configuração que adaptam adequadamente uma função de rede para atender um grupo de clientes específicos, criando várias linhas de produto pelo ajuste de pontos de variação em seu código-fonte. O EMS, no que lhe concerne, pode ser usado para intermediar a comunicação dos clientes com as funções compartilhadas, atribuindo-lhes credenciais com diferentes níveis de acesso às mesmas. Além disso, o EMS pode interfacear a execução de uma gama de novas possíveis operações de gerência, como as de configuração de linhas de produto. Através de um cenário de testes e dos protótipos COVEN e HoLMES, demonstrou-se a possibilidade do provimento de funções de rede compartilhadas entre serviços de diferentes clientes, assim como o monitoramento particularizado destes.

Por fim, para o cenário de emulação de NFV, foi apresentado o NIEP, um emulador dedicado a instanciação e execução de ambientes de rede baseados em NFV. Nesse contexto, foi proposta a expansão do escopo de atuação do emulador através da utilização de um EMS para o interfaceamento da comunicação entre o NIEP (agindo como um OSS) e diferentes plataformas de execução de VNF. Considerando que a solução de EMS adotada seja regida pela arquitetura descrita no Capítulo 5, a expansão do emulador considerou apenas a implementação do protocolo Ve-Vnfm-em para habilitar a comunicação com qualquer plataforma de execução de VNF. Particularmente, o NIEP se beneficia muito da estratégia de *drivers* do protótipo HoLMES (Capítulo 5), permitindo este seja o único EMS operando no sistema. Com isso, também foi possível incluir o suporte à plataforma COVEN (Capítulo 4), o que consequentemente disponibiliza funcionalidades inovadoras para a emulação, como execução de funções de rede compostas por múltiplos componentes e processamento NSH nativo. Através das modificações realizados no emulador NIEP e da utilização dos protótipos COVEN e HoLMES, foram demonstrados os impactos relacionado a sobrecarga no tempo de resposta (RTT) em ambientes emulados e não-emulados na execução de operações de gerenciamento para funções de rede em cenários de teste específicos.

De forma abrangente, este capítulo apresentou algumas oportunidades que exploram características e funcionalidades presentes nas arquiteturas de plataformas de execução de VNF e de EMS definidas, em ordem, nos Capítulos 4 e 5. Nesse caso, os protótipos COVEN e HoLMES surgem como meios práticos para alcançar tais oportunidades no paradigma NFV. Por fim, também foram apresentados cenários de experimentação e casos de teste que evidenciam a viabilidade das oportunidades discutidas no decorrer do capítulo.

7 MAPEAMENTO PERSONALIZADO DE SERVIÇOS DE REDE BASEADO EM HEURÍSTICAS GENÉTICAS

Este capítulo apresenta uma estratégia de mapeamento de serviços de rede em ambientes multi-domínio. Inicialmente, a Seção 7.1 traz uma breve fundamentação relacionada a heurísticas genéticas. Em seguida, a Seção 7.2 descreve a solução *Genetic Service Mapping* (GeSeMa), uma alternativa flexível e parametrizável para a execução da integração de serviços de rede através do mapeamento multidomínio. A Seção 7.3 descreve os estudos de caso considerados para a experimentação da solução proposta, além de discutir os resultados obtidos através de um conjunto de testes. Finalmente, a Seção 7.4 sumariza e discute o capítulo na totalidade.

7.1 HEURÍSTICAS GENÉTICAS

Algoritmos genéticos se baseiam nos princípios darwinianos de evolução das espécies para resolver uma série de problemas (*e.g.*, otimização, agrupamento, simulação, etc.) (Hasançebi e Erbatur, 2000). Esses algoritmos analisam indivíduos representados através de um cromossoma (*i.e.*, um vetor, que representa uma possível solução) composto por um conjunto de genes (*i.e.*, características pontuais do problema), cada um determinado por um alelo (*i.e.*, um valor que um gene pode assumir). Os indivíduos são submetidos a um processo evolutivo, tipicamente baseado nas operações de cruzamento e mutação, para obter novos indivíduos mais aptos a solucionar o problema. Nas operações de cruzamento, dois ou mais indivíduos são selecionados e seus cromossomas parcialmente combinados para criar um novo indivíduo descendente. Já em operações de mutação, um indivíduo é submetido a uma mudança particular no alelo de um gene específico. É importante ressaltar que algoritmos genéticos são heurísticas e não garantem a obtenção do resultado ótimo global. Além disso, por executarem operações estocásticas, múltiplas execuções de uma mesma heurística genética podem retornar resultados heterogêneos entre si. Em especial, as heurísticas genéticas são adequadas para resolverem problemas com grande espaço de busca, obtendo soluções factíveis em um tempo de execução viável.

Quando utilizadas para solucionar problemas de otimização, as heurísticas genéticas se subdividem em duas classes principais: mono-objetivo e multiobjetivo. Heurísticas dedicadas a problemas mono-objetivo visam otimizar apenas uma função ou métrica de avaliação, enquanto aquelas desenvolvidas para problemas multiobjetivo determinam uma relação de custo-benefício entre múltiplas funções e métricas, normalmente definida através da análise das fronteiras de Pareto. Indivíduos em uma mesma fronteira não apresentam relação de dominância entre si, isto é, para cada indivíduo existe ao menos uma métrica ou função entre o conjunto analisado cujo resultado é melhor em relação aos demais. Também, problemas de otimização podem apresentar restrições quanto a regiões indisponíveis do espaço de busca e combinações de genes que geram indivíduos inválidos. Diversas técnicas podem ser utilizadas para tratar essas restrições: controle da população inicial, especificação de mutações inválidas, descarte de indivíduos, entre outros. No contexto do GeSeMa são considerados problemas de otimização multiobjetivo com restrições e utilizados os algoritmos *Nondominated Sorting Genetic Algorithm II* (NSGAII) (Deb et al., 2002) e *Strength Pareto Evolutionary Algorithm 2* (SPEA2) (Zitzler et al., 2001).

7.2 *GENETIC SERVICE MAPPING*

A solução *Genetic Service Mapping* [1] (GeSeMa) baseia-se em heurísticas genéticas para realizar o mapeamento de serviços de rede em ambientes com múltiplos domínios. O objetivo é oferecer uma solução flexível e customizável, que permita ao usuário definir não apenas o serviço e suas dependências, mas também as métricas e objetivos de otimização que devem ser avaliados. As informações são providas ao GeSeMa por um modelo de requisição estruturado pela linguagem de serialização de dados *YAML Ain't Markup Language* (YAML). Esta subseção apresenta o modelo de requisição e a metodologia de mapeamento multidomínio adotada pela solução proposta.

7.2.1 Modelo de Requisição

O modelo de requisição da solução GeSeMa, ilustrado na Figura 7.1, apresenta três objetos principais que especificam: (i) a estrutura do serviço a ser mapeado e de suas funções de rede (`SERVIÇO`); (ii) as métricas, restrições e objetivos de otimização do mapeamento (`MÉTRICAS`); e (iii) os domínios disponíveis e suas características individuais (`DOMÍNIOS`). Particularmente, o objeto de `SERVIÇO` carrega a especificação da topologia do serviço de rede requisitado. Essa topologia é representada através de uma cadeia de caracteres especificada seguindo as regras sintáticas e semânticas do modelo *Service ChAin Grammar* (SCAG) (Fulber-Garcia et al., 2019a). Da mesma forma, o objeto `SERVIÇO` especifica os requisitos computacionais de cada uma das funções de rede utilizadas na topologia do serviço, estes dados estão incluídos no subobjeto `FUNÇÕES`. É importante observar que para cada função especificada na topologia do serviço existe também uma entrada correspondente no subobjeto `FUNÇÕES`. Essa entrada, registrada com o identificador da função, contém dados relativos aos requisitos mínimos da função como, por exemplo, memória, núcleos de processamento virtuais e interfaces de rede virtuais, estes definidos por valores positivos do tipo inteiro.

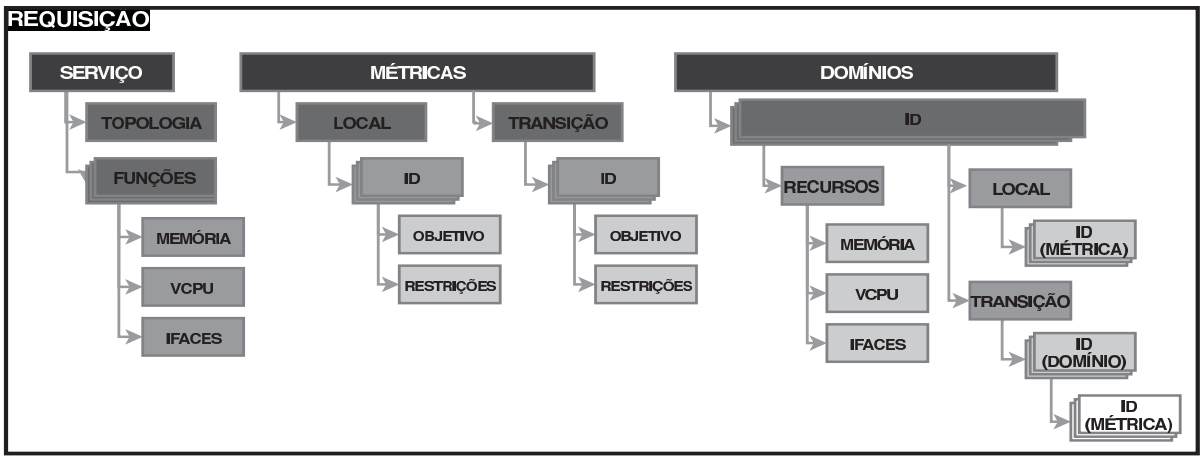


Figura 7.1: Modelo de Requisição GeSeMa

O objeto `MÉTRICAS` especifica critérios utilizados pela heurística genética do GeSeMa na busca, avaliação e otimização de candidatos a resolver a requisição de mapeamento. As métricas empregadas na solução proposta podem ser de duas categorias: local e transição. Métricas locais são avaliadas no momento da alocação de uma função de rede em um dado domínio (*e.g.*, custo da alocação, sobrecarga do domínio, etc.), enquanto métricas de transição

[1] Disponível em https://github.com/ViniGarcia/NFV-FLERAS

são utilizadas somente na ocorrência de uma mudança de domínio (*e.g.*, atraso, saltos, distância geográfica, etc.). Essas categorias são representadas no documento de requisição através dos subobjetos `LOCAL` e `TRANSIÇÃO`. Dessa forma, cada um desses subobjetos podem conter múltiplas métricas. Cada métrica deve ser unicamente identificada (`ID`), além de apresentar dois atributos obrigatórios: `OBJETIVO` e `RESTRIÇÕES`. Os objetivos declaram como a métrica deve ser computada em busca de resultados ótimos. Isso ocorre ao definir o problema como uma `MAXIMIZAÇÃO` ou `MINIMIZAÇÃO`. Finalmente, as restrições consistem de uma lista de cadeias de caracteres que definem limiares de aceitação para o valor de uma métrica. Para isso os operadores relacionais "<", ">", "<=", ">=", "==" e "! =" podem ser utilizados em conjuntos a valores reais ou inteiros (*e.g.*, "< 30", "== 0", ">= 15, 5").

Por fim, o objeto `DOMÍNIOS` especifica as infraestruturas físicas e virtuais disponíveis e suas transições, gerando um grafo direcionado, no qual os vértices carregam informações de métricas locais e as arestas armazenam dados relativos às métricas de transição. Um domínio particular é caracterizado por um subobjeto com identificador único (`ID`). Cada subobjeto de domínio, por sua vez, contempla três outros subobjetos: `RECURSOS`, `LOCAL` e `TRANSIÇÃO`. O subobjeto de `RECURSOS` informa a disponibilidade do domínio em relação à memória (`MEMÓRIA`), núcleos virtuais de processamento (`VCPU`) e interfaces virtuais de rede (`IFACES`). Já os subobjetos `LOCAL` e `TRANSIÇÃO` são utilizados na declaração de valores de *benchmarking* (ou adquiridos em catálogos de provedores de infraestrutura) utilizados no processo de otimização. Esses subobjetos têm forte relação com o objeto de `MÉTRICAS`, devendo haver correspondência completa entre os identificadores das métricas declaradas e os identificadores dos *benchmarkings* fornecidos. Em especial, cada entrada no subobjeto de `TRANSIÇÃO` declara, primeiramente, para qual domínio a transição ocorre (utilizando o identificador do mesmo) e só então define os valores das métricas a serem utilizadas na otimização da transição propriamente dita.

7.2.2 Metodologia Genética de Mapeamento Multidomínio

A solução GeSeMa é baseada em dois algoritmos genéticos amplamente utilizados: NSGAII (Deb et al., 2002) e SPEA2 (Zitzler et al., 2001). A decisão por um dos algoritmos cabe ao usuário, este podendo usufruir de qualidades específicas dos mesmos que melhor se adequam às peculiaridades do serviço de rede e aos objetivos do mapeamento requisitado. As principais características da solução proposta são sumarizadas a seguir:

1. **Indivíduos**: cada indivíduo apresenta um cromossoma modelado através de um vetor de tamanho N, onde $N > 1$ equivale à quantidade de funções de rede que compõem o serviço (*i.e.*, cada função de rede é mapeada para uma posição do vetor). Os genes consistem de números inteiros com valor (*i.e.*, alelo) presente no intervalo de indexadores $[0, M - 1]$, sendo $M > 0$ a quantidade de domínios disponíveis para a realização do mapeamento.

2. **População**: a população inicial é criada de maneira aleatória, garantido apenas que nenhum dos indivíduos gerados viola dependências de domínios particulares (*i.e.*, fixando o valor do domínio presente na dependência na posição do vetor representante da função que a carrega). O tamanho da população é um parâmetro decidido pelo usuário.

3. **Objetivos e restrições**: objetivos e restrições (políticas, topologia de rede, recursos computacionais e dependências de domínio) são avaliados para todos os indivíduos em cada geração. Na ocorrência de um indivíduo inválido, este é registrado em uma lista tabu, evitando reavaliações em uma eventual nova ocorrência do mesmo. Além disso,

mecanismos de recuperação de indivíduos inválidos também são utilizados (discutidos posteriormente).

4. **Seleção**: a seleção de indivíduos para passarem pelo processo de cruzamento é realizada através do mecanismo de torneio. Nesse mecanismo, I indivíduos são selecionados aleatoriamente e aquele presente na fronteira de Pareto mais externa é escolhido. O tamanho do torneio $I > 1$ é determinado pelo usuário.

5. **Cruzamento**: quatro operadores de cruzamento são fornecidos aos usuários: *Simulated Binary Crossover* (SBX), *Half Uniform Crossover* (HUC), *Partially Mapped Crossover* (PMC) e *Subtour Selection Crossover* (SSC). Cada um apresenta comportamentos operacionais diferentes, sendo que a escolha por um deles deve considerar as particularidades de cada requisição (Hasançebi e Erbatur, 2000). A taxa de cruzamento (*i.e.*, probabilidade de aplicação do operador) por geração também pode ser configurada pelo usuário.

6. **Mutação**: dois operadores de mutação são providos: mutação por redefinição e mutação por troca. A redefinição escolhe um gene aleatório e substitui seu alelo também de forma aleatória. A troca, no que lhe concerne, escolhe dois genes aleatórios e substitui seus alelos entre si. O operador de mutação nunca é aplicado em alelos com dependências de domínios. Assim como acontece para a taxa de cruzamento, o usuário também pode ajustar a taxa de mutação.

A execução da solução GeSeMa acontece em dois processos principais: (i) validação e aplicação da requisição e configurações genéticas; e (ii) criação e evolução da população. O primeiro processo identifica os arquivos e parâmetros de entrada e os analisa de modo a verificar sua validade. Inicialmente, o arquivo de requisição é validado conforme o modelo especificado anteriormente, mapeando as estruturas definidas em alto nível para elementos iteráveis pela solução genética (*i.e.*, lista e dicionários padronizados). Em seguida, as configurações genéticas já citadas (*i.e.*, tamanho da população, tamanho do torneio de seleção, operador/taxa de cruzamento e operador/taxa de mutação), em conjunto com o número máximo de gerações (também definido pelo usuário), são verificados e, caso sejam válidos, são utilizados para inicializar os operadores genéticos da solução. A conclusão do primeiro processo gera um conjunto de recursos que serão utilizados para a determinação e evolução de indivíduos aptos a resolver o problema de mapeamento durante o segundo processo.

A execução do segundo processo da solução GeSeMa é sumarizada na Figura 7.2. Inicialmente, o serviço de rede requisitado, provido na forma de uma cadeia de caracteres segundo a gramática SCAG (Fulber-Garcia et al., 2020b), é remodelado para ser processado pelas soluções genéticas (Figura 7.2: A e B). Além disso, uma população inicial é gerada de forma a criar um conjunto aleatório de indivíduos válidos quanto à topologia da rede (transições entre domínios válidas) e às dependências de domínio (fixando funções com dependências em seus respectivos domínios). Em seguida, é realizada a avaliação dos indivíduos (Figura 7.2: C) onde estes são verificados quanto à sua validade em relação a restrições de recursos dos domínios escolhidos e políticas. Assim, um procedimento iterativo de agregação define o valor dado a cada uma das métricas de otimização para cada um dos indivíduos avaliados. Então, uma sequência de seleções (Figura 7.2: D) e de aplicações dos operadores genéticos de cruzamento e mutação (Figura 7.2: E e F, respectivamente) é realizada a fim de evoluir a população. Todas as etapas ilustradas pela Figura 7.2: C, D, E e F consistem das operações aplicadas para a conclusão de uma geração (Figura 7.2: G). A cada geração os melhores resultados (fronteira de Pareto) são salvos separadamente para serem reconsiderados em gerações posteriores. Ao final de um

número definido de gerações, a fronteira de Pareto determinada pela solução é retornada como resultado do processamento (Figura 7.2: H).

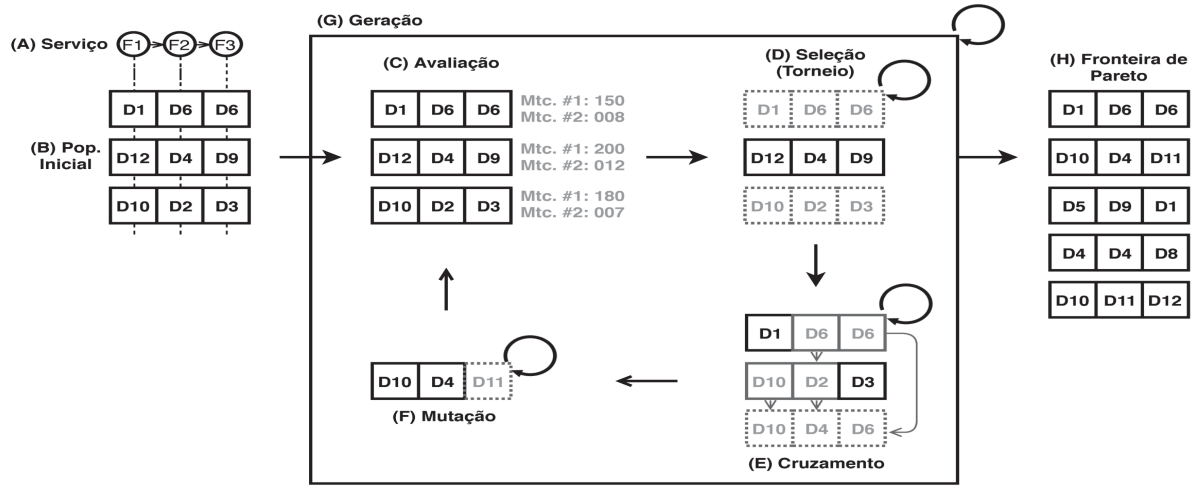


Figura 7.2: Sumarização do Processamento GeSeMa

Em especial, a etapa de avaliação executa diversas tarefas que viabilizam e otimizam as demais etapas do algoritmo genético. A avaliação é realizada de forma iterativa, percorrendo o cromossoma de cada indivíduo, gene por gene. Assim, a cada iteração são calculadas as métricas locais de acordo com o alelo (domínio) e as métricas de transição, caso necessário, quando um alelo do gene anterior de uma função de rede conectada tem valor diferente do alelo do gene atual. É importante notar que, além do seu próprio alelo, cada gene apresenta um vetor de conexões contendo índices de genes anteriores, estes indicando as funções de rede diretamente conectadas à função de rede representada pelo gene atual. Dessa forma, topologias de rede com ramificações podem ser representadas por um cromossoma linear. O conjunto de avaliações de cada alelo são todos finalmente agregados e então usados para classificar o indivíduo avaliado em relação à dominância de Pareto. Indivíduos considerados inválidos são incluídos em uma lista tabu. Assim, em caso de uma nova ocorrência dos mesmos, duas ações podem ser tomadas: substituição do indivíduo por um novo indivíduo aleatório (caso sejam infringidas restrições topológicas e políticas) ou redução de redundância de domínios (caso sejam infringidas restrições de recursos). Dependências de domínio não são tratadas durante a avaliação, já que nunca são infringidas. Isso ocorre devido à garantia das mesmas na população inicial e a não aplicação de mutações nos alelos dos genes representantes de funções com dependências.

7.3 EXPERIMENTAÇÃO E RESULTADOS

Nesta seção apresentamos uma avaliação do GeSeMa em dois estudos de caso[2]. O primeiro estudo de caso é apresentado na Subseção 7.3.1 e consiste no mapeamento de um serviço de cache hierárquico em uma rede composta por 114 domínios. Este estudo de caso permite avaliar a convergência e o tempo de execução do GeSeMa, entre outras métricas. Também, comparamos o GeSeMa com outras soluções de mapeamento de serviços baseadas em heurísticas clássicas. No segundo estudo de caso (Subseção 7.3.2), comparamos GeSeMa com a solução GA+LCB. Neste estudo de caso, empregamos a mesma topologia de rede (composta por 114 domínios) na qual mapeamos uma cadeia de serviço genérica que consiste em 9 funções de rede.

[2]A implementação está disponível em https://github.com/ViniGarcia/NFV-FLERAS

O GeSeMa foi configurado com as mesmas restrições do GA+LCB, e ambos foram executados com a mesma configuração de avaliação.

7.3.1 Mapeamento de Caches Multimídia Hierárquicas

O GeSeMa foi avaliado por meio de um estudo de caso em que um sistema hierárquico de caches multimídia é mapeado em uma topologia correspondente à rede Amazon AWS, composta por 114 domínios (Wikileaks, 2018). O serviço conecta n funções de cache individuais em uma topologia linear. A primeira função é a cache mestre que recebe e distribui atualizações de conteúdo de um servidor multimídia. As outras funções de cache recebem e fazem solicitações de atualizações de conteúdo de/para seu respectivo cache predecessor. Dessa forma, o conteúdo é distribuído hierarquicamente pela topologia do serviço. Além disso, as seguintes restrições se aplicam. Cada domínio pode hospedar no máximo uma cache. O objetivo dos *stakeholders* é maximizar uma métrica local e uma métrica de transição: a densidade de usuários do serviço multimídia no domínio (local) e a distância geográfica entre caches vizinhas (transição). De forma abstrata, o objetivo é mapear caches para regiões geograficamente distantes entre si e com alta densidade de usuários.

A topologia da rede é totalmente conectada, no sentido de que cada nó pode se comunicar diretamente com qualquer outro, sem ter que empregar intermediários, assim a topologia pode ser representada por um grafo completo. Note que poderíamos ter empregado uma topologia arbitrária (em vez de um grafo completo), porém optamos por empregar o grafo completo para tornar o espaço de busca mais desafiador para o GeSeMa. Cada domínio x (vértice) tem um valor associado relacionado a densidade de usuários v_x^{du} (métrica local). Cada conexão entre dois domínios (arestas) x e y tem um valor que corresponde à sua distância geográfica v_{xy}^{gd} (métrica de transição). Em uma etapa preliminar, testamos várias configurações genéticas para determinar os operadores e a taxa de cruzamento e mutação, tamanho do torneio, modo de população inicial e tamanho da população.

Para este estudo de caso empregamos o algoritmo SPEA2; os resultados para NSGA2 foram ligeiramente inferiores. O SPEA2 foi configurado com os seguintes parâmetros: taxa de cruzamento SBX de 30%, taxa de mutação de substituição de 5%, seleção por torneio binário, população inicial aleatória e tamanho da população de 50 indivíduos. As topologias de serviço consistiam em 7, 9 e 11 caches conectadas linearmente. Os testes foram executados em uma máquina com processador Intel Core I5-3330 (3.0GHz) com 8GB RAM (DDR3, 1600MHz), executando Ubuntu 16.04 e Python 3.5.2. Escolhemos tamanhos de serviço (*i.e.*, o número de funções de rede) que não podem ser calculados usando busca exaustiva em tempo viável (na mesma máquina). Todos os arquivos, programas e *scripts* utilizados estão disponíveis em https://github.com/ViniGarcia/NFV-FLERAS/tree/GesemaExperiments. Os experimentos foram executados 30 vezes alcançando um nível de confiança de 95%.

O primeiro experimento consiste em um teste de convergência. O objetivo deste experimento é validar a viabilidade de GeSeMa na exploração e aprofundamento do espaço de busca, convergindo para uma fronteira de Pareto (sendo ou não a fronteira ótimo global). Os mapeamentos de serviços evoluem por um número indeterminado de gerações, parando quando nenhuma modificação ocorre na fronteira de Pareto após um conjunto de 1500 gerações. Caso tenha ocorrido uma modificação, um novo conjunto de 1500 gerações é executado. Os resultados deste experimento são mostrados nas Figuras 7.3, 7.4 e 7.5 para as topologias de serviço com 7, 9 e 11 funções, respectivamente. As linhas representam a média das fronteiras relativas de Pareto dos candidatos mais bem avaliados em uma determinada geração, as barras de erro mostram

as melhores e piores fronteiras de Pareto encontradas entre os melhores candidatos da mesma geração.

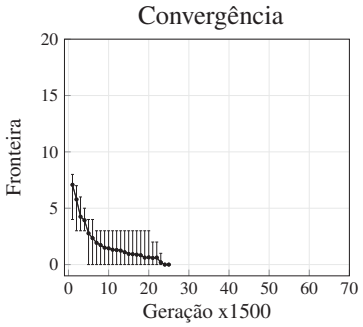


Figura 7.3: Conv. (7 VNFs)

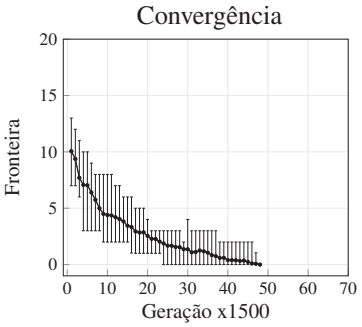


Figura 7.4: Conv. (9 VNFs)

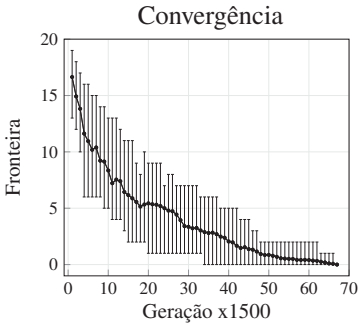


Figura 7.5: Conv. (11 VNFs)

As fronteiras relativas são calculadas da seguinte forma: (i) um número pré-definido de gerações (no nosso caso, 1500 gerações) é avaliado por um algoritmo evolutivo pré-configurado na rodada r; (ii) as fronteiras são computadas e os indivíduos mais aptos da rodada r (doravante denominada "fronteira de Pareto r") são obtidos e salvos; (iii) a fronteira de Pareto r é mesclada com a fronteira de Pareto $r-1$, criando o conjunto $s_{[r-1,r]}$; (iv) as fronteiras relativas de $s_{[r-1,r]}$ são calculadas e sua fronteira de Pareto identificada; (v.i) se nenhum indivíduo presente exclusivamente na fronteira de Pareto r ocorre na fronteira de Pareto de $s_{[r-1,r]}$, consideramos que o algoritmo convergiu, e o próximo passo - (vi) - É executado; (v.ii) se houver pelo menos um indivíduo que apareça tanto na fronteira de Pareto r quanto na fronteira de Pareto de $s_{[r-1,r]}$ (mas não na fronteira $r-1$) , consideramos que a população ainda está evoluindo, e os passos (i), (ii), (iii), (iv) e (v) são repetidos; (vi) as fronteiras de Pareto de cada rodada ($[1;r]$) são mescladas no conjunto $s_{[1,r]}$; (vii) as fronteiras relativas de $s_{[1,r]}$ são calculadas; e finalmente, (viii) informações sobre as fronteiras relativas que aparecem em $s_{[1,r]}$ são atribuídas a cada indivíduo nas fronteiras de Pareto das rodadas $[1;r]$.

No primeiro experimento, a execução do GeSeMa com a configuração descrita anteriormente resultou em uma correlação positiva entre o número de gerações necessárias para fazer o algoritmo genético convergir e a complexidade do problema. Isso é consequência do grande número de candidatos subótimos gerados no início da execução do GeSeMa (a exploração é mais significativa do que o aprofundamento nessa fase), substituídos em poucas gerações. Assim, normalmente, quanto maior o espaço de busca (número de domínios disponíveis), mais complexo é o problema (número de VNFs na topologia do serviço), e mais exploração é necessária para encontrar regiões apropriadas do espaço de busca a serem aprofundadas. Pela mesma razão, o número de transições de fronteira aumenta conforme o espaço de busca e a complexidade do problema. No experimento, o mapeamento da topologia do serviço com 7 funções convergiu após 36.000 gerações em 8 fronteiras; com 9 funções convergiu após 72.000 gerações em 13 fronteiras; e com 11 funções convergiu após 100500 gerações em 19 fronteiras.

O segundo experimento mostra o tempo de execução (em segundos) de GeSeMa à medida que o número de gerações aumenta. Este experimento foi executado para as topologias com 7 (Figura 7.6), 9 (Figura 7.7) e 11 (Figura 7.8) funções. Os resultados revelam uma correlação positiva entre o tempo de execução e o número de gerações. No entanto, notamos também que o tempo de execução apresenta pouca variação de acordo com que os tamanhos da topologia do serviço variam para o mesmo número de gerações. Isso é consequência da restrição que especifica que cada domínio hospeda no máximo uma única cache. Assim, a probabilidade de criar candidatos inválidos aumenta conforme o número de cromossomos. No entanto, esses

candidatos inválidos são descartados antes de serem avaliados, o que reduz o impacto no tempo de execução. Mas, apesar de manter o tempo de execução estável, isso dificulta a melhoria dos candidatos e também atrasa a convergência do algoritmo genético.

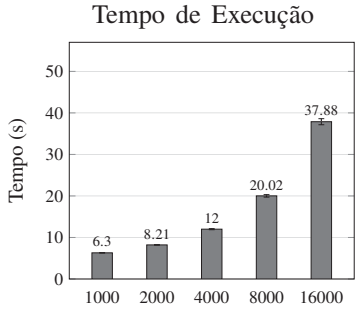


Figura 7.6: Tempo de Execução do GeSeMa (7 VNFs)

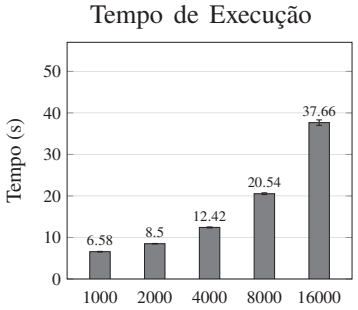


Figura 7.7: Tempo de Execução do GeSeMa (9 VNFs)

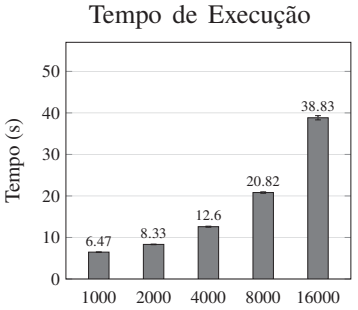


Figura 7.8: Tempo de Execução do GeSeMa (11 VNFs)

O terceiro experimento modifica a função objetivo empregada para mapear o serviço de rede. Estas modificações não implicam em nenhuma alteração no algoritmo. Todas as modificações são feitas no documento de requisição de serviço através do objeto `METRICS`. Foram consideradas três funções objetivo diferentes: (i) a função objetivo padrão com a maximização da densidade de usuários e da distância geográfica entre caches vizinhos; (ii) uma função mono-objetivo com a maximização da densidade de usuários; e (iii) uma função mono-objetivo com a maximização da distância geográfica entre caches vizinhos. Neste experimento, nosso critério de parada é baseado em tempo, com o laço principal do GeSeMa sendo executado por 10 segundos.

Tabela 7.1: Modificação da Função Objetivo

	Maximização		
	Densidade de Usuários Distância Geográfica	**Densidade de Usuários**	**Distância Geográfica**
Densidade de Usuários	129261	141176	156314.20
Distância Geográfica	144831.67	(42313.69, 141176)	(156314.20, 106576)
Densidade de Usuários	–	150153	–
Distância Geográfica	–	–	162025.35

Os resultados do terceiro experimento são mostrados na Tabela 7.1. A primeira coluna mostra os resultados para a configuração multiobjetivo com a melhor relação custo-benefício e o mesmo peso (ou seja, igual a 0,5) para cada métrica de otimização específica. A segunda coluna mostra o melhor candidato na fronteira de Pareto para o problema de otimização multiobjetivo, mas considerando apenas a densidade de usuários. Além disso, apresentamos o melhor resultado para a avaliação da densidade de usuários mono-objetivo na segunda linha da mesma coluna. A última coluna mostra o melhor candidato na fronteira de Pareto para o problema de otimização multiobjetivo, levando em consideração apenas a distância geográfica na primeira linha. Também apresentamos o melhor resultado para a terceira configuração de avaliação (distância geográfica mono-objetivo) na terceira linha da terceira coluna. Na segunda e terceira colunas, para a linha de configuração da avaliação multiobjetivo, os dados entre parênteses representam os resultados de ambas as métricas empregadas para avaliar o candidato: (distância geográfica, densidade de usuários).

Os resultados demonstram a capacidade do GeSeMa em lidar com diferentes configurações de avaliação apenas modificando o documento de solicitação de serviço. Com

certeza, quanto mais específica for a configuração da avaliação, melhores serão os candidatos ao mapeamento resultante. Assim, no experimento, se apenas a densidade de usuários for relevante para algum problema específico, é melhor empregar otimização mono-objetivo. O mesmo ocorre para a distância geográfica entre caches vizinhos. No entanto, as otimizações multiobjetivo são as melhores opções quando a configuração da avaliação depende de várias métricas. Assim, posteriormente é possível avaliar a fronteira de Pareto retornada de acordo com alguns critérios, como o uso de pesos.

Os próximos experimentos comparam os resultados do GeSeMa com duas soluções alternativas de mapeamento baseadas em busca heurística: busca aleatória e busca gulosa k-estocástica. A busca aleatória randomiza os domínios que irão alocar as funções de rede de uma determinada topologia de serviço. Esta solução deve garantir a criação de candidatos válidos em relação às dependências de domínio e restrições de topologia de rede. A busca gulosa k-estocástica randomiza $k \geq 1$ domínios que possivelmente podem alocar uma função de rede de uma topologia de serviço. Então, a heurística usa uma estratégia gulosa para encontrar a melhor opção local considerando a configuração da avaliação. Semelhante à busca aleatória, o algoritmo guloso k-estocástico também deve garantir que apenas candidatos válidos sejam criados, com relação às dependências de domínio e restrições de topologia de rede. Ambas as soluções foram adaptadas para processar o mesmo documento de requisição de serviço utilizado pelo GeSeMa, permitindo assim a customização de suas funções de avaliação e uma comparação justa com o próprio GeSeMa.

Todas as soluções foram configuradas para executar seu laço principal por 10 segundos. O laço principal consiste na criação e avaliação de indivíduos. Assim, não são considerados no tempo de execução outras rotinas internas, como processamento de requisições, instanciação de grafos e registro de saída. Outras configurações relevantes são as seguintes. Para o GeSeMa executando o algoritmo SPEA2: emprega-se uma taxa de cruzamento SBX de 30%, uma taxa de mutação por substituição de 5%, a seleção foi feita com um torneio binário, a população inicial é aleatória e o tamanho da população é de 50 indivíduos. Nas Figuras, os resultados para a busca aleatória clássica são rotulados como “Aleatório”; busca gulosa k-estocástica com $k = 2$ e $k = 4$ são rotulados com “S2-Guloso” e “S4-Guloso”, respectivamente. Cada execução das soluções de mapeamento retorna a fronteira de Pareto local, que depois é utilizada para calcular as fronteiras de Pareto relativas (*i.e.*, os candidatos na fronteira de Pareto relativa dominam todos os outros candidatos nas fronteiras de Pareto locais encontradas em qualquer execução de qualquer algoritmo).

O quarto experimento apresenta a média das fronteiras de Pareto relativas dos candidatos retornados pelas soluções de mapeamento. As figuras 7.9, 7.10 e 7.11 mostram esses resultados para as topologias com, respectivamente, 7, 9 e 11 funções. As barras cinza mostram os resultados obtidos para todos os candidatos nas fronteiras relativas (caso “Completo”), enquanto as barras brancas mostram os resultados para os dez primeiros candidatos das fronteiras relativas de cada solução (caso “Top 10”). O GeSeMa aplicado para serviços de mapeamento consistindo de 7 e 9 funções de rede apresentou a melhor média relativa da fronteira de Pareto para ambos os casos “Completo” e “Top 10”. Para o mapeamento de 11 funções, o GeSeMa apresentou um pior resultado para o caso “Completo” em comparação com S4-Guloso. No entanto, para 11 funções, o GeSeMa ainda alcança melhores resultados que o S4-Guloso no caso “top 10”. A degradação do GeSeMa no caso “Completo” ocorre devido ao maior número de candidatos nas fronteiras de Pareto (373 candidatos) em comparação com o número de candidatos do S4-Guloso (200 candidatos). Assim, os melhores candidatos gerados pelo GeSeMa são melhores que os melhores candidatos gerados pelo S4-Guloso (ver caso “Top 10”), mas o GeSeMa retorna mais resultados que incluem candidatos menos ajustados do que aqueles retornados de S4-Guloso. O

número total de candidatos recuperados de cada solução de mapeamento durante os experimentos é apresentado na Tabela 7.2. Outra observação relevante é que entre os dez primeiros candidatos, apenas GeSeMa apresentou todos os candidatos na fronteira relativa de Pareto (*i.e.*, 0). Este fato mostra a eficiência e estabilidade da solução proposta para mapear a topologia do serviço mesmo com a variação do número de funções.

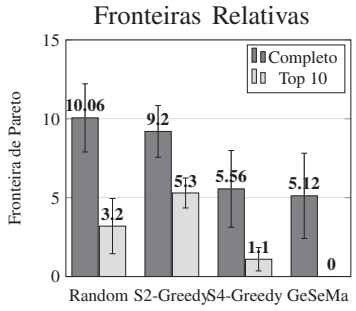


Figura 7.9: Comparação de Fronteiras (7 VNFs)

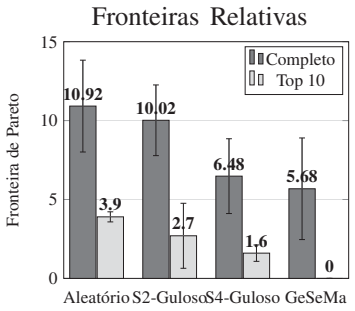


Figura 7.10: Comparação de Fronteiras (9 VNFs)

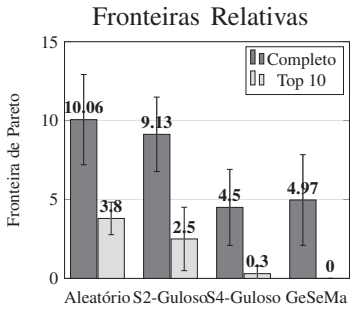


Figura 7.11: Comparação de Fronteiras (11 VNFs)

Tabela 7.2: Número de Candidatos Recuperados das Soluções de Mapeamento

	Aleatório	**S2-Guloso**	**S4-Guloso**	**GeSeMa**
7 Funções	291	241	225	353
9 Funções	338	299	312	422
11 Funções	274	242	200	373

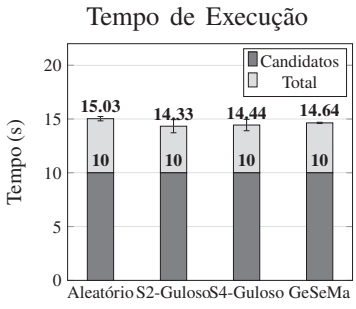


Figura 7.12: Comparação do Tempo de Execução (7 VNFs)

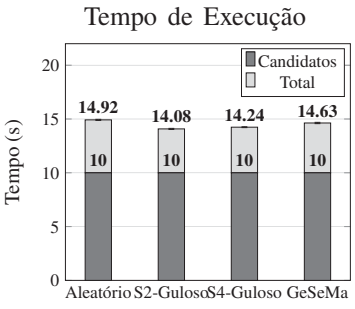


Figura 7.13: Comparação do Tempo de Execução (9 VNFs)

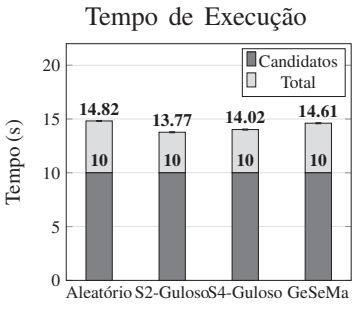


Figura 7.14: Comparação do Tempo de Execução (11 VNFs)

Por fim, os últimos experimentos comparam o tempo total de execução (em segundos) das soluções para mapear topologias com 7 (Figura 7.12), 9 (Figura 7.13) e 11 (Figura 7.14). Observa-se que 10 segundos do tempo total de execução são reservados para executar o laço principal de cada uma das soluções. O tempo extra gasto pelas várias soluções para executar operações internas, como processamento de solicitações, criação de gráficos, instanciação de objetos e gravação de saída, foi muito semelhante. As diferenças dos tempos totais de execução não ultrapassaram 1,05 segundos - 7,6% - mesmo no pior cenário (mapeamento de 11 funções, que apresentou a maior diferença entre S2-Guloso e Aleatório). Em particular, o GeSeMa apresenta tempos de execução ligeiramente menores que o algoritmo Aleatório (0,39, 0,29 e 0,21 segundos para mapear 7, 9 e 11 funções, respectivamente) e tempos de execução ligeiramente maiores que o S2-Guloso (0,31, 0,55 e 0,84 segundos para mapear 7, 9 e 11 funções) e S4-Guloso (0,2, 0,39 e 0,59 para 7, 9 e 11 funções, respectivamente).

7.3.2 Comparação Entre GeSeMa e GA+LCB

O GA+LCB é uma solução de mapeamento baseada em um algoritmo genético mono-objetivo (Li et al., 2020). Além do processo de mapeamento tradicional (mapeamento das principais funções de rede de um serviço de rede), o GA+LCB inclui um mecanismo de mapeamento de *backup* que cria um esquema de recuperação para o serviço de rede solicitado. Porém, como o GeSeMa não cria *backups*, para efeito de comparação, o GA+LCB é executado para mapear apenas as funções principais, e não os *backups*. A função objetivo GA+LCB foi configurada como uma maximização da importância do domínio modificado (imp_k de (Li et al., 2020)), que consiste na maximização de três métricas – disponibilidade de link (da_k), disponibilidade de largura de banda (dc_k) e o fator de disponibilidade (A_k) – e a minimização de uma única métrica – atraso entre domínios (dd_k). A solução GA+LCB calcula essa configuração de avaliação como $E = w1 * nor(da_k) + w2 * nor(dc_k) + w3 * nor(A_k) + w4 * (1 - nor(dd_k))$, onde nor indica uma função de normalização e w_n o peso da métrica ($\sum_{n=1}^{4} w_n = 1$). O GeSeMa, por sua vez, avalia os candidatos usando as mesmas métricas e objetivos, mas utilizando a estratégia de fronteiras de Pareto.

Neste estudo de caso, o GeSeMa e o GA+LCB são empregados para mapear um serviço de rede com 9 funções genéricas de rede. A topologia da rede é a mesma composta por 114 domínios globais da AWS usados nos experimentos da Subseção 7.3.1. O mapeamento das funções de rede não deve exceder os limites de recursos computacionais dos domínios, e não mais do que duas funções de rede devem ser mapeadas para cada domínio. Ambas as soluções foram configuradas para observar restrições de atraso máximo e disponibilidade mínima. Os valores das métricas dc_k e A_k são definidos aleatoriamente nos intervalos $[100, 500]$ e $[0,95, 0,99]$, respectivamente; o valor de da_k é 114 para todos os domínios (a topologia da rede é um grafo completo); e o valor de dd_k é definido considerando a distância geográfica entre pares de domínios $gd_{k,k+n}$ na curva $gd_{k,k+n} * (1 - e^{nor(gd_{k,k+n})*-4}) * 0,05$. Conforme exigido pelo GA+LCB, o domínio inicial e o domínio final são especificados no documento de requisição.

Os parâmetros genéticos de GA+LCB e GeSeMA foram configurados para serem os mais semelhantes possíveis. GA+LCB inclui um cruzamento de metade da população usando um algoritmo nativo. Assim, configuramos o GeSeMa com uma taxa de cruzamento de 0,5 usando o algoritmo SBX (SBX tem comportamento semelhante ao algoritmo de cruzamento GA+LCB). A taxa de mutação é definida como 0,05, o GA+LCB usa um algoritmo de mutação simples e específico; o GeSeMa usa um algoritmo de mutação de substituição. O GA+LCB executa uma seleção por roleta tradicional; O GeSeMa emprega um seletor por torneio binário. O GA+LCB cria a população inicial com base em um algoritmo de caminho k mais curto; O GeSeMa cria a população inicial aleatoriamente. O GA+LCB usa um algoritmo genético mono-objetivo projetado com recursos de elitismo; o GeSeMa adota SPEA2. O tamanho da população de 50 indivíduos foi o mesmo para ambas as soluções, assim como a execução de 20000 gerações. Os experimentos foram realizados na mesma máquina utilizada para o primeiro estudo de caso (Subseção 7.3.1). Todos os algoritmos, *scripts*, conjuntos de dados e requisições usados para comparar o GA+LCB com o GeSeMa estão disponíveis em https://github.com/ViniGarcia/NFV-FLERAS/tree/GesemaExperiments. Os experimentos foram repetidos 30 vezes, alcançando um nível de confiança de 95%.

O primeiro experimento compara a qualidade dos candidatos retornados pelo GeSeMa e GA+LCB. Usamos a média das fronteiras relativas de Pareto para a comparação. A Figura 7.15 mostra as fronteiras médias de candidatos retornados para dois casos: "Completo" (fronteiras de todos os candidatos de todas as execuções são usadas para calcular o valor médio) e "Top 10" (fronteiras de os dez principais candidatos de todas as execuções são usados para calcular o valor médio). A solução GA+LCB apresentou melhor média das fronteiras relativas no caso

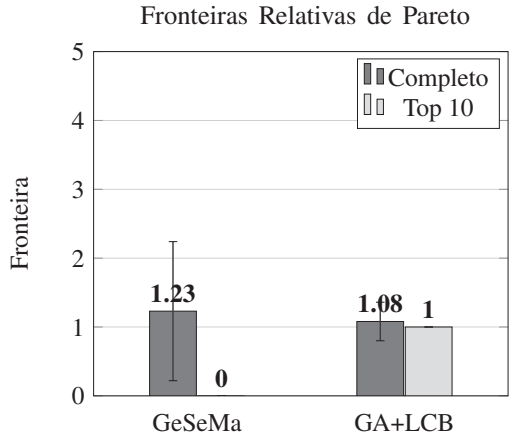


Figura 7.15: Comparação de Fronteiras (Genéticos)

"Completo". No entanto, GeSeMa supera os resultados de GA+LCB no experimento "Top 10". Esse comportamento ocorre devido ao número de candidatos retornados do GA+LCB a cada execução: exatamente um. Assim, GA+LCB retorna um total de 30 candidatos com o melhor valor *E* obtido em cada execução da solução. O GeSeMa, por sua vez, retorna toda a fronteira de Pareto, que normalmente contém vários candidatos. Neste experimento, a solução GeSeMa forneceu aproximadamente 49 candidatos por execução, com um total de 1463 candidatos avaliados no caso "Completo". Alguns destes candidatos não são mais bem avaliados do que os retornados pelo GA+LCB, mas, como demonstrado pelo caso "Top 10", os melhores candidatos do GeSeMa são mais ajustados aos objetivos considerados do que os melhores candidatos do GA+LCB.

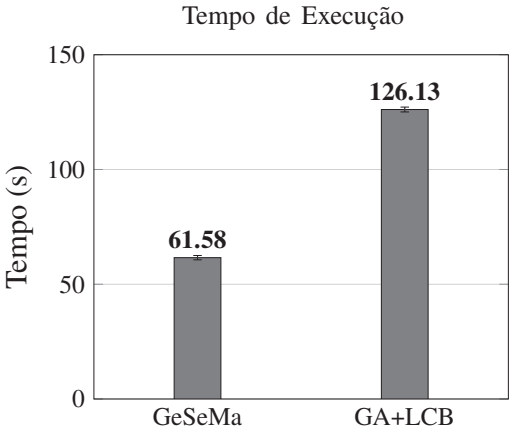


Figura 7.16: Comparação do Tempo de Execução (Genéticos)

O segundo experimento compara os tempos médios de execução do GA+LCB e GeSeMa para mapear o serviço na topologia de rede AWS. A Figura 7.16 mostra os resultados. O GeSeMa apresentou melhor tempo médio de execução, sendo 104% mais rápido que o GA+LCB. Esses resultados podem ser explicados da seguinte forma. Primeiro, o GeSeMa emprega uma estratégia de população inicial aleatória leve, enquanto GA+LCB usa uma heurística de k-menores caminhos para criar uma população inicial possivelmente otimizada. Assim, a estratégia GA+LCB requer a execução de algoritmos de caminho mais curto que requerem um maior tempo para serem executados em grandes topologias de rede. Em segundo lugar, a avaliação de múltiplas métricas de otimização com um algoritmo genético mono-objetivo requer um índice agregado (em GA+LCB,

chamado E). A criação deste índice impõe um tempo extra para processar a normalização e ponderação exigida por cada geração. Em terceiro lugar, o GA+LCB não possui nenhum mecanismo para evitar a avaliação de candidatos que já foram descartados, mas reaparecem durante a execução do algoritmo genético. O GeSeMa, por sua vez, usa uma lista tabu para ignorar esses candidatos.

7.4 SUMARIZAÇÃO E DISCUSSÃO

As limitações relacionadas ao gerenciamento no contexto do paradigma NFV não são restritas às funções de rede, suas plataformas e sistemas auxiliares. Na verdade, ainda existem questões de gerenciamento em aberto em diversos cenários, como na implantação de serviços virtualizados de rede no substrato físico. Essas limitações estão relacionadas principalmente na limitação de gerentes e operadores de rede em personalizar o processo de implantação de forma que ele atenda por completo as suas necessidades de otimização, além de considerar outras restrições que potencialmente ocorrem, como ao se trabalhar com cenários híbridos de serviços de rede onde funções físicas e virtualizadas coexistem. Apesar de todas as tarefas de implantação (composição, integração e programação da execução) e suas técnicas em específico apresentarem tais limitações, em diferentes graus, a tarefa de integração através da técnica de mapeamento multidomínio é uma das menos exploradas neste contexto.

Sendo assim, este capítulo apresentou uma solução baseada em uma heurística genética que aborda o problema de personalização da execução da integração por mapeamento multidomínio. Essa solução, chamada GeSeMa, utiliza um modelo de requisição pelo qual o seu operador pode definir uma série de parâmetros de configuração. Esses parâmetros permitem definir topologias de serviço genéricas, diferentes tipos de dependências (entre elas, as de domínio), métricas a serem avaliadas como parte da função objetivo, além do modelo de avaliação de tais métricas. Essa flexibilidade operacional permite a criação de funções objetivo capazes de representar necessidades únicas e personalizadas. Também, a opção pelo uso de heurísticas genéticas habilita o operador da solução a determinar uma série de algoritmos internos, como aqueles relacionados a seleção, cruzamento e mutação, além do próprio algoritmo genético base a ser empregado. Por fim, a solução proposta possibilita que os operadores indiquem parâmetros de execução do algoritmo genético, como taxa de cruzamento e mutação, tamanho da população e critério de parada. Isso permite que o GeSeMa se adapte a cenários de execução heterogêneos, se ajustando, por exemplo, a diferentes disponibilidades de tempo de execução e recursos computacionais.

Dois estudos de caso foram conduzidos para verificar a viabilidade da solução proposta e suas capacidades em se adequar à avaliação de diferentes funções objetivo e topologias de serviços virtualizados. O primeiro estudo de caso consiste no mapeamento de um serviço de caches hierárquicas em uma topologia de rede inspirada na Amazon AWS. Este estudo de caso busca avaliar a convergência e o tempo de execução do GeSeMa, além de comparar a solução proposta com outras soluções de mapeamento baseadas em heurísticas clássicas. Já o segundo estudo de caso almeja comparar o GeSeMa com uma solução estado da arte, chamada GA+LCB. Neste caso, a mesma topologia de rede (Amazon AWS) foi utilizada, sendo mapeada uma cadeia de serviço genérica com 9 funções de rede. Uma vez que ambas soluções utilizam heurísticas genéticas, o GeSeMa foi executado considerando configurações e restrições iguais o muito similares aquelas adotadas pelo GA+LCB. Os resultados obtidos demonstraram a flexibilidade do GeSeMa em se adequar a diferentes cenários de execução, além de sua eficiência em gerar mapeamentos de serviços otimizados, superando, nos casos testados, as demais soluções consideradas.

8 CONCLUSÃO

As tecnologias de virtualização de rede iniciaram uma revolução na área. Essas tecnologias permitem, por exemplo, substituir equipamentos dedicados, chamados de *physical appliances*, para soluções de *software* executando em equipamentos de propósito geral. Para tanto, estratégias de virtualização existentes, como virtualização completa, paravirtualização, conteinerização e virtualização baseada em processos, são usualmente empregadas. O paradigma NFV é um dos paradigmas que constitui o arcabouço das redes virtualizadas. Através da substituição completa dos equipamentos físicos por instâncias virtualizadas, a tecnologia NFV objetiva flexibilizar a rede, além de reduzir os custos de capital e operação. Esforços de padronização do paradigma NFV são constantemente conduzidos, tendo entre os mais proeminentes aqueles de instituições como ETSI e IETF. Entre esses esforços, destaca-se a criação da arquitetura de referência do paradigma NFV pela ETSI, a qual é amplamente adotada. Essa arquitetura é composta de três domínios de trabalho: VI, NFV-MANO e VNF. Especialmente, o domínio de VNF compreende dois elementos operacionais: a própria VNF, elemento responsável pela execução de uma NF em uma plataforma de execução; e o EMS, elemento dedicado ao gerenciamento direto de uma ou mais instâncias de VNF. Evidencia-se que os domínios de trabalho de VI e NFV-MANO têm recebido grande atenção recentemente. Porém, o domínio de VNF tem sido pouco explorado, não conta com arquiteturas internas para seus elementos operacionais e a implementação de tais elementos não incluem funcionalidades inovadoras em NFV.

As lacunas que podem ser caracterizadas no domínio de trabalho de VNF resultam em grandes dificuldades de gerência de um ambiente NFV. Ressalta-se que uma diversidade de elementos de gerência são previstos na arquitetura NFV. Entre esses elementos estão presentes no contexto do NFV-MANO e aqueles que residem no domínio de trabalho de VNF. Entretanto, devido à falta de ferramentas adequadas, operações de gerência normalmente providas em sistemas NFV-MANO nem sempre podem ser aplicadas nas instâncias de VNF. Esse fenômeno ocorre por dois motivos principais. O primeiro se refere ao fato de que as plataformas de execução de VNF disponíveis não preveem operações para a gerência interna de funções de rede ou o fazem de forma muito limitada. O segundo motivo consiste na ausência do EMS nos ambientes NFV, diminuindo as capacidades de gerência das instâncias de VNF. Como consequência, sistemas NFV-MANO disponíveis atualmente limitam-se a gerenciar a instância virtualizada que suporta uma VNF, não abordando o ciclo de vida interno da VNF propriamente dita.

Nesse contexto, este trabalho objetiva especificar os elementos operacionais do domínio de trabalho de VNF, propondo arquiteturas de referência para plataformas de execução de VNF e para o EMS. Adicionalmente, objetiva-se sugerir modelos de implementação, desenvolver protótipos baseados nas arquiteturas propostas, realizar um conjunto de testes de avaliação das arquiteturas pelos seus protótipos e salientar cenários práticos de aplicação. Com isso, além de estabelecer meios de gerenciamento holístico do domínio de trabalho de VNF, objetiva-se dispor ferramentas que possam auxiliar na concretização de ambientes NFV completos e totalmente gerenciáveis, aptos de atender os requisitos do paradigma NFV (*i.e.*, portabilidade, desempenho, integração, gerência/orquestração e escalabilidade).

Considerando objetivos propostos, este trabalho apresentou uma arquitetura interna para plataformas de execução de VNF, além de especificar cada um de seus módulos operacionais internos e pontos de conexão entre os mesmos. Essa arquitetura prevê um módulo operacional especialmente dedicado ao gerenciamento das plataformas, o MA. Também, essa arquitetura

flexibiliza a utilização e permite a configuração de diferentes ferramentas para acesso à rede e *frameworks* para processamento de pacotes. Além disso, a arquitetura compreende o suporte a funcionalidades inovadoras no contexto do paradigma NFV, a exemplo da fragmentação de funções de rede em componentes, bem como a sua gerência. Além do citado, é habilitado que o processamento NSH seja executado nativamente através das plataformas implementadas em conformidade com a arquitetura. Uma plataforma protótipo, chamada COVEN, concretizou a arquitetura proposta como uma prova de conceito. Essa plataforma foi avaliada e obteve sucesso nos testes executados.

Também, uma arquitetura para o elemento EMS foi apresentada. O EMS fornece um sistema de gerência que permite a comunicação direta com instâncias de VNF, em específico com o MA de suas plataformas de execução. Essa comunicação pode ser iniciada de forma espontânea uma vez configurado, por exemplo, um gatilho disparado por um agente de monitoramento do EMS, ou ser requisitada por um agente externo, como VNFM ou OSS/BSS. A arquitetura propicia, através dos módulos AS e AIf, a íntegra das operações previstas no protocolo Ve-Vnfm-em da ETSI, garantindo completa compatibilidade com sistemas já existentes que o implementam. Entretanto, mesmo atuando em diferentes frentes, a tarefa fundamental de um EMS corresponde à criação de uma interface unificada para acesso e gerenciamento de plataformas de execução de VNF heterogêneas, estas disponibilizando operações e empregando tecnologias e protocolos de comunicação diferentes entre si. Sendo assim, ao reconhecerem somente um protocolo (Ve-Vnfm-em) e comunicarem através de uma tecnologia única (a utilizada no AIf do EMS), os agentes externos podem operar internamente em qualquer instância de VNF presente em um ambiente NFV. A viabilidade de implementação da arquitetura proposta foi evidenciada no protótipo HoLMES. O protótipo foi avaliado em um estudo de caso que consistiu da execução de operações típicas do gerenciamento de uma VNF através do EMS para três plataformas de execução distintas. Os resultados atestaram as funcionalidades do EMS, também evidenciando uma sobrecarga no tempo de execução das operações decorrente do seu uso considerada satisfatória para uma vasta gama de cenários.

No decorrer deste trabalho foram discutidas oportunidades de aplicação das arquiteturas e protótipos desenvolvidos. Entre as oportunidades, destaca-se o aprofundamento do monitoramento das funções de rede virtualizadas, propiciando a utilização de métricas de caixa-branca. Igualmente, destaca-se o suporte ao compartilhamento de uma VNF, sendo possível uma mesma instância servir a vários clientes. Por fim, também é considerada a oportunidade de aprimoramento de emuladores NFV, disponibilizando funcionalidades inovadoras relacionadas à arquitetura de plataformas de execução de VNF. Mais, busca-se aprimorar o gerenciamento das funções de rede em emuladores pelo uso de soluções baseadas na arquitetura de EMS proposta.

Ainda em relação aos objetivos principais deste trabalho, uma metodologia de definição e parametrização de funções objetivo e modelos de avaliação para a tarefa de integração de serviços virtualizados de rede no substrato físico foi apresentada. Esta metodologia opera através de um documento de requisição, o qual descreve o conjunto de métricas que devem ser avaliadas, assim como provê os dados necessários para a realização desta avaliação. Esse modelo foi aplicado no contexto específico da técnica de mapeamento multidomínio, o que gerou uma solução parametrizável baseada em metaheurísticas genéticas, chamada GeSeMa. Além de permitir a parametrização da função objetivo e do modelo de avaliação, o GeSeMa também habilita seus usuários a determinar parâmetros relacionados aos próprios algoritmos genéticos, como operadores, taxa de mutação e cruzamento, tamanho de população e quantidade de gerações. Com isso, essa solução pode ser aplicada em diferentes cenários, se adequando, por exemplo, ao tempo de execução ou quantidade de recursos computacionais disponíveis. A solução foi testada e comparada com diversas outras baseadas em heurísticas clássicas e em heurísticas genéticas.

Os resultados dos estudos de caso considerados demonstraram a viabilidade e flexibilidade do GeSeMa, sendo capaz de se adequar e avaliar diversas funções objetivo e obtendo resultados de mapeamento e um tempo de execução competitivo às demais alternativas.

Trabalhos futuros incluem o aprofundamento da pesquisa relacionada aos cenários de aplicação das arquiteturas e protótipos de VNF e EMS propostos. O objetivo principal consiste em explorar desafios técnicos relativos ao monitoramento de funções de rede; detecção de gargalos; identificação e mitigação de ataques; segurança para compartilhamento de funções; utilização de serviços compartilhados em ambientes federados; estratégias para achatamento da curva de aprendizagem do paradigma NFV através de sistemas intuitivos de emulação e simulação; e o desenvolvimento de materiais didáticos para a popularização do paradigma. Além dos aspectos tecnológicos e acadêmicos, também é objetivo aplicar NFV, na prática, em projetos e ações que resultem em benefícios imediatos no contexto socioeconômico brasileiro. Dentre os projetos nessa esfera estão a utilização do paradigma NFV para a criação de redes didáticas locais, além do emprego deste para habilitar redes metropolitanas de comunicação de forma rápida e eficiente.

Já no contexto da integração personalizável de serviços de rede, em trabalhos futuros, pretende-se tornar o GeSeMa dinâmico e on-line, evoluindo continuamente o mapeamento de serviços e sugerindo migrações em tempo real, tanto na topologia da rede quanto na configuração de objetivos e avaliação. Desta forma, a solução conseguirá enfrentar o dinamismo dos dispositivos em redes específicas, como 5G e IoT. Além disso, busca-se um GeSeMa adaptativo capaz de funcionar efetivamente em cenários catastróficos e em redes de campo de batalha, onde os recursos disponíveis são instáveis e podem mudar a qualquer momento. Por fim, aproveitando a flexibilidade da personalização da configuração de avaliação, pretende-se ainda desenvolver uma versão de serviço do GeSeMa para ser explorada no contexto de *marketplaces* NFV, como FENDE (Bondan et al., 2019) e T-NOVA (Xilouris et al., 2014).

REFERÊNCIAS

Abujoda, A. e Papadimitriou, P. (2016). Distnse: Distributed network service embedding across multiple providers. Em *IEEE Int. Conf. on Communication Systems and Networks*, páginas 1–8.

Al-Salim, A. M., Lawey, A. Q., El-Gorashi, T. E. H. et al. (2017). Energy efficient big data networks: Impact of volume and variety. *Transactions on Network and Service Management*, 15(1):458–474.

Antonenko, V., Smeliansky, R. e Plakunov, A. (2018). Cube: Multi-user virtual function life-cycle orchestration technique. Em *International Scientific and Technical Conference Modern Computer Network Technologies*, páginas 1–8, Moscou, Rússia.

Benzekki, K., El Fergougui, A. e Elbelrhiti Elalaoui, A. (2016). Software-defined networking (sdn): a survey. *Security and Communication Networks*, 9(18):5803–5833.

Berlin, G. (2021). Open baton documentation. `https://openbaton.github.io/documentation/`. Acessado em 11/01/2021.

Bondan, L., dos Santos, C. R. P. e Granville, L. Z. (2014). Management requirements for clickos-based network function virtualization. Em *International Conference on Network and Service Management*, páginas 447–450, Rio de Janeiro, Brasil.

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

Boubendir, A. et al. (2016). Naas architecture through sdn-enabled nfv: Network openness towards web communication service providers. Em *IEEE/IFIP Network Operations and Management Symposium*, páginas 722–726.

Canonical (2021a). cloud-init: the industry standard multi-distribution method for cross-platform cloud instance initialization. `https://cloudinit.readthedocs.io/en/latest/`. Acessado em 04/02/2021.

Canonical (2021b). Ubuntu in the cloud. `https://ubuntu.cloud`. Acessado em 26/01/2021.

Cao, J. et al. (2016). Vnf placement in hybrid nfv environment: Modeling and genetic algorithms. Em *IEEE Int. Conf. on Parallel and Distributed Systems*, páginas 769–777.

Carpio, F. et al. (2017). Vnf placement with replication for load balancing in nfv networks. Em *IEEE Int. Conf. on Communications*, páginas 1–6.

Cavalca, U., Mesquita, C., Pereira, A. C. e Carrano, E. G. (2014). A methodology for traffic shaping optimization in next generation networks. *International Journal of Computer Information Systems and Industrial Management Applications*, 6:474–483.

Chen, Q., Wang, H. e Liu, N. (2019). Integrating networking, storage, and computing for resilient battlefield networks. *IEEE Communications Magazine*, 57(8):56–63.

Costan, V. e Devadas, S. (2016). Intel sgx explained. *Cryptology ePrint Archive*, 2016(86):1–118.

da Cruz Marcuzzo, L., Fulber-Garcia, V., Cunha, V., Corujo, D., Barraca, J. P., Aguiar, R. L., Schaeffer-Filho, A. E., Granville, L. Z. e dos Santos, C. R. P. (2012). Dynamic interoperability between multi-tenant saas applications. Em *International Conference on Interoperability for Enterprise Systems and Applications*, páginas 217–226, Valência, Espanha.

Daniel-Simion, D. e Dan-Horia, G. (2011). Traffic shaping and traffic policing impacts on aggregate traffic behaviour in high speed networks. Em *IEEE International Symposium on Applied Computational Intelligence and Informatics*, páginas 465–467. IEEE.

Das, P., Rahnamay-Naeini, M., Ghani, N. e Hayat, M. M. (2017). On the vulnerability of multi-level communication network under catastrophic events. Em *2017 International Conference on Computing, Networking and Communications (ICNC)*, páginas 912–916. IEEE.

Deb, K., Pratap, A., Agarwal, S. e Meyarivan, T. (2002). A fast and elitist multiobjective genetic algorithm: Nsga-ii. *IEEE Transactions on Evolutionary Computation*, 6(2):182–197.

Dietrich, D., Abujoda, A. e Papadimitriou, P. (2015). Network service embedding across multiple providers with nestor. Em *IFIP Networking*, páginas 1–9.

Dino, L. A. e Costa, D. (2021). Uso da internet por crianças e adolescentes no brasil: dinâmicas e desafios. *RE@ D-Revista de Educação a Distância e Elearning*, 4(1):25–41.

Dong, L. e Clemm, A. (2021). High-precision end-to-end latency guarantees using packet wash. Em *IFIP/IEEE International Symposium on Integrated Network Management*, páginas 259–267.

DPDK, L. F. (2021). Dpdk: Data plane development kit. `https://www.dpdk.org`. Acessado em 26/01/2021.

Dräxler, S., Karl, H., Peuster, M. et al. (2017). Sonata: Service programming and orchestration for virtualized software networks. Em *International Conference on Communications Workshops*, páginas 973–978, Paris, França.

Ehlert, S., Petgang, S., Magedanz, T. et al. (2006). Analysis and signature of skype voip session traffic. Em *International Association of Science and Technology for Development Conference*, páginas 1–13, Lanzarote, Espanha.

ETSI, E. T. S. I. (2021). Open source mano documentation. `https://osm.etsi.org/docs/user-guide/`. Acessado em 11/01/2021.

Evans, J. W. e Filsfils, C. (2010). *Deploying IP and MPLS QoS for multiservice networks: theory and practice*. Elsevier.

Finkbeiner, B., Hahn, C., Stenger, M. et al. (2019). Monitoring hyperproperties. *Formal methods in system design*, 54(3):336–363.

Flauzino, J. W. V., Fulber-Garcia, V., Souza, G. V. d. et al. (2020). Além do openstack: Disponibilizando o suporte para funções virtualizadas de rede nfv-mano no cloudstack. Em *Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos*, páginas 435–448, Rio de Janeiro, Brasil.

Fontanini, M. (2021). libtins: Packet crafting and sniffing library. `http://libtins.github.io`. Acessado em 08/03/2021.

Fulber-Garcia, V., Gaiardo, G. d. F., Marcuzzo, L. d. C. et al. (2018). Demons: A ddos mitigation nfv solution. Em *International Conference on Advanced Information Networking and Applications*, páginas 769–776, Cracóvia, Polônia.

Fulber-Garcia, V., Huff, A., Santos, C. R. P. d. et al. (2020a). Network service topology: Formalization, taxonomy and the custom specification model. *Computer Networks*, 178:107337.

Fulber-Garcia, V., Luizelli, M. C., dos Santos, C. R. P. e Duarte, E. P. (2020b). Cusco: A customizable solution for nfv composition. Em *Int. Conf. on Advanced Information Networking and Applications*, páginas 204–216.

Fulber-Garcia, V., Marcuzzo, L. d. C., Huff, A. et al. (2019a). On the design of a flexible architecture for virtualized network function platforms. Em *Global Communications Conference*, páginas 1–6, Waikoloa, EUA.

Fulber-Garcia, V., Marcuzzo, L. d. C., Souza, G. V. et al. (2019b). An nsh-enabled architecture for virtualized network function platforms. Em *Conference on Advanced Information Networking and Applications*, páginas 376–387, Matsue, Japão.

Fulber-Garcia, V., Tavares, T., Marcuzzo, L. et al. (2020c). On the design and development of emulation platforms for nfv-based infrastructures. *International Journal of Grid and Utility Computing*, 11(2):230–242.

Gallo, M., Ghamri-Doudane, S. e Pianese, F. (2018). Climbos: A modular nfv cloud backend for the internet of things. Em *International Conference on New Technologies, Mobility and Security*, páginas 1–5, Paris, França.

Garzarella, S. (2021). ptnetmap: a netmap passthrough for virtual machines. `https://github.com/stefano-garzarella/ptnetmap`. Acessado em 29/01/2021.

Gelberger, A., Yemini, N. e Giladi, R. (2013). Performance analysis of software-defined networking (sdn). Em *International Symposium on Modelling, Analysis and Simulation of Computer and Telecommunication Systems*, páginas 389–393. IEEE.

Ghaznavi, M., Jalalpour, E., Wong, B. et al. (2020). Fault tolerant service function chaining. Em *Annual conference of the ACM Special Interest Group on Data Communication on the Applications, Technologies, Architectures, and Protocols for Computer Communication*, páginas 198–210.

Gonzalez, A. J., Nencioni, G., Kamisinski, A. et al. (2018). Dependability of the nfv orchestrator: State of the art and research challenges. *IEEE Communications Surveys & Tutorials*, 20(4):3307–3329.

Gu, S., Li, Z., Wu, C. et al. (2016). An efficient auction mechanism for service chains in the nfv market. Em *International Conference on Computer Communications*, páginas 1–9, San Francisco, EUA.

Guerassimov, S., Cordeiro, E. e Schwaighofer, L. (2016). Nfv and opnfv. Em *Seminars Future Internet and Innovative Internet Technologies and Mobile Communications*, páginas 55–61, Munique, Alemanha.

Guerzoni, R. et al. (2012). Network functions virtualisation: an introduction, benefits, enablers, challenges and call for action, introductory white paper. Em *SDN and OpenFlow World Congress*, volume 1, páginas 5–7.

Gupta, V., Devar, S. K., Kumar, N. H. et al. (2017). Modelling of iot traffic and its impact on lorawan. Em *Global Communications Conference*, páginas 1–6, Singapura.

Gurevich, Y. e Huggins, J. K. (1992). The semantics of the c programming language. Em *International Workshop on Computer Science Logic*, páginas 274–308, San Miniato, Itália.

Halpern, J. M. e Pignataro, C. (2015). Service function chaining (sfc) architecture. Relatório Técnico RFC 7665, Internet Engineering Task Force.

Hamdan, M., Mohammed, B., Humayun, U., Abdelaziz, A., Khan, S., Ali, M. A., Imran, M. e Marsono, M. N. (2020). Flow-aware elephant flow detection for software-defined networks. *IEEE Access*, 8:72585–72597.

Han, S., Jang, K., Park, K. et al. (2010). Packetshader: A gpu-accelerated software router. *SIGCOMM Computer Communication Review*, 40(4):195–206.

Hanan, A. H. A., Idris, M. Y., Kaiwartya, O. et al. (2017). Real traffic-data based evaluation of vehicular traffic environment and state-of-the-art with future issues in location-centric data dissemination for vanets. *Digital Communications and Networks*, 3(3):195–210.

Handley, M. (2006). Why the internet only just works. *BT Technology Journal*, 24(3):119–129.

Hasançebi, O. e Erbatur, F. (2000). Evaluation of crossover techniques in genetic algorithm based optimum structural design. *Computers & Structures*, 78(1-3):435–448.

Herrera, J. G. e Botero, J. F. (2016). Resource allocation in nfv: A comprehensive survey. *Transactions on Network and Service Management*, 13(3):518–532.

Hmaity, A., Savi, M. et al. (2016). Virtual network function placement for resilient service chain provisioning. Em *Workshop on Resilient Networks Design and Modeling*, páginas 245–252, Halmstad, Suécia.

Hu, F., Hao, Q. e Bao, K. (2014). A survey on software-defined network and openflow: From concept to implementation. *IEEE Communications Surveys & Tutorials*, 16(4):2181–2206.

Hwang, J., Ramakrishnan, K. K. e Wood, T. (2015). Netvm: High performance and flexible networking using virtualization on commodity platforms. *Transactions on Network and Service Management*, 12(1):34–47.

Imran, M., Said, A. e Hasbullah, H. (2010). A survey of simulators, emulators and testbeds for wireless sensor networks. Em *International Symposium on Information Technology*, páginas 897–902, Kuala Lumpur, Malásia.

Intel (2021). Nff-go: Network function framework for go. `https://github.com/intel-go/nff-go`. Acessado em 26/01/2021.

Jaeger, B. (2015). Security orchestrator: Introducing a security orchestrator in the context of the etsi nfv reference architecture. Em *Trustcom/BigDataSE/ISPA*, páginas 1255–1260, Tianjin, China.

Kaur, K., Mangat, V. e Kumar, K. (2020). A comprehensive survey of service function chain provisioning approaches in sdn and nfv architecture. *Computer Science Review*, 38:100298.

Khebbache, S. et al. (2018). A multi-objective non-dominated sorting genetic algorithm for vnf chains placement. Em *IEEE Annual Consumer Communications & Networking Conference*, páginas 1–4.

Kivity, A., Laor, D., Costa, G., Enberg, P., Har'El, N., Marti, D. e Zolotarov, V. (2014). Osv—optimizing the operating system for virtual machines. Em *USENIX Annual Technical Conference*, páginas 61–72, Philadelphia, EUA.

Kohler, E., Morris, R., Chen, B. et al. (2000). The click modular router. *Transactions on Computer Systems*, 18(3):263–297.

Kong, J., Kim, I., Wang, X., Zhang, Q., Cankaya, H. C., Xie, W., Ikeuchi, T. e Jue, J. P. (2017). Guaranteed-availability network function virtualization with network protection and vnf replication. Em *Global Communications Conference*, páginas 1–6. IEEE.

Lamport, L., Shostak, R. e Pease, M. (2019). The byzantine generals problem. Em *Concurrency: the Works of Leslie Lamport*, páginas 203–226.

Lantz, B., Heller, B. e McKeown, N. (2010). A network in a laptop: rapid prototyping for software-defined networks. Em *SIGCOMM Workshop on Hot Topics in Networks*, páginas 1–6, Nova Delhi, Índia.

Laufer, R., Gallo, M., Perino, D. e Nandugudi, A. (2016). Climb: Enabling network function composition with click middleboxes. *Computer Communication Review*, 46(4):17–22.

Li, G. e Wei, M. (2014). Everything-as-a-service platform for on-demand virtual enterprises. *Information Systems Frontiers*, 16(3):435–452.

Li, Y., Gao, L., Xu, S., Ou, Q., Yuan, X., Qi, F., Guo, S. e Qiu, X. (2020). Cost-and-qos-based nfv service function chain mapping mechanism. Em *IEEE Network Operations and Management Symposium*, páginas 1–9, Budapest, Hungary.

Ma, N. et al. (2017). A model based on genetic algorithm for service chain resource allocation in nfv. Em *IEEE Int. Conf. on Computer and Communications*, páginas 607–611.

Maji, S., Veeraraghavan, M., Buchanan, M., Alali, F., Ros-Giral, J. e Commike, A. (2017). A high-speed cheetah flow identification network function (cfinf). Em *IEEE Conference on Network Function Virtualization and Software Defined Networks*, páginas 1–7. IEEE.

Mandal, V., Mussah, A. R., Jin, P. et al. (2020). Artificial intelligence-enabled traffic monitoring system. *Sustainability*, 12(21):9177.

Marcuzzo, L. d. C., Fulber-Garcia, V., Cunha, V. et al. (2017). Click-on-osv: A platform for running click-based middleboxes. Em *Symposium on Integrated Network and Service Management*, páginas 885–886, Lisboa, Portugal.

Marku, E., Biczók, G. e Boyd, C. (2019). Towards protected vnfs for multi-operator service delivery. Em *Conference on Network Softwarization*, páginas 19–23, Paris, França.

Martins, J., Ahmed, M., Raiciu, C. et al. (2014). Clickos and the art of network function virtualization. Em *Symposium on Networked Systems Design and Implementation*, páginas 459–473, Seattle, EUA.

Masmoudi, F., Loulou, M. e Kacem, A. (2014). Multi-tenant services monitoring for accountability in cloud computing. Em *International Conference on Cloud Computing Technology and Science*, páginas 620–625, Singapura.

Matias, J., Garay, J., Toledo, N., Unzilla, J. e Jacob, E. (2015). Toward an sdn-enabled nfv architecture. *IEEE Communications Magazine*, 53(4):187–193.

MiniOS, X. P. H. (2021). Minios: Unikernel operating system. `https://wiki.xenproject.org/wiki/Mini-OS`. Acessado em 25/01/2021.

Mueller, F. (1993). A library implementation of posix threads under unix. Em *Usenix Winter*, páginas 29–42, San Diego, USA.

Naik, P., Kanase, A., Patel, T. et al. (2018). libvnf: Building virtual network functions made easy. Em *Symposium on Cloud Computing*, páginas 212–224, Carlsbad, EUA.

Naik, P., Shaw, D. K. e Vutukuru, M. (2016). Nfvperf: Online performance monitoring and bottleneck detection for nfv. Em *Conference on Network Function Virtualization and Software Defined Networks*, páginas 154–160, Palo Alto, EUA.

NFVISG, N. I. S. G. (2013). Network functions virtualisation (nfv); virtualisation requirements. Relatório Técnico GS NFV 004, European Telecommunications Standards Institute.

NFVISG, N. I. S. G. (2014). Network function virtualisation (nfv); architectural framework. Relatório Técnico GS NFV 002, European Telecommunications Standards Institute.

NFVISG, N. I. S. G. (2015). Network functions virtualisation (nfv); resiliency requirements. Relatório Técnico REL NFV 001, European Telecommunications Standards Institute.

NFVISG, N. I. S. G. (2016). Network functions virtualisation (nfv); management and orchestration; report on architectural options. Relatório Técnico IFA NFV 009, European Telecommunications Standards Institute.

NFVISG, N. I. S. G. (2018a). Network function virtualisation (nfv); terminology for main concepts in nfv. Relatório Técnico GS NFV 003, European Telecommunications Standards Institute.

NFVISG, N. I. S. G. (2018b). Network functions virtualisation (nfv); management and orchestration; report on management of nfv-mano and automated deployment of em and other oss functions. Relatório Técnico IFA NFV 021, European Telecommunications Standards Institute.

NFVISG, N. I. S. G. (2019). Network functions virtualisation (nfv); trust; report on certificate management. Relatório Técnico SEC NFV 005, European Telecommunications Standards Institute.

NFVISG, N. I. S. G. (2020a). Network functions virtualisation (nfv): Management and orchestration; ve-vnfm reference point; interface and information model specification. Relatório Técnico IFA NFV 008, European Telecommunications Standards Institute.

NFVISG, N. I. S. G. (2020b). Network functions virtualisation (nfv); management and orchestration; vnf descriptor and packaging specification. Relatório Técnico IFA NFV 011, European Telecommunications Standards Institute.

NFVISG, N. I. S. G. (2020c). Network functions virtualisation (nfv): Protocols and data models; restful protocols specification for the ve-vnfm reference point. Relatório Técnico SOL NFV 002, European Telecommunications Standards Institute.

NFVISG, N. I. S. G. (2020d). Network functions virtualisation (nfv); security; access token specification for api access. Relatório Técnico SEC NFV 022, European Telecommunications Standards Institute.

Nobre, J., Rosario, D., Both, C., Cerqueira, E. e Gerla, M. (2016). Toward software-defined battlefield networking. *IEEE Communications Magazine*, 54(10):152–157.

ntop pf_ring (2021). pf_ring: High-speed packet capture, filtering and analysis. `https://www.ntop.org/products/packet-capture/pf_ring`. Acessado em 17/02/2021.

Nucci, A. e Papagiannaki, K. (2009). *Design, measurement and management of large-scale IP networks: Bridging the gap between theory and practice*. Cambridge U. Press.

OpenOnload, X. (2021). Openonload: High performance user-level network stack. `https://github.com/Xilinx-CNS/onload`. Acessado em 17/02/2021.

OpenStack (2021). Openstack tacker documentation. `https://docs.openstack.org/tacker/latest/`. Acessado em 11/01/2021.

OPNFV, L. F. (2021). Samplevnf: The opnfv vnf project. `https://wiki.opnfv.org/display/SAM/SampleVNF+-+Home`. Acessado em 26/01/2021.

Ordonez-Lucena, J., Ameigeiras, P., Lopez, D. et al. (2017). Network slicing for 5g with sdn/nfv: Concepts, architectures, and challenges. *Communications Magazine*, 55(5):80–87.

Pearson, S. e Charlesworth, A. (2009). Accountability as a way forward for privacy protection in the cloud. Em *International Conference on Cloud Computing*, páginas 131–144, Bangalore, Índia.

Peuster, M., Karl, H. e Van Rossem, S. (2016). Medicine: Rapid prototyping of production-ready network services in multi-pop environments. Em *Conference on Network Function Virtualization and Software Defined Networks*, páginas 148–153, Palo Alto, EUA.

Qu, L., Assi, C., Shaban, K. e Khabbaz, M. (2016). Reliability-aware service provisioning in nfv-enabled enterprise datacenter networks. Em *International Conference on Network and Service Management*, páginas 153–159. IEEE.

Quinn, P., Elzur, U. e Pignataro, C. (2018). Network service header (nsh). Relatório Técnico RFC 8300, Internet Engineering Task Force.

Quinn, P. e Nadeau, T. (2015). Problem statement for service function chaining. Relatório técnico, Internet Engineering Task Force.

Red Hat (2021). Coreos: Container linux. `https://getfedora.org/en/coreos?stream=stable`. Acessado em 09/03/2021.

Resma, K. S., Sharvani, G. S. e Paul, A. T. (2019). A closer look at the network middleboxes. Em *International Conference on Advanced Technologies in Intelligent Control, Environment, Computing & Communication Engineering*, páginas 167–171, Bengaluru, India.

Richardson, M. e Wallace, S. (2012). *Getting started with raspberry PI.* "O'Reilly Media, Inc.".

Riera, J. F., Batallé, J., Bonnet, J., Días, M., McGrath, M., Petralia, G., Liberati, F., Giuseppi, A., Pietrabissa, A., Ceselli, A. et al. (2016). Tenor: Steps towards an orchestration platform for multi-pop nfv deployment. Em *IEEE Conf. on Network Softwarization*, páginas 243–250.

Rizzo, L. (2012). netmap: A novel framework for fast packet i/o. Em *USENIX Security Symposium*, páginas 101–112, Bellevue, EUA.

Rodis, P. e Papadimitriou, P. (2021). Intelligent network service embedding using genetic algorithms. Em *IEEE Symposium on Computers and Communications*, páginas 1–7, Athens, Greece.

Saeed, A., Dukkipati, N., Valancius, V., The Lam, V., Contavalli, C. e Vahdat, A. (2017). Carousel: Scalable traffic shaping at end hosts. Em *Conference of the ACM Special Interest Group on Data Communication*, páginas 404–417.

Salopek, D., Vasic, V., Zec, M. et al. (2014). A network testbed for commercial telecommunications product testing. Em *International Conference on Software, Telecommunications and Computer Networks*, páginas 372–377, Split, Croácia.

Sanchoyerto, A., Solozabal, R., Blanco, B. et al. (2019). Orchestration of mission-critical services over an nfv architecture. Em *International Conference on Artificial Intelligence Applications and Innovations*, páginas 70–77, Hersonissos, Grécia.

Sanner, M. F. et al. (1999). Python: A programming language for software integration and development. *J Mol Graph Model*, 17(1):57–61.

Santos, G. L., Bezerra, D. d. F., Rocha, E. d. S., Ferreira, L., Moreira, A. L. C., Gonçalves, G. E., Marquezini, M. V., Recse, Á., Mehta, A., Kelner, J. et al. (2022). Service function chain placement in distributed scenarios: a systematic review. *Journal of Network and Systems Management*, 30(1):1–39.

Sarvotham, S., Riedi, R. e Baraniuk, R. (2001). Connection-level analysis and modeling of network traffic. Em *ACM SIGCOMM Workshop on Internet Measurement*, página 99–103.

Sauvanaud, C., Lazri, K., Kaâniche, M. et al. (2016). Towards black-box anomaly detection in virtual network functions. Em *International Conference on Dependable Systems and Networks Workshop*, páginas 254–257, Toulouse, França.

Savi, M., Tornatore, M. e Verticale, G. (2019). Impact of processing-resource sharing on the placement of chained virtual network functions. *Transactions on Cloud Computing*.

SecDev (2021). Scapy: Packet crafting for python2 and python3. `https://scapy.net`. Acessado em 08/03/2021.

Sector, I. T. S. (2000). Principles for a telecommunications management network. Relatório técnico, ITU.

Sekar, V., Egi, N., Ratnasamy, S. et al. (2012). Design and implementation of a consolidated middlebox architecture. Em *Symposium on Networked Systems Design and Implementation*, páginas 323–336, San Jose, EUA.

Services, G. T. S. G. e Aspects, S. (2018). Management and orchestration; architecture framework. Relatório técnico, 3GPP.

Sherry, J., Hasan, S., Scott, C. et al. (2012). Making middleboxes someone else's problem: Network processing as a cloud service. *ACM SIGCOMM Computer Communication Review*, 42(4):13–24.

Sonkoly, B., Czentye, J.and Szabo, R. et al. (2015). Multi-domain service orchestration over networks and clouds: A unified approach. Em *ACM Conference on Special Interest Group on Data Communication*, páginas 377–378, Londres, Reino Unido.

Soualah, O., Mechtri, M., Ghribi, C. et al. (2018). A green vnfs placement and chaining algorithm. Em *Network Operations and Management Symposium*, páginas 1–5, Taipei, Taiwan.

Stroustrup, B. (1999). An overview of the c++ programming language. *Handbook of object technology*, 1(1):15–1 – 15–24.

Stucki, S., Sánchez, C., Schneider, G. et al. (2021). Gray-box monitoring of hyperproperties with an application to privacy. *Formal Methods in System Design*, páginas 1–34.

Tavakoli-Someh, S. et al. (2019). Utilization-aware virtual network function placement using nsga-ii evolutionary computing. Em *Conf. on Knowledge Based Engineering and Innovation*, páginas 510–514.

Tavares, T., Marcuzzo, L., Fulber-Garcia, V. et al. (2018). Niep: Nfv infrastructure emulation platform. Em *Conference on Advanced Information Networking and Applications*, páginas 173–180, Cracóvia, Polônia.

The Alpine Project (2021). Alpine linux: Small, simple and secure. `https://alpinelinux.org`. Acessado em 09/03/2021.

The GNU Netcat Project (2021). Netcat: A networking utility. `http://netcat.sourceforge.net`. Acessado em 10/03/2021.

The Iperf Project (2021). Iperf: The tcp/udp bandwidth measurement tool. `https://iperf.fr`. Acessado em 12/03/2021.

The libnet Developer Community (2021). libnet: Packet crafting. `https://github.com/libnet/libnet`. Acessado em 08/03/2021.

The Linux Foundation (2021). Vpp: Vector packet processing. `https://fd.io/docs/vpp/master`. Acessado em 08/03/2021.

The MongoDB Project (2021). Mongodb: The database for modern applications. `https://mongodb.com`. Acessado em 02/04/2021.

The PostgreSQL Project (2021). Postgresql: The world's most advanced open source relational database. `https://postgresql.org`. Acessado em 02/04/2021.

The SQLite Project (2021). Sqlite: Small, fast, self-contained, high-reliability, full-featured, sql database engine. https://sqlite.org. Acessado em 02/04/2021.

The Tiny Core Project (2021). Tiny core linux: The core project. http://tinycorelinux.net. Acessado em 09/03/2021.

Venâncio, G., Garcia, V. F., d. C. Marcuzzo, L. et al. (2021). Beyond vnfm: Filling the gaps of the etsi vnf manager to fully support vnf life cycle operations. *International Journal of Network Management*, página e2068.

Venâncio, G., Turchetti, R. C., Camargo, E. T. et al. (2020). Vnf-consensus: A virtual network function for maintaining a consistent distributed software-defined network control plane. *International Journal of Network Management*, página e2124.

Vu, A., Dinh, N. e Kim, Y. (2016). Modeling of service function chaining in network function virtualization environment. Em *Korean Institute of Communication Sciences Conference*, páginas 1100–1101, Seul, Coréia.

Wang, L., Mao, W., Zhao, J. e Xu, Y. (2021). Ddqp: A double deep q-learning approach to online fault-tolerant sfc placement. *IEEE Transactions on Network and Service Management*, 18(1):118–132.

Wang, Y., Lu, P., Lu, W. e Zhu, Z. (2017). Cost-efficient virtual network function graph (vnfg) provisioning in multidomain elastic optical networks. *IEEE Journal of Lightwave Technology*, 35(13):2712–2723.

Wette, P., Draxler, M., Schwabe, A. et al. (2014). Maxinet: Distributed emulation of software-defined networks. Em *Networking*, páginas 1–9, Trondheim, Noruega.

Wikileaks (2018). Amazon atlas 2015. https://wikileaks.org/amazon-atlas/. Accessed in 01 Apr. 2021.

Xilouris, G., Kourtis, M., McGrat, M. et al. (2015). T-nova: Network functions as-a-service over virtualised infrastructures. Em *Conference on Network Function Virtualization and Software Defined Network*, páginas 13–14, San Francisco, EUA.

Xilouris, G., Trouva, E., Lobillo, F. et al. (2014). T-nova: A marketplace for virtualized network functions. Em *European Conference on Networks and Communications*, páginas 1–5, Bolonha, Itália.

Yasukata, K., Huici, F., Maffione, V. et al. (2017). Hypernf: Building a high performance, high utilization and fair nfv platform. Em *Symposium on Cloud Computing*, páginas 157–169, Santa Clara, EUA.

Yi, B., Wang, X., Li, K., Huang, M. et al. (2018). A comprehensive survey of network function virtualization. *Computer Networks*, 133:212–262.

Yousaf, F. Z., Sciancalepore, V., Liebsch, M. et al. (2019). Manoaas: A multi-tenant nfv mano for 5g network slices. *IEEE Communications Magazine*, 57(5):103–109.

Zhang, Q., Wang, X., Kim, I., Palacharla, P. e Ikeuchi, T. (2016a). Vertex-centric computation of service function chains in multi-domain networks. Em *IEEE Conf. on Network Softwarization*, páginas 211–218.

Zhang, W., Liu, G., Zhang, W. et al. (2016b). Opennetvm: A platform for high performance network service chains. Em *Workshop on Hot topics in Middleboxes and Network Function Virtualization*, páginas 26–31, Florianópolis, Brasil.

Zhang, X., Huang, Z. et al. (2017). Online stochastic buy-sell mechanism for vnf chains in the nfv market. *Journal on Selected Areas in Communications*, 35(2):392–406.

Zheng, C., Lu, Q., Li, J. et al. (2018). A flexible and efficient container-based nfv platform for middlebox networking. Em *Annual ACM Symposium on Applied Computing*, páginas 989–995, Pau, França.

Zitzler, E., Laumanns, M. e Thiele, L. (2001). Spea2: Improving the strength pareto evolutionary algorithm. *TIK Report*, 103.

APÊNDICE A – ARQUITETURA SFC TOLERANTE A FALHAS E INTRUSÃO

Como apresentado no Capítulo 2, uma *Service Function Chain* (SFC) é um serviço virtual de rede definido como uma composição, em cadeia, de instâncias de VNF. Essas cadeias de funções tipicamente objetivam executar funcionalidades de rede elaboradas através da transmissão e processamento de tráfego em topologias de serviço (Fulber-Garcia et al., 2020a). A *Internet Engineering Task Force* define uma arquitetura de referência para SFC (Quinn e Nadeau, 2015), baseada em três componentes principais: *Service Classifier* (SC), *Service Function Forwarder* (SFF) e as próprias instâncias de VNF. Apesar de serviços de rede virtualizados serem amplamente explorados na literatura recente, vários aspectos de tolerância a falhas ainda são marginalmente abordados neste contexto. Dentre as estratégias para a construção de serviços virtuais tolerantes a falhas, é possível citar aquelas que se dedicam a identificar, tolerar e, eventualmente, recuperar falhas por parada em instâncias de VNF (Qu et al., 2016; Kong et al., 2017; Ghaznavi et al., 2020; Wang et al., 2021).

A solução REACH (*REliability-Aware service CHaining*) (Qu et al., 2016) propõe a utilização de uma estratégia baseada em ILP para tolerar falhas por parada em VNFs. O REACH emprega múltiplos nodos funcionando como *backups* para atender os requisitos de tolerância a falhas do serviço. Além disso, uma heurística gulosa é proposta para simplificar a ILP, tornando o tempo de execução da solução viável. Já em (Kong et al., 2017), outra solução para garantir a alta disponibilidade de SFCs é proposta. Nesse caso, os autores consideram a utilização de *backups* tanto para os enlaces quanto para as instâncias relacionadas às VNFs de um serviço. Assim, através de um algoritmo heurístico, a solução define a quantidade de réplicas a serem criadas para cada VNF presente em uma topologia completa.

A estratégia apresentada em (Ghaznavi et al., 2020), chamada de FTC (*Fault Tolerant Chaining*), exibe um sistema de tolerância a falhas por parada em SFCs. Utilizando o próprio fluxo de processamento dos pacotes, o sistema FTC extrai e copia informações sobre o estado das VNFs. Porém, o FTC não utiliza réplicas das instâncias de VNF para salvar tais estados. Na verdade, cada instância de VNF na topologia de serviço age como uma réplica para a sua VNF sucessora. Tal característica torna o FTC um sistema não compatível com a arquitetura de referência SFC IETF, uma vez que demanda comunicação direta entre instâncias de VNF diferentes. Finalmente, a solução proposta em (Wang et al., 2021) assume a utilização de instâncias em modo *standby* para prover tolerância a falhas em serviços de rede virtualizados. Essa proposta emprega dois métodos de *Deep Reinforcement Learning* (DRL), cujo objetivo é decidir, de maneira eficiente, o mapeamento e a implantação de instâncias de VNF (tanto as ativas, quanto as *standby*) no substrato físico. Além disso, para SFCs que possuem VNFs *stateful*, um mecanismo extra é empregado para enviar constantemente o estado de uma instância de VNF ativa para as suas réplicas em *standby*. Vale ressaltar que essa estratégia de replicação de estado de VNFs *stateful* é bastante custosa em termos de mensagens trocadas no sistema. Por fim, a solução em (Wang et al., 2021) também não é nativamente compatível com a arquitetura SFC IETF.

Apesar da existência das soluções de tolerância a falhas previamente apresentadas, todas as estratégias consideradas tratam apenas a possível ocorrência de falhas por parada (*crash*) em instâncias de VNF. Sendo assim, como parte do conjunto de contribuições desta Tese, propõe-se uma arquitetura para a replicação e tolerância a falhas em SFCs que considera a possibilidade dos três componentes da arquitetura SFC da IETF (SC, SFF e VNF) sofrerem falhas por parada, omissão ou intrusão (falhas bizantinas). É importante ressaltar que a arquitetura proposta é

completamente compatível com a arquitetura de referência SFC IETF. Além disso, a arquitetura foi implementada na forma de um protótipo, visando a demonstração de sua capacidade em manter serviços corretos mesmo após a ocorrência de falhas por parada ou intrusões decorrentes de ataques *man-in-the-middle*. Em seguida, a Seção A.1 apresenta a arquitetura proposta, denominada *Fault- & Intrusion-Tolerant SFC* (FIT-SFC); então, na Seção A.1.1 a implementação de tal arquitetura é abordada, cenários de teste são detalhados, e resultados preliminares obtidos são discutidos; finalmente, a Seção A.1.2 apresenta considerações finais sobre a arquitetura FIT-SFC.

A.1 FIT-SFC: ARQUITETURA

A presente seção apresenta a arquitetura denominada FIT-SFC que prevê tolerância a falhas para todos os elementos operacionais da arquitetura SFC da IETF e utiliza técnicas de replicação para garantir: (*i*) que nenhum pacote seja perdido ou duplicado na presença de falhas; e (*ii*) que a arquitetura tolere f falhas *crash* ou bizantinas (Lamport et al., 2019). Uma falha bizantina é uma falha arbitrária, sendo normalmente utilizada para representar um componente que sofreu uma intrusão decorrente de um ataque. Neste trabalho um ataque consiste na modificação não autorizada de pacotes que percorrem a SFC.

O modelo de sistema adotado é assíncrono, isto é, não há garantias temporais fortes e cada componente do sistema pode atrasar arbitrariamente para processar e enviar pacotes na SFC. Todos os elementos operacionais de uma SFC—SC, SFF e VNF—podem sofrer falhas bizantinas. Por decorrência, os elementos podem também sofrer falhas por parada, ou omitir pacotes específicos. Entretanto, para cada elemento operacional, o número máximo de unidades falhas é f. A estratégia proposta é baseada em replicação.

A Figura A.1 ilustra a arquitetura geral da FIT-SFC. O tráfego é recebido pelas réplicas de SC, responsáveis pela classificação e encaminhamento deste para as réplicas de SFF. Para que uma aplicação cliente encaminhe tráfego de rede para um serviço provido através da arquitetura proposta, a ela deve ser acrescentado um *wrapper*. Tal *wrapper* tem duas funcionalidades: (i) enviar cada pacote para as réplicas do SC; (ii) marcar cada pacote enviado com um *timestamp* local, que pode ser implementado como um contador local de pacotes transmitidos para a SFC. O objetivo é permitir a identificação única de cada pacote, descartando cópias de pacotes já processados. Assim, cada réplica de SC encaminha uma única vez cada pacote recebido para todas as réplicas de SFF. Caso uma entidade seja maliciosa, a FIT-SFC garante que todos os componentes da SFC decidam pelo mesmo pacote em cada estágio, considerando a existência de um quórum de pacotes coincidentes. Se esse critério não for alcançado, entretanto, o encaminhamento e processamento do tráfego é interrompido (em qualquer estágio).

O SFF recebe cópias de cada pacote de todas as réplicas de SC. Uma vez que podem haver até f instâncias de SC falhas, é necessário que o SFF faça uma votação para selecionar o pacote correto a ser encaminhado para as funções de rede – note que versões diferentes de pacotes podem ser recebidas devido a falhas bizantinas. Esse processo de votação é realizado também pelas instâncias de VNF, que da mesma forma recebem pacotes de todas as réplicas de SFF.

Para viabilizar esta funcionalidade, um elemento adicional é incorporado às réplicas de SFF e VNF, denominado *Voter*. A partir do momento em que o *Voter* recebe $2f + 1$ cópias idênticas de um pacote, este é autorizado a processar e encaminhar o mesmo para o destino seguinte. O número de cópias adotado pode ser entendido da seguinte forma: de maneira inicial, para existir uma decisão unânime entre todos os componentes de um determinado sistema é necessário o recebimento de uma quantidade mínima de mensagens provenientes de componentes corretos, sendo que esta deve superar o número de mensagens provenientes de (ou omitidas por)

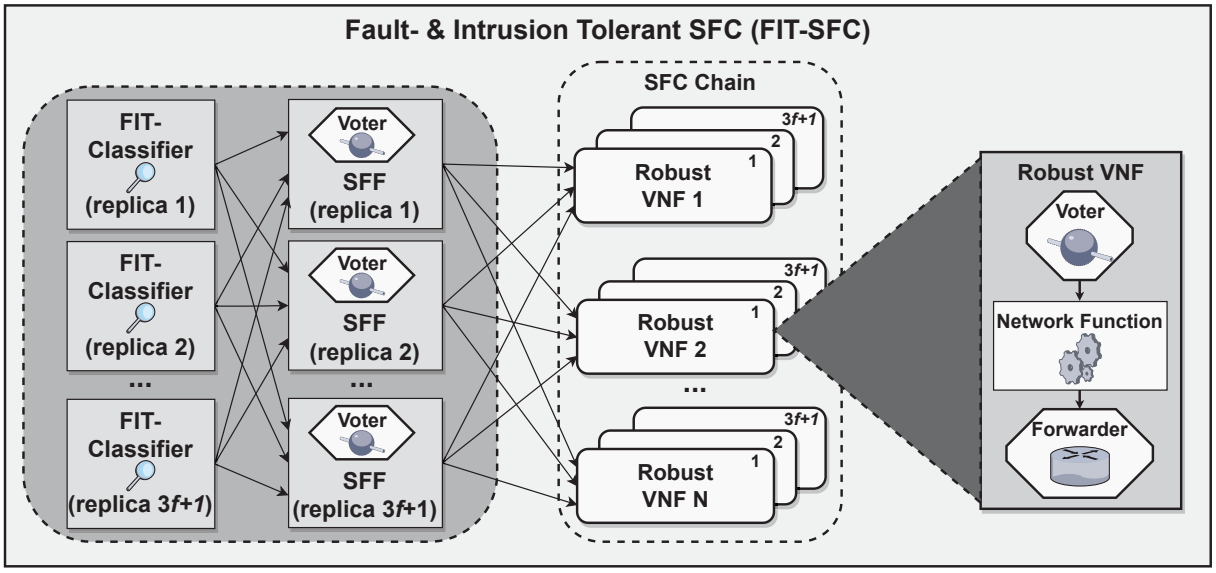


Figura A.1: Arquitetura FIT-SFC Simplificada

componentes falhos. Considerando que n seja a quantidade total de componentes em um sistema, e f seja a quantidade de componentes falhos neste mesmo sistema, a Inequação A.1 representa a condição mínima para prover tolerância a falhas.

$$n - f > f \tag{A.1}$$

Mas, em um cenário de falhas bizantinas, dos $n–f$ componentes que recebem mensagens suficientes para formar maioria (quórum) em um determinado ponto do sistema, f podem operar de forma maliciosa, não omitindo mensagens, mas alterando o seu conteúdo. Sendo assim, f componentes enviam mensagens diferentes daquela enviada pelo único componente não malicioso que recebeu um número suficiente de mensagens para a obtenção do quórum. Com isso, o próximo conjunto de componentes a receber estas mensagens jamais obterão um quórum, e não darão prosseguimento ao processamento das mensagens. O cenário descrito é ilustrado na Figura A.2, considerando $n = 3$ e $f = 1$.

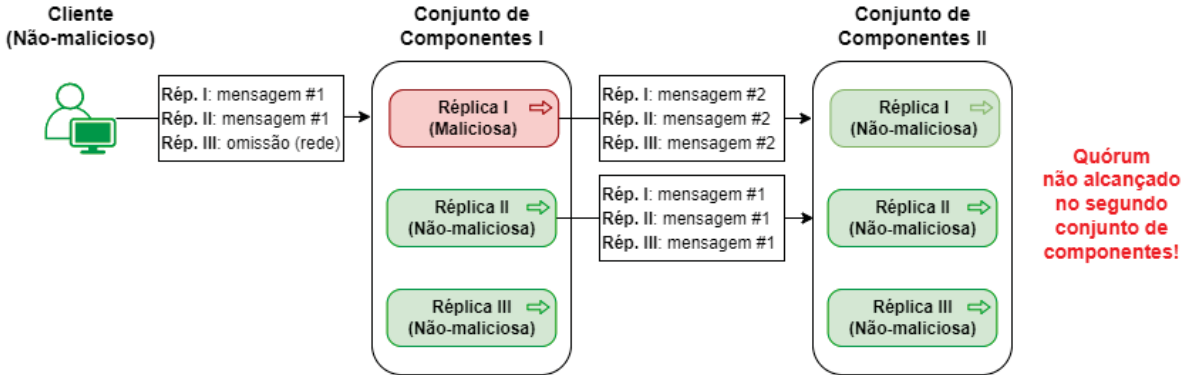


Figura A.2: Cenário de Impossibilidade de Quórum (Falhas Bizantinas)

A solução para o cenário apresentado consiste em considerar a possibilidade de f mensagens recebidas serem maliciosas e, portanto, descartáveis. Essa solução, porém, altera o número necessário de réplicas de cada componente, já que é fundamental garantir a existência de um quórum de mensagens verdadeiras apesar do envio de f mensagens maliciosas de um

ponto a outro do sistema. A Inequação A.2 manifesta a condição de tolerância a falhas bizantinas em sistemas distribuídos, resultando na necessidade de, no mínimo, $3f + 1$ réplicas de cada componente operando nos mesmos.

$$n - f - f > f \tag{A.2}$$

Porém, a formação de um quórum simples não é suficiente para tratar casos em que o próprio cliente assume o papel de uma entidade maliciosa. Nesse caso, há um conluio entre o cliente e o primeiro componente com o qual este se comunica no sistema. Em um sistema simples tolerando uma falha (quatro réplicas por componente), sendo uma réplica do primeiro componente do sistema maliciosa (falha bizantina), o conluio pode acontecer como descrito a seguir: o cliente envia uma mensagem qualquer para o componente malicioso, uma mensagem do tipo #1 para um componente não malicioso, e uma mensagem do tipo #2 para um segundo componente não malicioso, omitindo uma mensagem que seria enviada para o último componente não malicioso. Em seguida, o componente malicioso encaminha uma mensagem do tipo #1 para metade dos componentes presentes no próximo conjunto do sistema, e do tipo #2 para a outra metade. Ao receberem as mensagens dos componentes não maliciosos, o segundo conjunto de componentes do sistema decidirá pelo processamento e encaminhamento de mensagens diferentes, sempre correspondendo à mensagem recebido do componente malicioso anterior. A Figura A.3 exibe o cenário descrito.

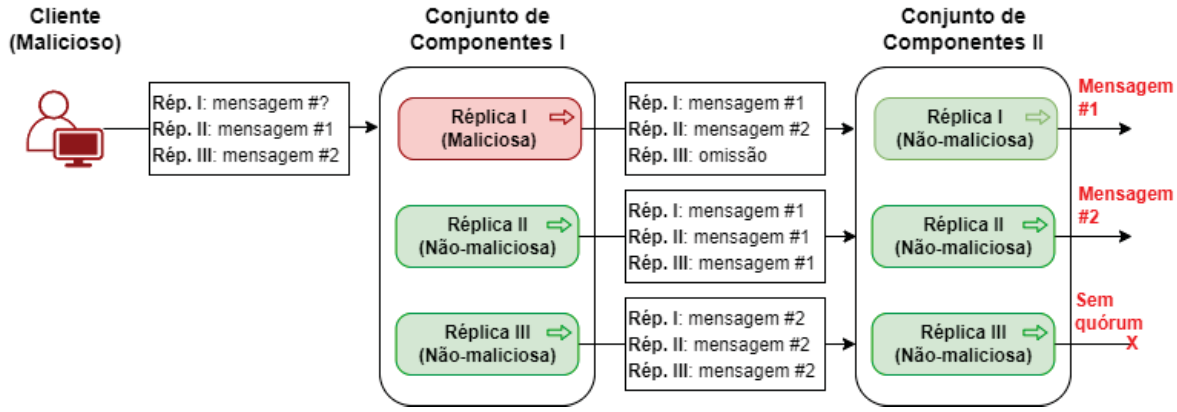


Figura A.3: Cenário de Conluio com Cliente Malicioso

A solução para conluios relacionados à presença de clientes maliciosos consiste em determinar o tamanho do quórum como $2f + 1$. Nesse caso, no cenário de conluio proposto, a formação de um quórum é impossibilitada a partir do segundo conjunto de componentes, cancelando o processamento do fluxo do cliente malicioso neste ponto. Sendo assim, a arquitetura proposta adota um número de réplicas por elemento operacional da SFC igual à $3f + 1$, sendo o quórum formado pelo recebimento de $2f + 1$ mensagens coincidentes.

A.1.1 Experimentação e Resultados Preliminares

O protótipo da FIT-SFC desenvolvido implementa os elementos operacionais da arquitetura SFC utilizando a linguagem Python 3.9.1 [1]. Os elementos realizam todas as trocas de mensagens previstas na Seção A.1, permitindo a detecção e mitigação de até f falhas bizantinas. Além da configuração de rotas e fluxos tradicionais da arquitetura SFC da IETF, os SCs da arquitetura FIT-SFC são configurados com informações das suas réplicas em execução, permitindo

[1] O código do protótipo está disponível em `https://github.com/ViniGarcia/FIT-SFC`

a troca de mensagens entre elas. Os SFFs recebem mensagens redundantes de todos os SCs e VNFs, portanto os primeiros também são configurados para reconhecer todas as réplicas de SC disponíveis, assim como todas as réplicas das VNFs dos serviços cadastrados. Por fim, as VNFs devem ter conhecimento de todas as réplicas de SFFs aos quais elas se relacionam, permitindo o envio e recebimento de tráfego entre tais elementos operacionais.

Os cenários de teste da arquitetura FIT-SFC foram construídos utilizando o emulador NIEP (Tavares et al., 2018). Dois cenários foram considerados: o primeiro cenário (C1) executa os elementos operacionais da arquitetura FIT-SFC, porém utiliza apenas uma instância de cada elemento, não tolerando falhas; já o segundo cenário (C2) executa quatro réplicas de cada elemento da arquitetura FIT-SFC, tolerando o total de uma falha. O serviço de rede empregado em todos os cenários consiste em uma cadeia linear com duas VNFs, cada uma executando um encaminhador de tráfego. Todos os experimentos relatados foram executados 50 vezes em uma máquina com processador Intel Core I5-3330 @ 3.0GHz, 8GB RAM DDR3 e Ubuntu 16.04.

O primeiro experimento avalia a progressão do tempo total de processamento de um pacote do momento da emissão do mesmo pelo cliente até o seu recebimento pelo servidor ao qual foi destinado, passando antes por todos os elementos operacionais da arquitetura FIT-SFC previamente descritos. A Figura A.4 mostra os resultados obtidos para os cenários estabelecidos, sendo as barras a média do tempo descrito e as barras de erro o desvio padrão.

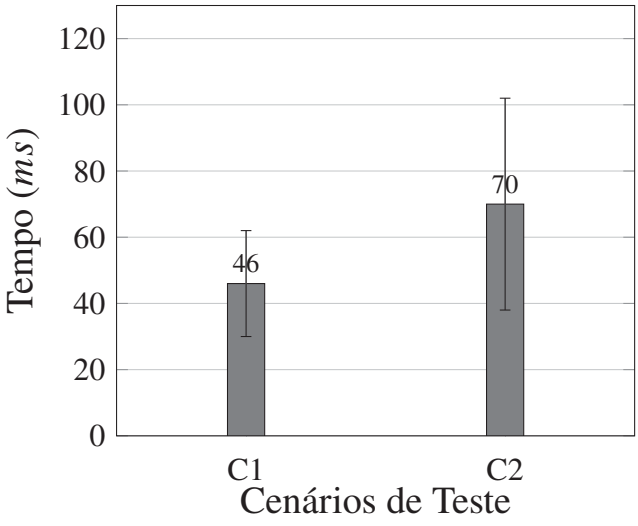


Figura A.4: Atraso Decorrente da Arquitetura FIT-SFC

Naturalmente, existe um aumento no tempo total de processamento dos pacotes conforme a quantidade de réplicas aumenta. Isso ocorre devido a dois fatores principais: (*i*) quanto maior a quantidade de falhas a serem toleradas, maior é a quantidade de pacotes redundantes que precisam ser recebidos e analisados; (*ii*) quanto maior o número de réplicas, maior é o número de pacotes transitando na rede. Observa-se um aumento sublinear do tempo entre o C1 e C2. Esse fenômeno ocorre visto que existe um conjunto de tarefas realizadas com um tempo similar independentemente do número de réplicas utilizadas, como a alocação do NSH nos pacotes pelo SC e a sua posterior manipulação pelas instâncias de SFF e VNF.

O segundo experimento utiliza a arquitetura FIT-SFC com quatro réplicas de cada elemento operacional necessário para a execução do serviço. Nesse caso, o serviço de rede empregado é o mesmo que foi descrito no primeiro experimento. O segundo experimento considera dois casos de testes: elementos operacionais param de responder devido a falhas por parada (caso *Crash*); e ataques *man-in-the-middle* modificando o conteúdo de pacotes

encaminhados entre os elementos (caso *Intrusão*). Consideramos, ainda, cinco cenários onde progressivamente as réplicas dos elementos falham ou são atacadas: todos os elementos corretos (C1); uma instância da primeira VNF do serviço falha/é atacada (C2); além de C2, uma instância de SC falha/é atacada (C3); além de C3, uma instância de SFF falha/é atacada (C4) e; além de C4, uma instância da segunda VNF do serviço falha/é atacada (C5).

Para os cenários do segundo experimento, os resultados são apresentados na Figura A.5. Nessa figura, as barras representam o tempo médio de processamento dos pacotes pela arquitetura e as barras de erro representam o desvio padrão.

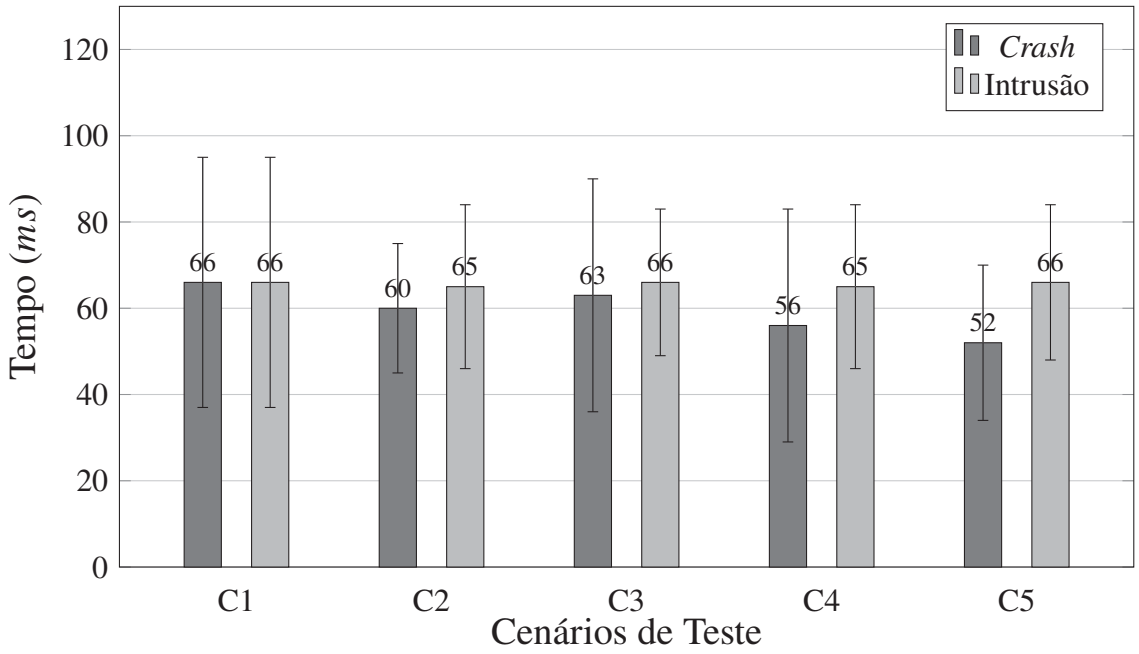


Figura A.5: Atraso em Cenários com Falhas por Parada e Intrusão

No caso de falhas por parada do segundo experimento, é possível notar que existe uma tendência de queda na média do tempo de processamento de pacotes pela FIT-SFC. Isso acontece porque quando uma réplica para de responder, a sobrecarga na rede diminui e menos pacotes são recebidos por cada uma das demais réplicas. Por outro lado, no caso dos ataques, esse fenômeno da redução no tempo de execução não acontece, pois o fluxo de pacotes dessas réplicas não é alterado. Por fim, é importante ressaltar que, no segundo experimento da arquitetura FIT-SFC, todas as falhas foram toleradas.

A.1.2 Considerações Finais

Este apêndice apresentou uma arquitetura baseada em replicação para prover tolerância a falhas em serviços de rede desenvolvidos de acordo com arquitetura SFC da IETF. A proposta, denominada FIT-SFC, considera duas contribuições principais para o estado da arte: a primeira consiste na extensão da tolerância falhas para além das VNFs, considerando também elementos de classificação e encaminhamento de tráfego; a segunda consiste na adequação da FIT-SFC para tolerar intrusões (falhas bizantinas). A avaliação da arquitetura proposta foi feita através da implementação de um protótipo e sua aplicação em diferentes cenários de teste relacionados a um estudo de caso, demonstrando sua resiliência ao tolerar de falhas por parada e intrusões decorrentes de ataques *man-in-the-middle*.

APÊNDICE B – ENGENHARIA DE TRÁFEGO BASEADA EM NFV

A engenharia de tráfego representa uma estratégia concreta para o provisionamento de Qualidade de Serviço (QoS - *Quality of Service*) em redes de computadores (Nucci e Papagiannaki, 2009; Cavalca et al., 2014). Entre seus principais papéis está o de garantir o bom desempenho da rede, enquanto mantém a utilização dos recursos computacionais da mesma em níveis desejados e previamente estabelecidos. Assim, o emprego de mecanismos de engenharia de tráfego viabiliza a criação e a manutenção de redes de computadores de forma mais robusta e eficiente. Dentre os principais mecanismos de engenharia de tráfego existentes estão os *policers* e os *shapers* (Evans e Filsfils, 2010; Daniel-Simion e Dan-Horia, 2011). *Policers* têm o propósito principal de eliminar picos em fluxos de dados, limitando os mesmos a uma taxa máxima predefinida. Já os *shapers* suavizam o perfil do tráfego de rede, atrasando os pacotes ao empregar filas consumidas de acordo com taxas preestabelecidas.

Tradicionalmente, a engenharia de tráfego é implementada no núcleo da rede através de equipamentos dedicados, ou na borda da rede como processos sendo executados diretamente em servidores de aplicação. A primeira estratégia (núcleo da rede) herda todas as limitações relacionadas à utilização de equipamentos físicos e dedicados para a execução de funções, como baixa flexibilidade e mobilidade, além de altos custos operacionais e de capital (Guerzoni et al., 2012). Já a segunda estratégia (borda da rede) pode resultar em sobrecargas dos servidores de aplicação, que passam a reservar parte de sua capacidade computacional para processar tráfego de rede em detrimento de utilizá-la para seu objetivo fim (*i.e.*, prover um determinado serviço) (Saeed et al., 2017).

Considerando o cenário apresentado, o paradigma NFV, e seus benefícios no contexto de escalabilidade, flexibilidade e gerenciamento, pode ser explorado como uma forma de implementação de mecanismos de engenharia de tráfego no núcleo das redes de computadores. Com isso, mecanismos de engenharia de tráfego passam a ser compreendidos como software instanciados em plataformas de VNF (*Virtualized Network Functions*), estes executados em hardware de propósito geral através de tecnologias de virtualização (Fulber-Garcia et al., 2019a, 2020a; Flauzino et al., 2020; Venâncio et al., 2021).

Atualmente, as estratégias para o uso de NFV no contexto de engenharia de tráfego incluem a lavagem de pacotes (*packet wash*) de Dong e Clemm (Dong e Clemm, 2021). Essa estratégia objetiva assegurar limites máximos de latência ponto a ponto com alta precisão, mesmo em redes congestionadas. O processo executado reduz o tamanho dos pacotes, descartando seletivamente fragmentos da sua carga útil sem causar interrupções na entrega. O algoritmo apresentado pelos autores minimiza o tempo de permanência (*dwell time*) de pacotes em roteadores e, por consequência, o atraso fim a fim de um serviço. Também, a lavagem de pacotes faz a negociação da transmissão de blocos de baixa prioridade, aumentando a possibilidade de o tráfego prioritário alcançar o seu destino considerando políticas de atraso predefinidas.

Além da lavagem de pacotes, é possível destacar a solução para mitigação de ataques distribuídos de negação de serviço (DDoS — *Distributed Denial of Service*) proposta em (Fulber-Garcia et al., 2018). Tal solução, chamada DeMONS, utiliza os conceitos do paradigma NFV, além de estratégias de alocação dinâmica de funções e um mecanismo de reputação, para processamento e análise do tráfego de rede. O DeMONS é uma solução híbrida composta por cinco módulos principais, sendo que um deles consiste em uma função de *policing*. Fluxos de pacotes processados pelo DeMONS são marcados com uma reputação entre 0 e 1. Fluxos com reputação 0 são imediatamente bloqueados em um *firewall*. Já fluxos com reputação maior que 0

são alocados em diferentes canais, que podem ser de baixa ou alta prioridade. O *policer*, em particular, limita a disponibilidade de banda em canais de baixa prioridade conforme o valor de reputação dos fluxos alocados nos mesmos. Os resultados apresentados no trabalho demonstram a viabilidade do DeMONS em mitigar eficientemente ataques DDoS através do uso de engenharia de tráfego.

Apesar da existência dos trabalhos citados anteriormente, a engenharia de tráfego ainda é marginalmente explorada dentro do paradigma NFV. As soluções existentes são orientadas a aplicações específicas, tendo seu desenvolvimento e funcionamento dedicados ao sistema ou solução ao qual fazem parte. Nesse contexto, como uma das contribuições da presente Tese, o NFV-TE (NFV *Traffic Engineering*)[1], um *framework* parametrizável para geração de funções de rede de engenharia de tráfego, é proposto. No NFV-TE, em resumo, o operador informa parâmetros do mecanismo de engenharia de tráfego desejado, a partir dois quais são gerados códigos-fonte de funções de rede. Essas funções são genéricas e flexíveis, podendo compor diferentes serviços virtualizados de rede. Dentre os mecanismos de *policing* e *shaping* disponibilizados no NFV-TE estão: *Single Rate Token Bucket Policer* (srTBP), *Single Rate Three Color Marker Policer* (srTCM-*Policer*), *Two Rate Three Color Marker Policer* (trTCM-*Policer*) e *Color-Aware Policers* para *policing*; além de *Single Rate Token Bucket Shaper* (srTBS) e *Leaky Bucket* (LB) para *shaping*.

A seguir, a Seção B.1 apresenta as principais características dos mecanismos de *policing* e *shaping* implementados no NFV-TE, assim como detalhes técnicos destas implementações; testes de validação das funções de rede geradas pelo NFV-TE são descritos, exibidos e discutidos na Seção B.2; por fim, considerações finais sobre o *framework* são delineadas na Seção B.3.

B.1 O *FRAMEWORK* NFV-TE

O *framework* proposto foi desenvolvido utilizando a linguagem de programação *Python 3*. Também, o Python 3 foi empregado como a linguagem padrão para a geração das funções de rede. As funções virtuais de engenharia de tráfego apresentam um modelo de especificação facilitado, visando reduzir os esforços dos operadores no exercício de tal atividade. O *framework* conta com uma interface gráfica simples, a qual permite o fornecimento de dados utilizados para gerar um arquivo *JSON* com todos os parâmetros do mecanismo de engenharia de tráfego desejado. Este arquivo inclui campos essenciais, como a categoria da função de engenharia de tráfego a ser desenvolvida (atualmente, *policing* ou *shaping*), o identificador da função de rede, além de um conjunto de parâmetros específicos relacionados a cada categoria de função. Nesse arquivo também devem ser definidos os nomes das *interfaces* de rede das/para as quais os pacotes são recebidos e enviados. Após a inserção dos dados, sua subsequente validação e criação do arquivo JSON de parâmetros, a geração de código-fonte de uma função de rede é então realizada, tendo como saída um arquivo de extensão “.py". Pequenas adaptações no código-fonte da função gerada podem ser necessárias para viabilizar a sua execução em qualquer plataforma de VNF. Esta seção se dedica a detalhar os parâmetros necessários para a geração de *policers* e *shapers* no NFV-TE, assim como a descrever o modo de funcionamento de cada um dos mecanismos disponibilizados.

B.1.1 *Policers*

Policers são implementados utilizando dois elementos principais: o *token*, representando um Byte de um pacote, e o *bucket*, o qual é uma estrutura de armazenamento de *tokens*.

[1]Disponível em https://github.com/jvmoreira/multiservice-networks

Particularmente, os *buckets* apresentam uma capacidade máxima de armazenamento, chamada *burst*. Também, os *tokens* em um *bucket* são consumidos a cada pacote processado pela função de rede, havendo uma reposição de acordo com políticas definidas levando em consideração o mecanismo de *policer* adotado.

Para geração de funções de *Policer* através do NFV-TE é preciso, inicialmente, definir o campo de categoria no arquivo JSON de configurações como "*policing*". Atualmente, nesta categoria podem ser geradas funções de rede para três mecanismos de *policers*: *Single Rate Token Bucket Policer* (srTBP), *Single Rate Three Color Marker Policer* (srTCM-*Policer*) e *Two Rate Three Color Marker Policer* (trTCM-*Policer*), adotando a variação ou não de *Color-Aware* para os dois últimos.

O *policer* srTBP atua verificando o tamanho de cada pacote, transmitindo os mesmos apenas caso haja *tokens* suficientes no *bucket*. Os parâmetros de configuração necessários para a geração de um srTBP são: "*bucket_size*"que corresponde a quantidade inicial de *tokens* no *bucket*; "*bucket_max_size*"que corresponde ao tamanho máximo do *bucket*; "*interval*"indicando o valor em segundos do intervalo entre cada reposição de *tokens*; e "*rate*"que indica à quantidade de *tokens* que são adicionados ao *bucket* a cada intervalo. O fluxograma de execução de um mecanismo de *policing* srTBP é ilustrado na Figura B.1.

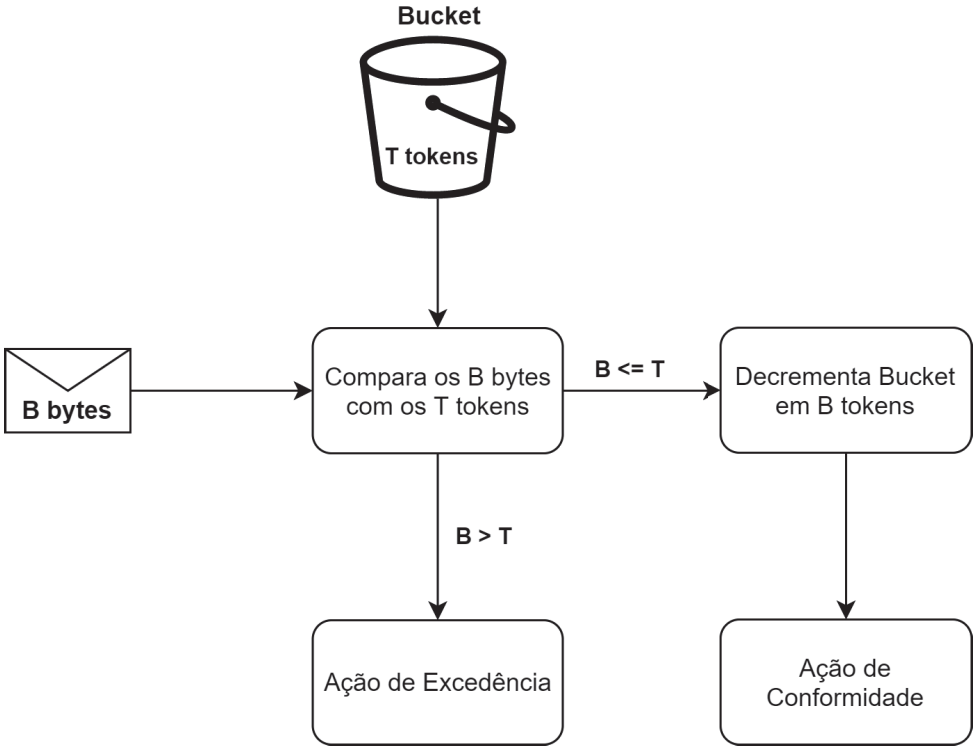


Figura B.1: Fluxograma de Execução do srTBP *Policer*

Já o funcionamento do srTCM-*Policer* se dá pela execução das ações referentes à marcação de pacotes em cores verde, amarela e vermelha. Para isso, considera-se a quantidade de *tokens* presentes em dois *buckets*: o de Conformidade (C) e o de Exceção (E). Tais *buckets* têm *tokens* adicionados a uma taxa constante e única. Assim, caso um pacote recebido tenha tamanho B menor ou igual ao número de *tokens* no *bucket* C, a ação verde é tomada e decrementa-se B *tokens* do *bucket* C. Caso contrário, se B for menor ou igual ao número de *tokens* no *bucket* E, a ação amarela é executada e decrementa-se B *tokens* do *bucket* E. Caso nenhuma das condições anteriores seja satisfeita, a ação vermelha é então executada. Os processos relacionadas a cada

ação são definidos pelo operador de rede. O fluxograma de execução de um mecanismo de *policing* srTCM é ilustrado na Figura B.2.

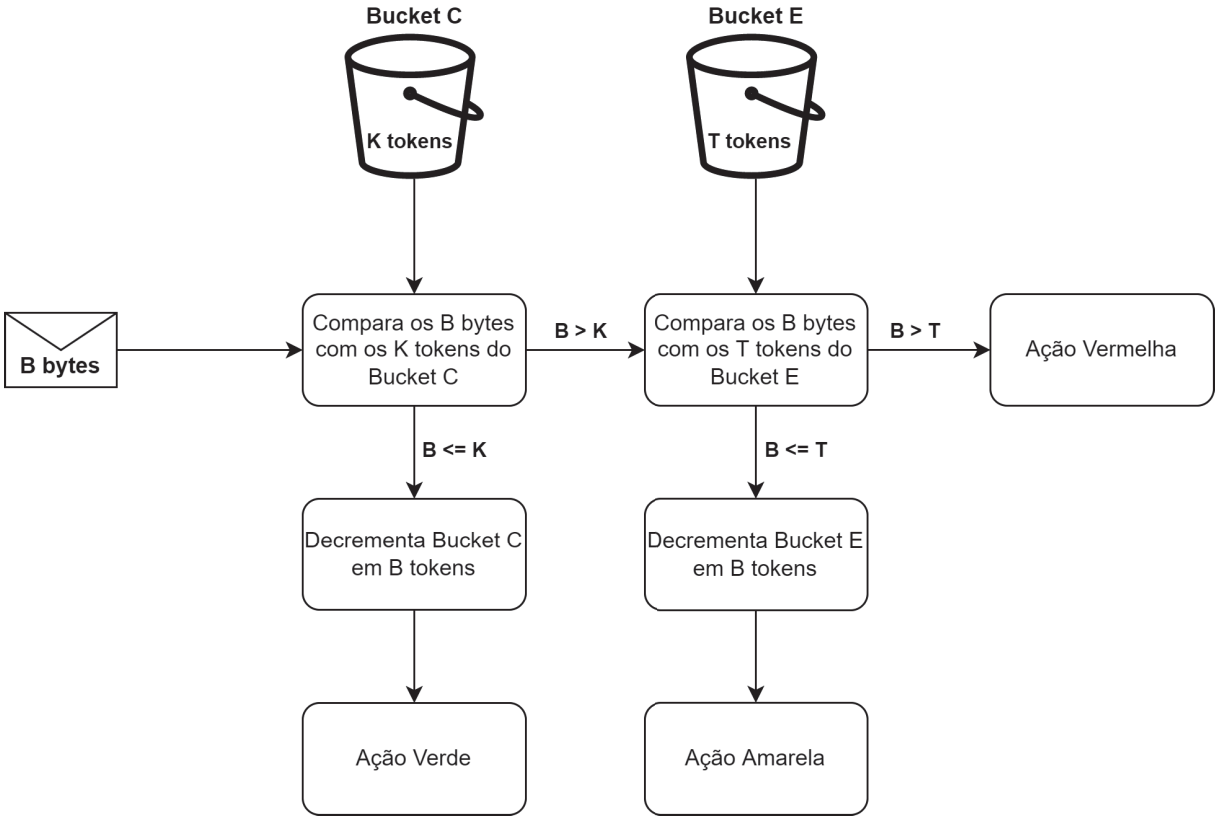


Figura B.2: Fluxograma de Execução do srTCM *Policer*

Os parâmetros de configuração necessários para o *Single Rate Three Color Marker Policer* são: "*interval*"que corresponde ao valor em segundos do intervalo entre cada reposição de *tokens*; "*rate*"que corresponde à quantidade de *tokens* que são adicionados aos dois *buckets* a cada intervalo; "*bucketF_size*" e "*bucketS_size*"que correspondem às quantidades iniciais de *tokens* nos *buckets* C e E, respectivamente; "*bucketF_max_size*" e "*bucketS_max_size*" que correspondem ao número máximo de *tokens* para os *buckets* C e E, respectivamente.

No que diz respeito ao trTCM-*Policer*, este atua de forma similar ao srTCM-*Policer* em relação a execução de ações correspondentes à marcação de pacotes nas cores verde, amarela e vermelha, conforme a quantidade de *tokens* presentes em dois *buckets*: o de Pico (P) e o de Conformidade (C). Tais *buckets* têm seus *tokens* incrementados a uma taxa constante e individualizada. Caso o pacote recebido tenha tamanho B maior do que o número de *tokens* disponíveis no *bucket* P, a ação vermelha é executada. Caso contrário, decrementa-se B *tokens* do *bucket* P e compara-se o tamanho do pacote com os *tokens* disponíveis no *bucket* C. Caso B seja maior que o número de *tokens* no *bucket* C, a ação amarela é executada; caso contrário decrementa-se B *tokens* do *bucket* C e a ação verde é realizada. Novamente, os processos de cada cor são definidos pelos operadores de rede. O fluxograma de execução de um mecanismo de *policing* trTCM é ilustrado na Figura B.3.

Os parâmetros de configuração necessários para o *Two Rate Three Color Marker Policer* são: "*interval*"que corresponde ao valor em segundos do intervalo entre cada reposição; "*rateS*" e "*rateF*"que correspondem à quantidade de *tokens* que são adicionados aos *buckets* P e C, respectivamente, a cada intervalo; "*bucketS_size*"e "*bucketF_size*" que correspondem

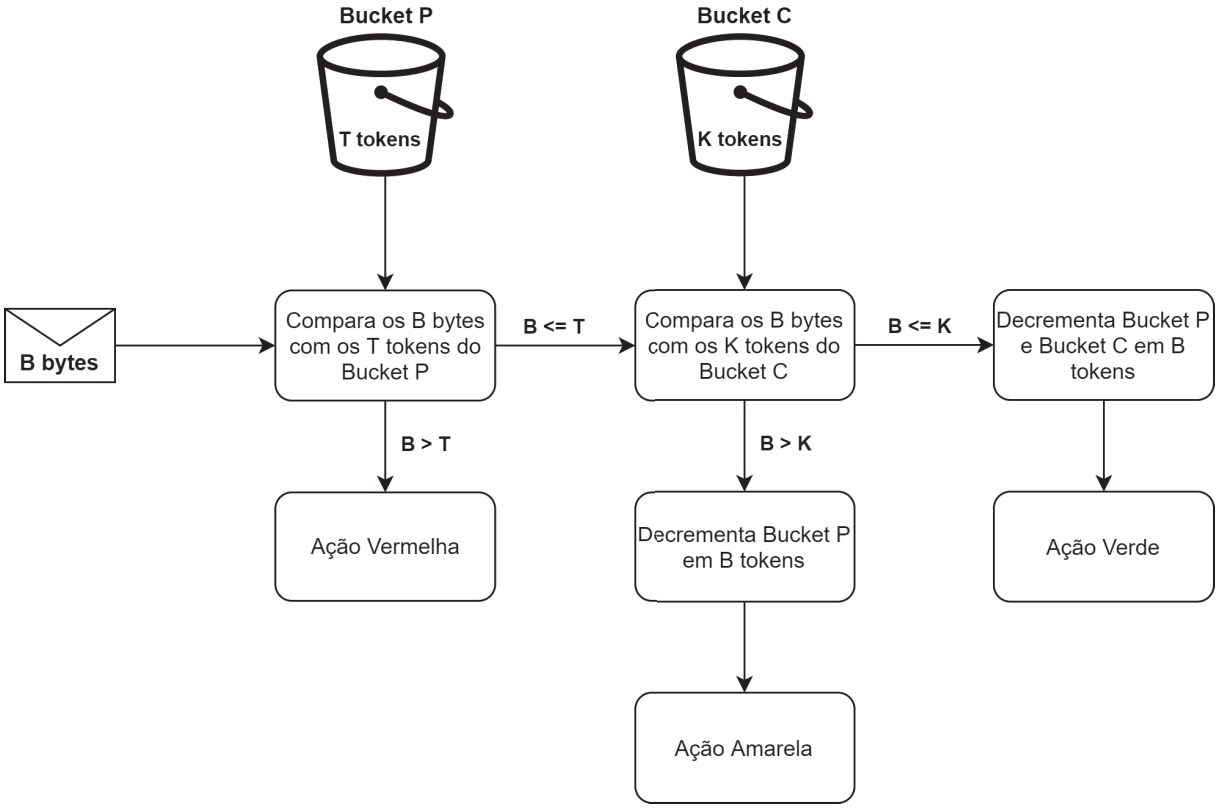


Figura B.3: Fluxograma de Execução do trTCM *Policer*

às quantidades iniciais de *tokens* nos *buckets* P e C, respectivamente; "*bucketS_max_size*" e "*bucketF_max_size*"que indicam o número máximo de *tokens* para os *buckets* P e C, em ordem.

Por fim, um *Color-Aware Policer*, consiste em um mecanismo de *policing* (dentre os apresentados até aqui) utilizado após uma função de marcação (*marker*). Esses mecanismos verificam a coloração atribuída a cada um dos pacotes recebidos e decidem pela execução de uma ação apropriada conforme a sua configuração preestabelecida. Ou seja, por mais que existam *tokens* suficientes para a tomada de ação de uma determinada cor conforme o mecanismo de *policing* utilizado, esta pode não ser executada devido a uma decisão prévia da função de marcação. Da mesma forma, uma ação mais restritiva em relação àquela indicada pela função de marcação pode ser tomada se não houverem *tokens* suficientes nos *buckets* do mecanismo de *policing* adotado. Foram implementadas as versões *Color-Aware* do srTCM e do trTCM. Os fluxogramas de execução de mecanismos de *policing* srTCM e trTCM em suas versões *color-aware* são ilustrados, respectivamente, nas Figuras B.4 e B.5.

Para a utilização da versão *Color-Aware*, o parâmetro de configuração *"color_aware"* deve ser marcado como verdadeiro. Com isso, os seguintes parâmetros também devem ser definidos: *"ca_bucketF_size"* e *"ca_bucketS_size"* que correspondem às quantidades iniciais de *tokens* nos *buckets* C e E no caso do srTCM-PCA e *buckets* C e P no caso do trTCM-PCA; *"ca_bucketF_max_size"* e *"ca_bucketS_max_size"* que correspondem ao número máximo de *tokens* para os *buckets* C e E no caso do srTCM-PCA e *buckets* C e P no caso do trTCM-PCA; o parâmetro *"ca_rate"* deve ser usado no srTCM-PCA para indicar a quantidade de *tokens* que são adicionados aos *buckets* C e E a cada intervalo; e os parâmetros *"ca_rateF"* e *"ca_rateS"* que descrevem, para o trTCM-PCA, a quantidade de *tokens* que são adicionados aos *buckets* C e P, respectivamente, a cada intervalo.

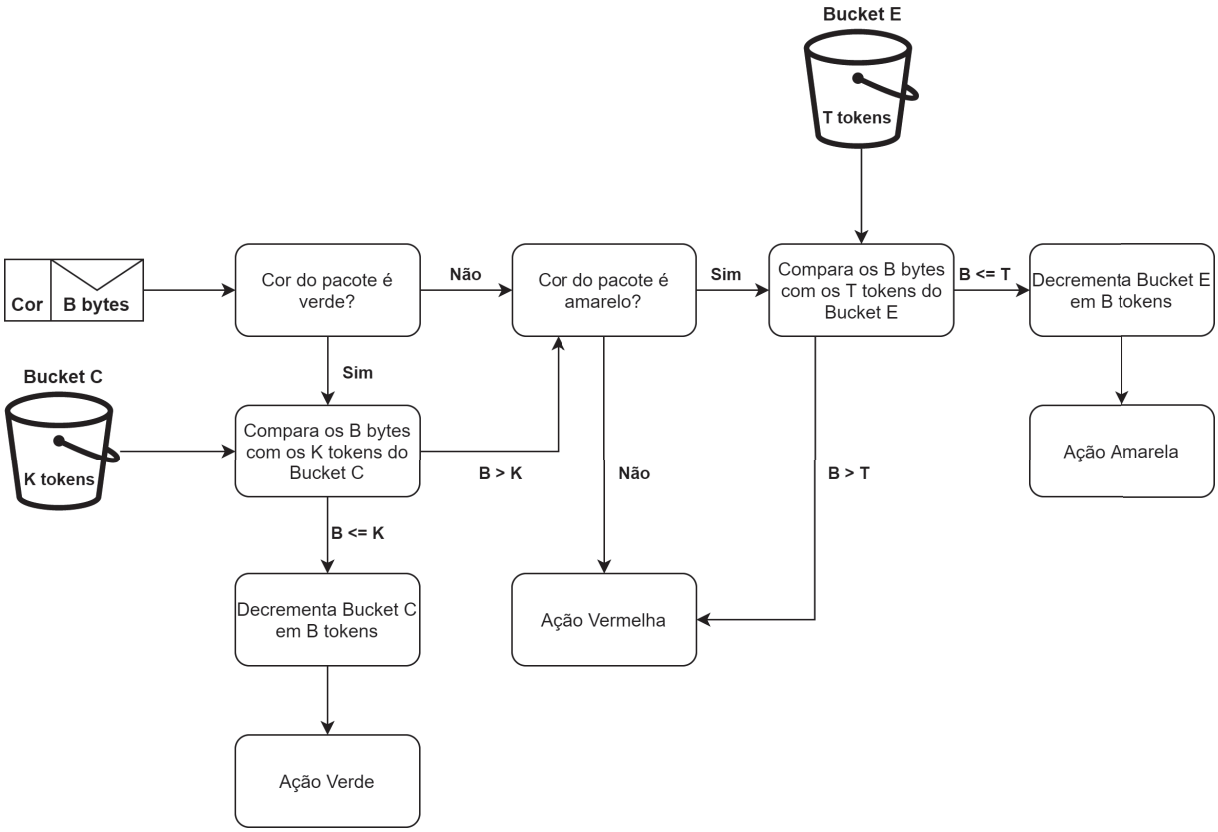


Figura B.4: Fluxograma de Execução do srTCM *Policer Color-Aware*

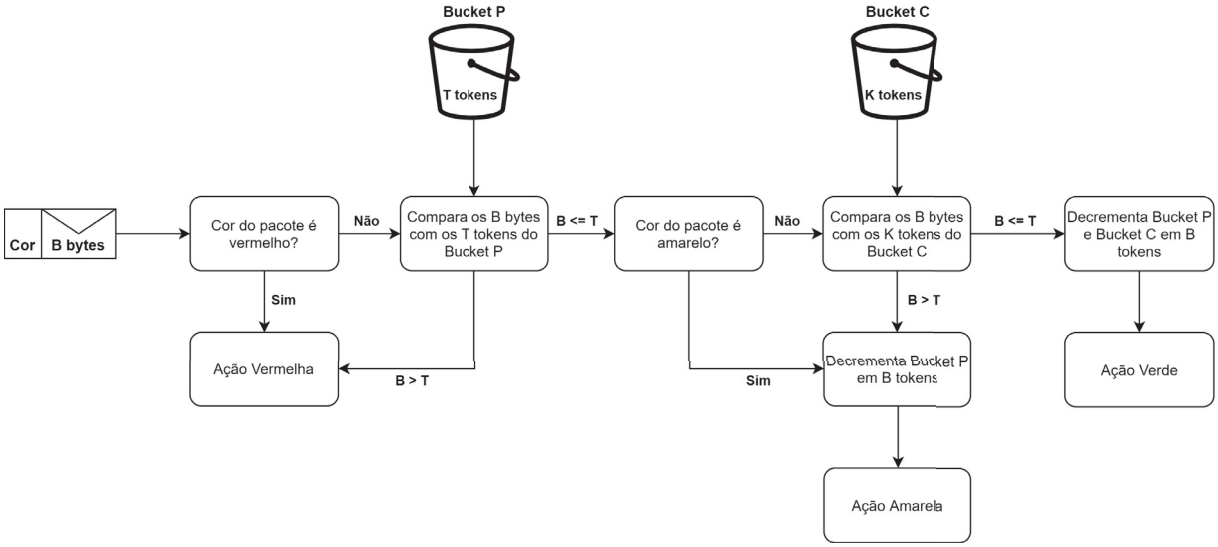


Figura B.5: Fluxograma de Execução do trTCM *Policer Color-Aware*

B.1.2 *Shapers*

Ao contrário dos *policers* que diretamente descartam tráfego de rede, os *shapers* utilizam filas para postergar a transmissão de pacotes para momentos futuros em que as condições apropriadas forem atingidas. Para a geração de *shapers* no NFV-TE, o campo de categoria do arquivo JSON deve ser definido como "*shaping*". Nesta categoria podem ser geradas funções para dois mecanismos de *shaping*: *Single Rate Token Bucket Shaper* (srTBS) e *Leaky Bucket* (LB).

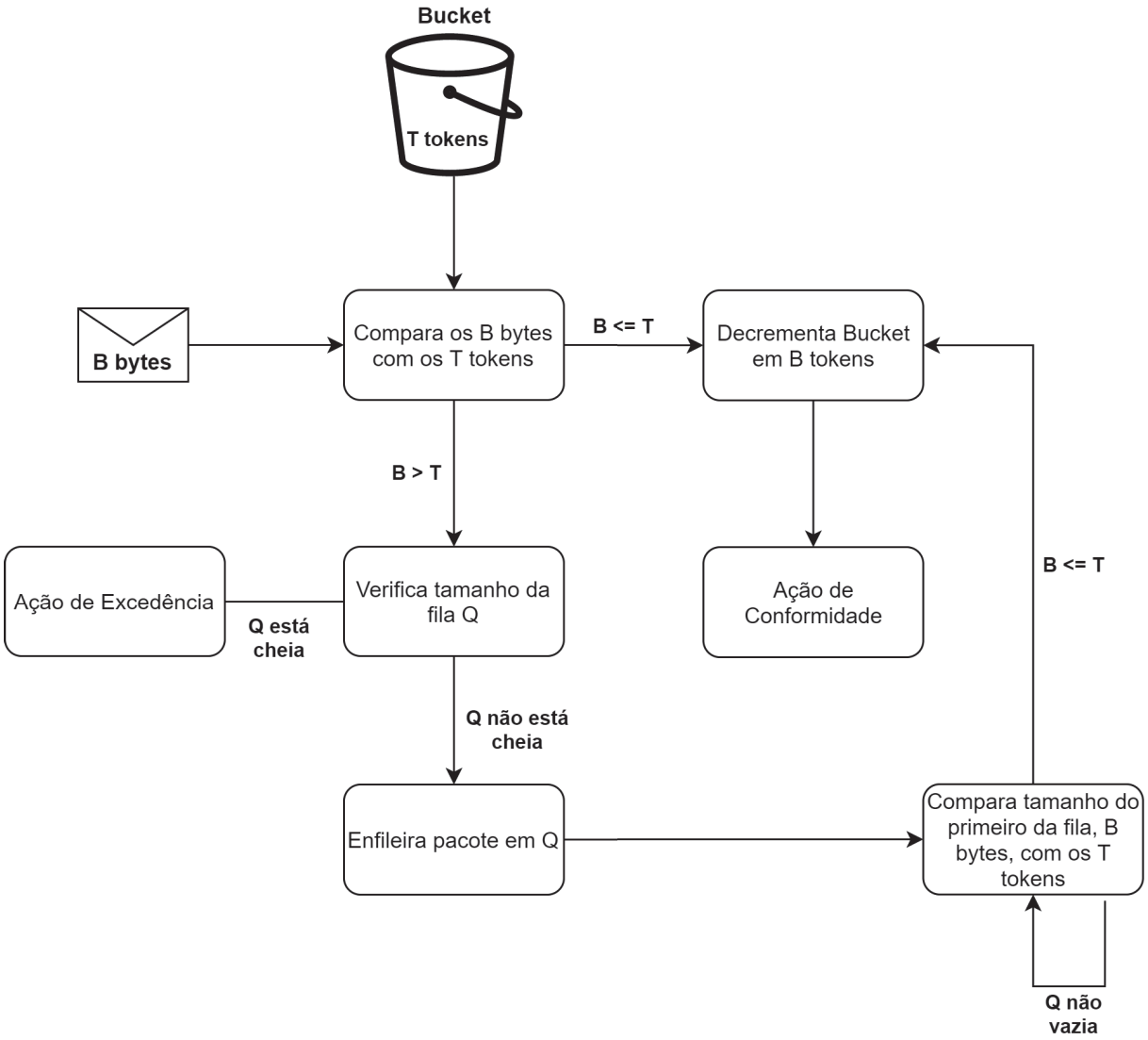


Figura B.6: Fluxograma de Execução do srTBS *Shaper*

O srTBS é definido da mesma forma que o *policer* srTBP. Assim, durante o processamento do tráfego de rede, são realizadas comparações entre o tamanho do pacote e a quantidade de *tokens* no *bucket*. Se a quantidade de bytes de um pacote não exceder a quantidade de *tokens* no *bucket*, este é imediatamente transmitido e os *tokens* correspondentes consumidos. Por outro lado, quando um pacote excede o *bucket*, ao invés deste ser imediatamente descartado, o mesmo é adicionado a uma fila, a qual é consumida à medida que *tokens* suficientes são adicionados ao *bucket*. Se a fila estiver cheia, porém, os pacotes que excederem o *bucket* são diretamente descartados. No fluxograma da Figura B.6 é demonstrado o comportamento descrito do srTBS.

Os parâmetros de configuração necessários para o *Single Rate Token Bucket Shaper* são: "*bucket_size*" que corresponde ao número inicial de *tokens* no *bucket*; "*bucket_max_size*" que corresponde ao número máximo de *tokens* no *bucket*; "*interval*" que corresponde ao valor em segundos do intervalo entre cada reposição; "*rate*" que corresponde à quantidade de *tokens* que são adicionados a cada intervalo; e "*queue_max_size*" que indica o tamanho máximo da fila de pacotes.

O *Leaky Bucket* (LB) é outro mecanismo de *shaping*. Entretanto, diferente do srTBS, onde o *bucket* armazena *tokens*, no mecanismo de LB, o *bucket* é preenchido com os próprios pacotes recebidos, armazenados em fila. Dessa forma, a profundidade do *bucket* define a quantidade

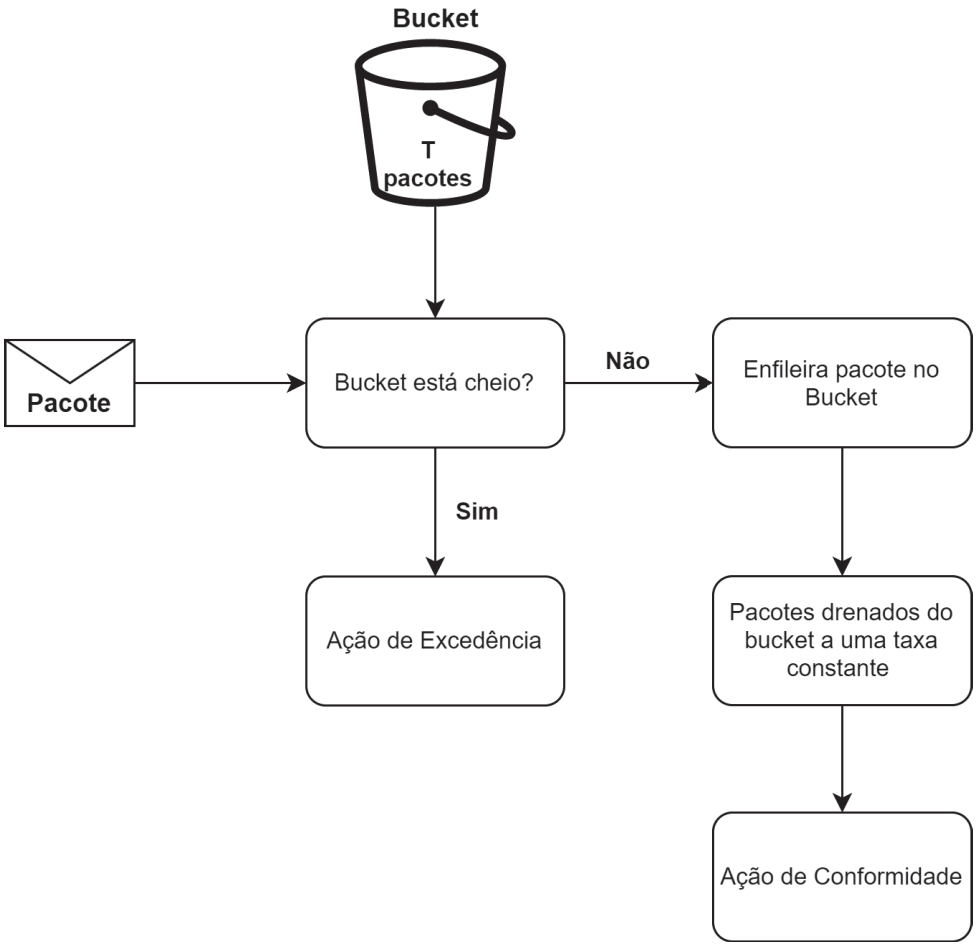


Figura B.7: Fluxograma de Execução do LB *Shaper*

máxima de pacotes que podem ser enfileirados nele. Porém, ao se atingir a capacidade máxima, os novos pacotes recebidos não podem ser armazenados, sendo imediatamente descartados. Também, é necessário que o mecanismo de LB consuma (*i.e.*, encaminhe ao próximo destino) os pacotes presentes no *bucket*. Para isso, é definida uma taxa constante de consumo, chamada *rate*. A taxa é aplicada utilizando um intervalo fixo e preestabelecido de tempo. Então, a cada intervalo, uma quantidade de pacotes equivalente ao valor de *rate* é drenada do *bucket*, em um processo executado como ilustrado na Figura B.7.

Os parâmetros de configuração necessários para o *Leaky Bucket* são: "*bucket_max_size*" que indica o tamanho máximo da fila (*bucket*); "*interval*" que corresponde ao valor em segundos do intervalo entre cada transmissão; e "*packets_to_release*" que corresponde à quantidade de pacotes transmitidos a cada intervalo.

B.2 EXPERIMENTAÇÃO E RESULTADOS

O funcionamento das funções de rede geradas pelo NFV-TE foram avaliadas considerando um ambiente de testes composto por três máquinas virtuais, todas executando o sistema Linux Ubuntu 18.4 em uma máquina hospedeira com o mesmo sistema operacional, CPU Intel i7-8750H e 4GB de RAM DDR3. A primeira VM assume a função de cliente, ou seja, envia um fluxo de pacotes para um servidor. A segunda VM assume a função de servidor, que recebe e processa os pacotes enviados pelos clientes. Por fim, situada entre o cliente e servidor, uma VM executa a

VNF gerada pelo *framework* NFV-TE, realizando o *policing* ou *shaping* do tráfego de pacotes entre cliente e servidor.

Para verificar o funcionamento dos mecanismos de *policing* e *shaping*, foram utilizados quatro perfis de tráfego de rede, sendo eles: Alfa (Sarvotham et al., 2001), Beta (Sarvotham et al., 2001), Elefante (Hamdan et al., 2020) e Guepardo (Maji et al., 2017). O perfil Alfa tem como característica o envio de rajadas de 10 pacotes de 348 bytes em 1 segundo, seguido por um período de silêncio de 2 segundos. O perfil Beta envia 10 pacotes por segundo, sendo que a cada segundo o tamanho em bytes dos pacotes enviados é alternado entre 148 e 348 bytes. O perfil Elefante tem um fluxo contínuo de pacotes com grande volume de dados, mas em menor quantidade em relação aos demais perfis, sendo enviados 5 pacotes por segundo com 1048 bytes cada. O perfil Guepardo apresenta o maior número de pacotes enviados por segundo, mas com a menor quantidade de bytes em cada um deles (68 bytes). Por fim, o perfil 10 Mbps apresenta as características de tráfego para um teste voltado a *thrpoughput*, adotando o envio de 1000 pacotes por segundo com 1250 bytes cada. A Tabela B.1 sumariza as características dos perfis de tráfego de rede utilizados.

	Tamanho do Pacote (Bytes)	**Pacotes por Segundo**	**Intervalo entre Envios (s)**	**Bytes por Segundo**
Alfa	348	10 ou 0*	0,1 (+2 a cada 10 pacotes)	3480 ou 0*
Beta	148 ou 348 (fluxos alternados)	10	0,1	1480 ou 3480 (fluxos alternados)
Elefante	1048	5	0,2	5240
Guepardo	68	20	0,05	1360
10 Mbps	1250	1000	0,001	1250000

*Serão transmitidos 0 pacotes (0 bytes) se o tráfego estiver no hiato de 2s sem transmissão

Tabela B.1: Perfis de Tráfego de Rede

Nos experimentos, foram utilizados intervalos de tempo padrão de 1 segundo tanto para a adição de *tokens* aos *buckets* ou para o consumo de pacotes em um *bucket*. Em todos os casos, quando necessário, os *buckets* são inicializados com as suas capacidades máximas de *tokens*. Além disso, durante um caso de teste, o número de pacotes transmitidos pelos tráfegos Alfa, Beta, Elefante e Guepardo será sempre igual a 500. Já para os experimentos com tráfego de 10Mbps são transmitidos 10 mil pacotes. Como experimentos preliminares, diferentes versões do *policer* srTBP e do *shaper* LB foram empregadas para a validação inicial do *framework* proposto.

O primeiro caso de teste busca validar o comportamento operacional de funções de *policing* srTBP e *shaping* LB quando submetidas a diferentes cenários de tráfego. Para isso, considera-se um tamanho máximo de *bucket* do srTBP de 4000 *tokens* e alterna-se a sua configuração de *rate* entre 1500 e 3900 *tokens* por segundo. Já para o *shaper* LB, a profundidade do *bucket* foi definida em 50 pacotes, e o *rate* é alternado entre 3 e 25 pacotes por segundo. Após instanciadas no ambiente de testes, as funções de rede processaram o tráfego conforme os perfis previamente apresentados. Para cada cenário considerado, as quantidades de pacotes transmitidos com sucesso entre cliente e servidor são expostas, respectivamente, nas Figuras B.8 e B.9.

Na Figura B.8 é possível verificar que, para o tráfego Guepardo, o mecanismo de srTBP não executou descartes de pacotes, encaminhando a totalidade dos mesmos do cliente ao servidor, independente do valor do *rate*. Por outro lado, o srTBP com *rate* igual à 1500 *tokens*, executou a transmissão de 86,6%, 70,4% e 29,8% da totalidade dos pacotes, para os tráfegos, respectivamente, Alfa, Beta e Elefante. Entretanto, ao aumentar o *rate* para 3900, o srTBP realizou a transmissão de 100% dos pacotes para os tráfegos Alfa e Beta, e atingiu 60,6% de

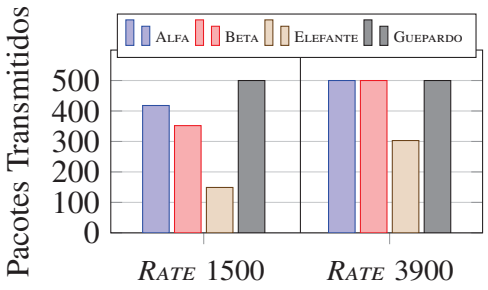


Figura B.8: srTBP (*Bucket* 4000)

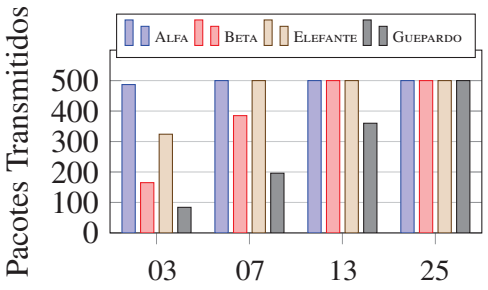


Figura B.9: LB (Taxa Constante)

pacotes transmitidos para o tráfego Elefante. Portanto, aumentar a quantidade de *tokens* que são adicionados ao *bucket* a cada intervalo de tempo gera um aumento no número de transmissões realizadas pelo srTBP para os diferentes perfis de tráfegos. Essa é uma consequência natural, uma vez que o *rate* determina a quantidade máxima de bytes processados a cada intervalo de tempo. Ressalta-se que o fato do tráfego Elefante nunca ser completamente repassado ao servidor é decorrência da quantidade total de bytes transmitidos por segundo pelo meno, que supera as taxas de *rate* em ambas configurações testadas.

A Figura B.9 apresenta os resultados de validação do *shaper* LB. Nesse caso, para o tráfego Guepardo, o LB transmite 16,8%, 39,2%, 72% e 100% quando o *rate* é igual a 3, 7, 13 e 25, respectivamente. Fato que ocorre devido à alta quantidade de pacotes enviados por segundo por esse perfil tráfego. Para o tráfego Alfa e Elefante, quando o *rate* é igual a 3, atinge-se 97,4% e 64,8% de transmissões dos pacotes recebidos, respectivamente. Porém, ao aumentar o *rate*, ambos os tráfegos atingem os 100% de transmissão. Já para o tráfego Beta, não se atinge 100% de transmissões quando o *rate* é igual a 3 e 7, alcançando, nestas configurações, 33% e 77% de pacotes encaminhados, respectivamente. Tal fenômeno é decorrência da transmissão de 10 pacotes por segundo pelo tráfego Beta, valor que supera a capacidade de consumo de *rates* 3 e 7.

Além da validação operacional das funções de rede através dos testes com diferentes perfis de tráfego, um segundo caso de teste, onde o cliente envia 10 Mbps de tráfego, é realizado como um experimento focado em *throughput*. Para o srTBP, os testes de *throughput* utilizam tamanhos máximos de *bucket* iguais a 700 mil, 900 mil e 1,1 milhão de *tokens*, sendo o *rate* adotado igual a 1,1 milhão de *tokens* por segundo. Os resultados obtidos são apresentados na Figura B.10.

Ao analisar o experimento com o srTBP é possível verificar que, para um *bucket* de tamanho igual a 700 mil *tokens*, o fluxo é restringido, visto que tal tamanho é menor que o *rate* padrão adotado. Assim, atinge-se, em média, a transmissão de 6,7 Mbps. Para um *bucket* de tamanho igual a 900 mil, obtém-se uma taxa de transmissão de 8,6 Mbps, em média. Por fim, para um *bucket* de tamanho igual a 1,1 milhão, mesmo valor do *rate*, é obtido uma taxa de transmissão de 10 Mbps, ou seja, todos os pacotes recebidos são encaminhados.

Já os experimentos de *throughput* para o LB utilizam *rates* de 500, 700, 900 e 1100 pacotes por segundo e um tamanho de *bucket* padrão de igual a 10% do *rate* adotado (50, 70, 90 e 110 pacotes, respectivamente). Os resultados obtidos são apresentados na Figura B.11. Após análise, é possível verificar que para o *rate* igual a 500, atinge-se transmissão de, em média, 5,6 Mbps, para 700 pacotes, atinge-se, em média, 8,4 Mbps, e para 900 e 1100 pacotes, atinge-se 10 Mbps. Essa progressão está diretamente ligada a relação entre *rate* e volume de pacotes recebidos pela função de rede. Além disso, o tamanho do *bucket* e o enfileiramento realizado no mesmo é a característica que permite o encaminhamento da totalidade dos pacotes mesmo com um *rate* inferior ao volume de chegada de tráfego, como acontece na adoção de um *rate* igual a 900.

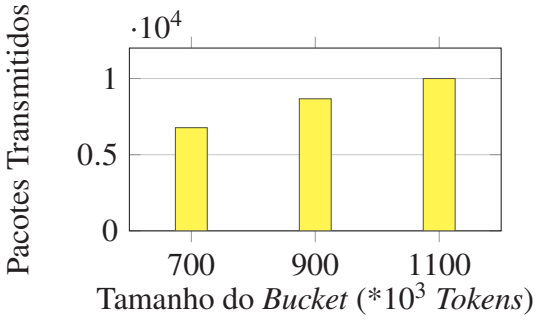


Figura B.10: srTBP (10Mbps)

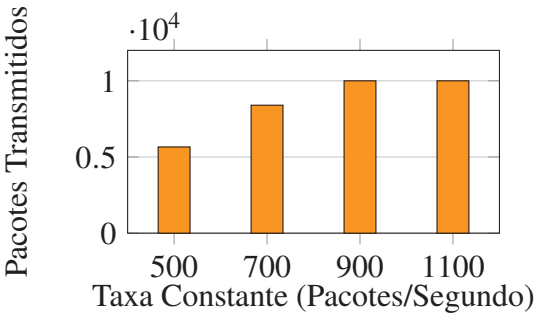


Figura B.11: LB (10Mbps)

Os testes até então discutidos apresentam indícios do funcionamento de alguns mecanismos de engenharia de tráfego gerados pelo NFV-TE, estes processando diferentes perfis e tipos de fluxos de rede. Os resultados obtidos, em conjunto com a análise dos mesmos, validam preliminarmente o comportamento operacional dos *shapers* e *policers* testados, provando serem adequados para o processamento de pacotes e controle de carga em uma rede de computadores.

B.3 CONSIDERAÇÕES FINAIS

As estratégias comumente adotadas para a implementação de soluções de engenharia de tráfego consideram a utilização de equipamentos dedicados operando no núcleo das redes de computadores, ou a execução de processos diretamente nos servidores de aplicação. Porém, essas abordagens apresentam importantes limitações quanto a flexibilidade e mobilidade de tais soluções no decorrer de seu ciclo de vida. Sendo assim, propõe-se o NFV-TE, um *framework* parametrizável para geração de funções de rede voltadas para engenharia de tráfego, desenvolvido no contexto do paradigma NFV. O objetivo do NFV-TE é incluir estratégias de engenharia de tráfego em um cenário de virtualização de núcleo de rede, herdando as características de flexibilidade, baixo custo e simplificação de gerenciamento inerentes ao paradigma NFV. Os principais mecanismos de engenharia de tráfego disponibilizados pelo NFV-TE foram avaliados em um cenário experimental, considerando uma variedade de casos de teste. Através dos resultados obtidos, foi possível observar a conformidade operacional das funções geradas com a especificação teórica das mesmas.

Como trabalhos futuros, objetiva-se disponibilizar o NFV-TE através de *marketplaces* (Xilouris et al., 2014; Bondan et al., 2019) para ser usado em ambientes de produção. Também, é objetivo incluir outros mecanismos de engenharia de tráfego no contexto de geração de funções do *framework*, como os *schedulers*. Por fim, almeja-se tornar o NFV-TE dinâmico, permitindo a reconfiguração dos mecanismos de engenharia de tráfego baseada em políticas. Essas políticas, por sua vez, determinam as configurações de *policers* e *shapers* segundo estados e condições performáticas observadas em uma determinada rede.

APÊNDICE C – PUBLICAÇÕES E SUBMISSÕES

C.1 TRABALHOS NO ÂMBITO DA TESE

1. **Fulber-Garcia, V.**; Marcuzzo, L. d. C.; Souza, G. V. d.; Bondan, L.; Nobre, J. C.; Schaeffer-Filho, A. E.; Santos, C. R. P. d.; Granville, L. Z.; Duarte Jr., E. P.. An NSH-Enabled Architecture for Virtualized Network Function Platforms. *International Conference on Advanced Information Networking and Applications* (AINA). 2019.

2. **Fulber-Garcia, V.**; Marcuzzo, L. d. C.; Huff, A.; Bondan, L.; Nobre, J. C.; Schaeffer-Filho, A. E.; Santos, C. R. P. d.; Granville, L. Z.; Duarte Jr., E. P.. On the Design of a Flexible Architecture for Virtualized Network Function Platforms. *Global Communications Conference (GLOBECOM)*. 2019.

3. **Fulber-Garcia, V.**; Luizelli, M. C.; Duarte Jr., E. P.; Santos, C. R. P. d.. Uma Solução Flexível e Personalizável para a Composição de Cadeias de Função de Serviço. *Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos - Workshop de Gerência e Operação de Redes e Serviços* (SBRC - WGRS). 2019.

4. Flauzino, J.; **Fulber-Garcia, V.**; Venâncio, G.; Duarte Jr., E. P.. Além do OpenStack: Disponibilizando o Suporte para Funções Virtualizadas de Rede NFV-MANO no CloudStack. *Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos* (SBRC). 2020.

5. **Fulber-Garcia, V.**; Tavares, T. N.; Marcuzzo, L. d. C.; Souza, G. V. d.; Franco, M. F.; Bondan, L.; Schaeffer-Filho, A. E.; Santos, C. R. P. d.; Turck, F. d.; Granville, L. Z.; Duarte Jr., E. P.. On the Design and Development of Emulation Platforms for NFV-based Infrastructures. *International Journal of Grid and Utility Computing* (IJGUC). 2020.

6. **Fulber-Garcia, V.**; Santos, C. R. P. d.; Spinosa, E. J.; Duarte Jr., E. P.. Mapeamento Customizado de Serviços de Rede em Múltiplos Domínios Baseado em Heurísticas Genéticas. *Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos* (SBRC). 2020.

7. **Fulber-Garcia, V.**; Luizelli, M. C.; Santos, C. R. P. d.; Duarte Jr., E. P.. CUSCO: A Customizable Solution for NFV Composition. *International Conference on Advanced Information Networking and Applications* (AINA). 2020.

8. **Fulber-Garcia, V.**; Huff, A.; Santos, C. R. P. d.; Duarte Jr., E. P.. Network Service Topology: Formalization, Taxonomy and the CUSTOM Specification Model. *Computer Networks* (ComNet). 2020.

9. Huff, A.; Souza, G. V. d.; **Fulber-Garcia, V.**; Duarte Jr., E. P.. Building Multi-domain Service Function Chains Based on Multiple NFV Orchestrators. *Conference on Network Function Virtualization and Software Defined Networks* (NFV-SDN). 2020.

10. **Fulber-Garcia, V.**; Flauzino, J.; Duarte Jr., E. P.. Destravando o Element Management System: Permitindo a Gerência Holística de Funções de Rede Virtualizadas. *Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos* (SBRC). 2021.

11. Flauzino, J.; **Fulber-Garcia, V.**; Huff, A.; Venâncio, G.; Duarte Jr., E. P.. Gerência e Orquestração de Funções e Serviços de Rede Virtualizados em Nuvem CloudStack. *Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos - Workshop de Gerência e Operação de Redes e Serviços* (SBRC - WGRS). 2021.

12. Souza, G. V. d.; **Fulber-Garcia, V.**; Marcuzzo, L. d. C.; Tavares, T. N.; Franco, M. F.; Bondan, L.; Schaeffer-Filho, A. E.; Santos, C. R. P. d.; Granville, L. Z.; Duarte Jr., E. P.. Beyond VNFM: Filling the Gaps of the ETSI VNF Manager to Fully Support VNF Life Cycle Operations. *International Journal of Network Management* (IJNM). 2021.

13. **Fulber-Garcia, V.**; Huff, A.; Marcuzzo, L. d. C.; Luizelli, M. C.; Schaeffer-Filho, A. E.; Granville, L. Z.; Santos, C. R. P. d.; Duarte Jr., E. P.. Customizable Deployment of NFV Services. *Journal of Network and Systems Management* (JNSM). 2021.

14. **Fulber-Garcia, V.**; Venâncio, G.; Duarte Jr., E. P.. Arquitetura e Mapeamento de Serviços Virtualizados de Rede Tolerantes a Falhas e Intrusão. *Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos - Workshop de Testes e Tolerância a Falhas* (SBRC - WTF). 2022.

15. Quiles, F. R.; Moreira, J. V.; **Fulber-Garcia, V.**; Duarte Jr., E. P.. NFV-TE: Uma Ferramenta para a Geração Automática de Funções Virtuais Rede para Engenharia de Tráfego. *Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos* (SBRC). 2022.

16. **Fulber-Garcia, V.**; Luizelli, M. C.; Santos, C. R. P. d.; Spinosa, E. J.; Duarte Jr., E. P.. Intelligent Mapping of Virtualized Services on Multi-Domain Networks. *International Conference on Intelligent Systems Design and Applications* (ISDA). 2022.

17. Quiles, F. R.; Moreira, J. V.; **Fulber-Garcia, V.**; Duarte Jr., E. P.. NFV-TE: Geração Automática de Funções Virtualizadas para Engenharia de Tráfego de Rede. *Revista Eletrônica de Iniciação Científica em Computação*. 2022. **[Submetido, em avaliação]**

18. **Fulber-Garcia, V.**; Flauzino, J.; Santos, C. R. P. d.; Duarte Jr., E. P.. HoLMES: A Comprehensive EMS to Unleash the Power of Holistic NFV Management. *Network Operations and Management Symposium*. 2023. **[Submetido, em avaliação]**

C.2 TRABALHOS NO ÂMBITO DE OUTRAS PESQUISAS

1. Camargo, F. E. d.; **Fulber-Garcia, V.**; Duarte Jr., E. P.. Uma Estratégia Bioinspirada para Escalonamento em Redes Sem Fio sob o Modelo SINR. *Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos - Workshop de Gerência e Operação de Redes e Serviços* (SBRC - WGRS). 2022.

2. **Fulber-Garcia, V.**; Engel, F.; Duarte Jr., E. P.. SK-Greedy: A Simple Heuristic Algorithm for Baseline Schedules in Wireless Networks under the SINR Model. *Latin-American Symposium on Dependable Computing - Workshop on Security, Privacy and Reliability on Wireless Sensing Networks* (LADC - WSensing). 2022.

3. **Fulber-Garcia, V.**; Engel, F.; Duarte Jr., E. P.. A Bioinspired Scheduling Strategy for Dense Wireless Networks under the SINR Model. *International Conference on Intelligent Systems Design and Applications* (ISDA). 2022.