# Uma Proposta para Auto-Configuração Especializada de Elementos de Borda em Redes SDN Utilizando a Arquitetura SONAr

**Daniel Ricardo Cunha Oliveira**


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

Uberlândia
2020

**Daniel Ricardo Cunha Oliveira**

# Uma Proposta para Auto-Configuração Especializada de Elementos de Borda em Redes SDN Utilizando a Arquitetura SONAr

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

Área de concentração: Ciência da Computação

Orientador: Prof. Flávio de Oliveira Silva, Ph.D.
Coorientador: Prof. Pedro Frosi Rosa, Ph.D.

Uberlândia
2020

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

O48
2020

Oliveira, Daniel Ricardo Cunha, 1974-
Uma Proposta para Auto-Configuração Especializada de Elementos de Borda em Redes SDN Utilizando a Arquitetura SONAr [recurso eletrônico] / Daniel Ricardo Cunha Oliveira. - 2020.

Orientador: Flávio de Oliveira Silva.
Coorientador: Pedro Frosi Rosa.
Dissertação (Mestrado) - Universidade Federal de Uberlândia, Pós-graduação em Ciência da Computação.
Modo de acesso: Internet.
Disponível em: http://doi.org/10.14393/ufu.di.2021.7
Inclui bibliografia.
Inclui ilustrações.

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

CDU: 681.3

Bibliotecários responsáveis pela estrutura de acordo com o AACR2:

Gizele Cristine Nunes do Couto - CRB6/2091

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

# ATA DE DEFESA - PÓS-GRADUAÇÃO

| | | | | | |
|---|---|---|---|---|---|
| Programa de Pós-Graduação em: | Ciência da Computação | | | | |
| Defesa de: | Mestrado Acadêmico, 42/2020, PPGCO | | | | |
| Data: | 22 de dezembro de 2020 | Hora de início: | 17:15 | Hora de encerramento: | 19:33 |
| Matrícula do Discente: | 11812CCP013 | | | | |
| Nome do Discente | Daniel Ricardo Cunha Oliveira | | | | |
| Título do Trabalho: | Uma Proposta para Auto-Configuração Especializada de Elementos de Borda em Redes SDN Utilizando a Arquitetura SONAr | | | | |
| Área de concentração: | Ciência da Computação | | | | |
| Linha de pesquisa: | Sistemas de Computação | | | | |
| Projeto de Pesquisa de vinculação: | - | | | | |

Reuniu-se, por videoconferência, a Banca Examinadora, designada pelo Colegiado do Programa de Pós-graduação em Ciência da Computação, assim composta: Professores Doutores: Rodrigo Sanches Miani - FACOM/UFU; Antônio Jorge Gomes Abelém - UFPA; Pedro Frosi Rosa - FACOM/UFU (coorientador) e Flávio de Oliveira Silva - FACOM/UFU, orientador do candidato.

Os examinadores participaram desde as seguintes localidades: Antônio Jorge Gomes Abelém - Belém/PA; Rodrigo Sanches Miani, Pedro Frosi Rosa e Flávio de Oliveira Silva - Uberlândia/MG. O discente participou da cidade de Uberlândia/MG.

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

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

**Aprovado.**

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

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

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

---

Documento assinado eletronicamente por **Flávio de Oliveira Silva**, **Professor(a) do Magistério Superior**, em 11/01/2021, às 11:02, conforme horário oficial de Brasília, com fundamento no art. 6º, § 1º, do Decreto nº 8.539, de 8 de outubro de 2015.

Documento assinado eletronicamente por **Antônio Jorge Gomes Abelém**, **Usuário Externo**, em 11/01/2021, às 15:57, conforme horário oficial de Brasília, com fundamento no art. 6º, § 1º, do Decreto nº 8.539, de 8 de outubro de 2015.

Documento assinado eletronicamente por **Rodrigo Sanches Miani**, **Professor(a) do Magistério Superior**, em 11/01/2021, às 20:15, conforme horário oficial de Brasília, com fundamento no art. 6º, § 1º, do Decreto nº 8.539, de 8 de outubro de 2015.

Documento assinado eletronicamente por **Pedro Frosi Rosa**, **Professor(a) do Magistério Superior**, em 15/01/2021, às 00:03, conforme horário oficial de Brasília, com fundamento no art. 6º, § 1º, do Decreto nº 8.539, de 8 de outubro de 2015.

A autenticidade deste documento pode ser conferida no site https://www.sei.ufu.br/sei/controlador_externo.php?acao=documento_conferir&id_orgao_acesso_externo=0, informando o código verificador **2470142** e o código CRC **A755C9B7**.

---

**Referência:** Processo nº 23117.076432/2020-31 SEI nº 2470142

*Este trabalho é dedicado a todos os acadêmicos e pesquisadores das Universidades públicas do Brasil, que mantém acesa a chama da pesquisa e do desenvolvimento do conhecimento brasileiro, mesmo nos tempos mais difíceis. Nosso respeito e nosso apoio são incondicionais.*

# Agradecimentos

Agradeço de coração todos os que me apoiaram nessa jornada, que nunca é fácil, mas torna-se menos árdua com a ajuda de pessoas especiais.

Em primeiro lugar, minha esposa Prof. Alessandra Araújo, MSc. – ela também uma dedicada acadêmica –, que sempre me incentivou em todas minhas iniciativas, incluindo o desafio de voltar à academia após mais de vinte anos ausente.

Ao Prof. Flávio Oliveira, que me aceitou como orientando e tem lutado tanto, e com sucesso, por concretizar parcerias entre Universidade e Indústria, e ao Prof. Pedro Frosi, que foi quem, em uma conversa por telefone, me convidou e incentivou a compôr o quadro discente da pós-graduação da FACOM/UFU.

Aos meus colegas e agora bons amigos Maurício e Natal, os "SONAr Fellas", pelo imenso apoio que recebi de vocês. Aprendi mais com vocês do que em qualquer outra etapa dessa pós-graduação e posso afirmar que, sem fazer parte desse time, certamente não haveria um final feliz nesta história.

À minha família: minha mãe Maria Tereza, minha sogra Eleuza Araújo e irmãos Tatiana e Fábio, aqueles que trazem sentido à vida e também sempre têm uma palavra de incentivo para nos ajudar a continuar.

A todos amigos e amigas de quem me ausentei durante essa etapa, mas nunca deixaram de estar ao meu lado: prometo voltar à convivência normal – ao menos até o início do doutorado...

*"O caminho do arquiteto é tornar-se proficiente no uso de suas ferramentas, primeiro para fazer planos realistas e, depois, para desempenhar o trabalho de acordo com o plano. (...) Passo a passo, percorra a estrada de mil quilômetros. (...) Hoje é a vitória sobre o você de ontem."*

*(Miyamoto Musashi, O Livro dos Cinco Anéis)*

*"Those friends thou hast, and their adoption tried;*
*Grapple them to thy soul with hoops of steel"*

*(William Shakespeare, Hamlet)*

# Resumo

A crescente complexidade das redes de computadores para suportar as atuais aplicações vem sendo percebida como um fator limitante em termos técnicos e econômicos para sua evolução. Uma resposta a esta questão é a proposta de novas arquiteturas de redes autônomas, baseadas em *Software-Defined Networks* (SDN), com implementação de funcionalidades chamadas *self-**. Destas funcionalidades, uma que se destaca é a *self-configuration* (auto-configuração). Alguns modelos de auto-configuração já foram propostos em um nível mais alto de definição, porém ainda há bastante trabalho prático a ser feito nesse sentido. Neste trabalho, o objetivo é a modelagem, a implementação e a testagem da proposta para um sistema de auto-configuração especializada de redes uma vez que um novo elemento de borda tenha sido inserido. Este sistema deverá reconhecer o elemento e configurar um determinado segmento de rede de modo que o mesmo funcione em um modelo "*plug-and-play*". A abordagem será baseada na arquitetura *Self-Organizing Networks Architecture* (SONAr), que está sendo desenvolvida por um grupo de pesquisadores da Faculdade de Computação da Universidade Federal de Uberlândia, e será incorporada como parte funcional neste projeto. Os resultados obtidos demonstram a viabilidade prática da funcionalidade implementada tanto em termos de efetividade quanto de eficiência, com seu desempenho superando em várias ordens de grandeza o tempo necessário para configuração manual de uma rede específica como as usadas em telecomunicações.

**Palavras-chave:** SON. Self-Organized Networks. SDN. Self-configuration. SONAr.

# Abstract

The increasing complexity of computer networks to support current applications has been perceived as a technical and economic limiting factor for their evolution. One answer to this matter is the proposal of new autonomic network architectures based on Software-Defined Networks (SDN), with implementation of features called self- *. Of these features, one that stands out is self-configuration. Some self-configuration models have already been proposed at a higher level of definition, but there is still a lot of practical work to be done in this regard. In this work, the goal is the modeling, implementation and testing of a specialized network self-configuration system when a new edge element has been inserted. This system must recognize the element and configure a specific network segment in a way that it works on a "plug-and-play model". The approach will be based on the Self-Organized Networks Architecture (SONAr) framework, which is being developed by a group of researchers from the Faculty of Computing of the Federal University of Uberlândia, and will be incorporated as a functional part of it. The results obtained demonstrate the practical feasibility of the implemented functionality both in terms of effectiveness and efficiency, with its performance exceeding by several orders of magnitude the time required for manual configuration of a specific network such as the used in telecommunications.

**Keywords:** SON. Self-Organized Networks. SDN. Self-configuration. SONAr.

# Lista de ilustrações

# Lista de tabelas

# Lista de siglas

**3GPP** *3rd Generation Partnership Project*

**ACoE** *Alarms Collecting Entities*

**ANR** *Automatic Neighbour Relation*

**AP** *Address Provider*

**API** *Application Programming Interface*

**ARP** *Address Resolution Protocol*

**BSLE** *Behavior Pattern Self-Learning Entity*

**BSS** *Business Support System*

**BTS** *Base Transceiver Station*

**CoE** *Collecting Entity*

**CoS** *Class of Service*

**COTS x86** *Commercial off-the-shelf x86*

**CP** *Control Plane*

**CPI** *Control Primitives Interceptor*

**CRAN** *Cloud Radio Access Network*

**DDoS** *Distributed Denial-of-Service*

**DHCP** *Dynamic Host Configuration Protocol*

**DSCP** *Differentiated Services Code Point*

**DSLAM** *Digital Subscriber Line Access Multiplexer*

**DSLE** *Diagnostics Self-Learning Entity*

**DWDM** *Dense Wavelength Division Multiplexing*

**EM** *Element Manager*

**EMS** *Element Management System*

**ERB** Estação Rádio-Base

**ETSI** *European Telecommunications Standards Institute*

**FCAPS** *Fault, Configuration, Accounting, Performance, Security*

**GNS3** *Graphical Network Simulator-3*

**HSS** *Home Subscriber Server*

**IA** Inteligência Artificial

**ICMP** *Internet Control Message Protocol*

**IEEE** *Institute of Electrical and Electronics Engineers*

**IOS** *Internetwork Operating System*

**IoT** *Internet-of-Things*

**IP** *Internet Protocol*

**IPFIX** *IP Flow Information Export*

**IPV4** *Internet Protocol Version 4*

**ISG** *Industry Specification Group*

**ISP** *Internet Service Provider*

**JDK** *Java Development Kit*

**JSON** *JavaScript Object Notation*

**LCoE** *Logs Collecting Entity*

**LLDP** *Link Layer Discovery Protocol*

**LTE** *Long Term Evolution*

**LTE-A** *Long Term Evolution Advanced*

**MAC** *Medium Access Control*

**MANO** *Management and Orchestration*

**MCoE** *Metrics Collecting Entity*

**MEC** *Mobile Edge Computing*

**MME** *Mobility Management Entity*

**MPLS** *Multiprotocol Label Switching*

**NAD** *Network Administration Dashboard*

**NBI** *Northbound Interface*

**NDB** *Network Database*

**NE** *Network Element*

**NEM** *Network Element Manager*

**NETCONF** *Network Configuration*

**NFV** *Network Functions Virtualization*

**NFVI** *Network Function Virtualization Infrastructure*

**NFVO** *Network Function Virtualization Orchestrator*

**NS** *Network Slicing*

**NSB** *Network Service Broker*

**NSLE** *Natural Language Processing Self-Learning Entity*

**OAM** *Operation, Administration and Management*

**OLT** *Optical Line Terminal*

**ONF** *Open Networking Foundation*

**ONOS** *Open Networking Operating System*

**OSI** *Open System Interconnection*

**OSS** *Operations Support System*

**OvS** *Open vSwitch*

**P** *Provider*

**PBNM** *Policy-based Network Management*

**PCP** *Priority Point Code*

**PCI** *Physical Cell Identity*

**PCRF** *Policy and Charging Resource Function*

**P-GW** *Packet Data Network Gateway*

**PE** *Provider Edge*

**PNF** *Physical Network Function*

**PSLE** *Prediction Self-Learning Entity*

**QoE** *Quality of Experience*

**QoS** *Quality of Service*

**RAN** *Radio Access Network*

**RCoE** *Results Collecting Entity*

**RDCL** *Reusable Function Blocks Description and Composition Language*

**REE** *Reusable Function Blocks Execution Environment*

**REST** *Representational State Transfer*

**RFB** *Reusable Function Block*

**RSLE** *Rating Self-Learning Entity*

**RSPAN** *Remote Switched Port Analyzer*

**RSTP** *Rapid Spanning Tree Protocol*

**SAE-GW** *System Architecture Evolution Gateway*

**SBI** *Southbound Interface*

**SCE** *Self-Configuration Entity*

**SCoE** *Samples Collecting Entity*

**SDN** *Software-Defined Networks*

**SDNC** *Software-Defined Network Controller*

**S-GW** *Serving Gateway*

**SHE** *Self-Healing Entity*

**SLA** *Service Level Agreement*

**SLE** *Self-Learning Entity*

**SNMP** *Simple Network Management Protocol*

**SO** Sistema Operacional

**SOE** *Self-Organizing Entity*

**SON** *Self-Organizing Network*

**SONAr** *Self-Organizing Networks Architecture*

**SOPE** *Self-Optimization Entity*

**SORE** *Self-Orchestration Entity*

**SPE** *Self-Protection Entity*

**SPLE** *Self-Planning Entity*

**SRE** *SONAr Results Extractor*

**SSCD** *Specialized Self-Configuration Description*

**SSCM** *Specialized Self-Configuration Module*

**STP** *Spanning Tree Protocol*

**TCoE** *Topology Collecting Entity*

**TI** Tecnologia da Informação

**TSLE** *Tuning Self-Learning Entity*

**UE** *User Equipment*

**UP** *User Plane*

**WAN** *Wide Area Network*

**vCore** *Virtual Core*

**VIM** *Virtualized Infrastructure Manager*

**VLAN** *Virtual Local Area Network*

**VID** *VLAN Id*

**VM** *Virtual Machine*

**VNF** *Virtual Network Function*

**VNFM** *Virtualized Network Function Manager*

**VoIP** *Voice over Internet Protocol*

**YANG** *Yet Another Next Generation*

# Sumário

# Capítulo 1

# Introdução

Se há um fato inquestionável na história recente da tecnologia, é o crescimento exponencial das aplicações e a densificação de dispositivos conectados à Internet, em especial se levarmos em consideração o acesso em dispositivos móveis. De fato, segundo relatório publicado pela Ericsson (ERICSSON, 2019), o tráfego global de dados móveis saiu de um patamar de cerca de 3 Exabytes/mês do início de 2014, para cerca de 37 EB/mês no terceiro quarto de 2019, ou seja, um crescimento de 1.000% em aproximadamente cinco anos. E a tendência é de que esse crescimento se mantenha em patamares exponenciais, sobretudo com adventos tecnológicos como a TV de ultra-resolução 8K, novos dispositivos de realidade virtual/aumentada, *cloud gaming*, *Internet-of-Things* (IoT) massiva e outras aplicações que serão possíveis através de novas tecnologias de acesso, como o 5G. Toda essa evolução traz, naturalmente, uma preocupação intrínseca com a infraestrutura que irá suportar toda essa carga.

Já em 2001, a IBM demonstrava uma apreensão com o aumento da complexidade das redes de computadores, além do respectivo aumento do custo que isso implica (GANEK; CORBI, 2003), em um artigo que define as bases e conceitos para computação autonômica (do inglês *autonomic computing*), incluindo os conceitos de *self-**.

As redes interconectaram sistemas distribuídos e heterogêneos da indústria de TI, criando uma rede global altamente complexa, que requer cada vez mais profissionais altamente especializados para instalar, configurar, operar, ajustar e manter o sistema. Nesse contexto, a IBM utilizou a definição de "computação autonômica" para representar a sua visão de como endereçar os desafios do aumento da complexidade das atuais redes de computadores. A expressão é uma analogia ao sistema nervoso autônomo, responsável por liberar o cérebro da carga de ter que lidar com funções vitais, porém de baixo nível, como a respiração, batimentos cardíacos, pressão arterial, controle de temperatura, fome, sede, etc (MCCORRY, 2007). Porém, somente no ano de 2011 foi criado um padrão que poderia conter a resposta definitiva a esse desafio: o conceito de *Software-Defined Networks*, ou SDN.

A *Open Networking Foundation* (ONF) – que viria a definir e padronizar as SDNs –

já existia há algum tempo, porém foi o principal destaque em maio de 2011 na evento Interop 2011 em Las Vegas, chamando a atenção do público para um laboratório focado no protocolo OpenFlow e suas implementações na prática. Segundo a definição da ONF (ONF, 2020b): "*Software-Defined Networking* (SDN) é uma arquitetura emergente que é dinâmica, gerenciável, eficiente em custos e adaptável, sendo ideal para as atuais aplicações de altas larguras de banda e de natureza dinâmica. Essa arquitetura desacopla as funções de controle e de encaminhamento da rede, permitindo que o controle da mesma torne-se diretamente programável e que a estrutura de mais baixo nível seja abstraída para aplicações e serviços de redes."

Com o advento das SDNs, houve alguma evolução nos modelos de redes auto gerenciáveis, com vários *frameworks* e modelos de arquitetura propostos, dos quais alguns se destacam, e serão mencionados na Seção 2.2.

Por fim, algumas propriedades foram definidas para se caracterizar uma rede como auto-gerenciável. Ainda no artigo da IBM (GANEK; CORBI, 2003), quatro funcionalidades fundamentais deveriam ser tratados por esse tipo de rede, sendo eles: auto-configuração (do inglês *self-configuration*), auto-cura (*self-healing*), auto-otimização (*self-optimization*) e auto-proteção (*self-protection*).

Neste trabalho, o foco será dado na primeira funcionalidade, a de auto-configuração, que será proposta em um contexto de um *framework* mais completo denominado SONAr, em desenvolvimento por pesquisadores da Faculdade de Computação da Universidade Federal de Uberlândia, e que fornece a base para implementação das funcionalidades self-* que compõem um sistema de *Self-Organizing Network* (SON). A arquitetura SONAr será apresentada em mais detalhes na Subseção 2.1.7.

## 1.1 Objetivos e Desafios da Pesquisa

O principal objetivo da pesquisa é investigar as alternativas para implementação de um sistema de auto-configuração de rede, preparando-a automaticamente para o recebimento de um elemento de borda com requerimentos específicos (isto é, que necessite configurações especiais que não sejam restritas ao caso geral, como a que ocorre na introdução de switches ou roteadores). Para isso, é necessário definir-se o modelamento de uma maneira de se descrever esses requerimentos (o que fazer e onde fazer) e realizar-se a análise de um caso prático desse processo em uma rede. Também é considerada que toda a abordagem seja realizada seguindo-se o modelo da arquitetura SONAr. O desafio é que toda estrutura necessária – incluindo encaminhamento, configuração de parâmetros de *Quality of Service* (QoS), *Virtual Local Area Network*s (VLANs), etc –, que é particular a cada caso de elemento de borda, seja configurada sem intervenção humana, e de forma mais rápida do que se fosse feita manualmente por um especialista.

Essa investigação será feita em um contexto da indústria, ou seja, utilizando cenários

reais usados na prática para que haja uma aplicabilidade efetiva desta implementação. Os testes serão feitos em ambientes virtualizados utilizando-se as aplicações reais, na forma de imagens dos softwares que rodam em elementos comerciais, em máquinas virtuais e/ou *containers*. Os exemplos serão retirados de casos práticos observados em uma operadora de rede e levantados através de entrevistas e pesquisas. Em resumo, além da descrição do modelamento a ser adotado, será realizada uma simulação com o intuito de se avaliar a efetividade da solução proposta para o caso geral e comparação de desempenho da mesma em termos de tempo de configuração em diversas topologias e escalas diferentes.

## 1.2 Hipótese

Utilizando-se a arquitetura SONAr, é possível desenvolver-se um modelo de auto-configuração para elementos de rede de borda específicos de modo versátil – ou seja, que consiga auto-configurar-se em um modelamento de rede não necessariamente genérico – e de forma mais rápida que se fosse feito de forma manual.

## 1.3 Contribuições

A principal contribuição será, naturalmente, a de apresentar um processo que automatiza um trabalho complexo e normalmente realizado por especialistas, com muita aderência às atuais necessidades das indústrias de telecomunicações e de Tecnologia da Informação (TI). Conforme já comentado, a quantidade de novos elementos a serem inseridos nas redes das operadoras de telecomunicações e provedoras de serviços de Internet está aumentando rapidamente: espera-se que o 5G seja a tecnologia de comunicação móvel de lançamento mais rápido na história da tecnologia (ERICSSON, 2019). Nesse contexto, um sistema de auto-configuração não só deverá tornar-se desejável, mas totalmente necessário nos próximos anos, para manutenção da viabilidade do negócio dessas empresas. Em uma abordagem mais genérica, o aumento da compexidade das redes irá demandar uma solução de redes auto-organizáveis com várias funcionalidades – em especial as quatro que definem o conceito de SON: auto-configuração, auto-otimização, auto-cura e auto-proteção (GANEK; CORBI, 2003).

Um sistema de auto-configuração em um contexto genérico pode não ser totalmente aplicável à indústria, uma vez que há muitas especificidades nas configurações das redes, especialmente no caso de uma rede de TI baseada em *Internet Protocol* (IP) que suporta uma infraestrutura de telecomunicações. Neste sentido, essa primeira facilidade (auto-configuração), a qual será o objeto de investigação desse trabalho, será inserido em um contexto mais específico, onde demanda-se uma descrição clara do que se quer configurar, em que elementos essa configuração deve ser aplicada e em quais situações. Essa solução também deverá ser flexível para permitir sua adequação a todos os casos ou, ao menos,

aos casos identificados nesse estudo. Além disso, deve prever também a existência de elementos de redes legadas, embora o escopo da auto-configuração especializada deva limitar-se aos elementos SDN.

## 1.4 Organização da Dissertação

Esta dissertação está organizada da seguinte forma: o Capítulo "Fundamentação Teórica" apresentará tanto os conceitos básicos teóricos de *Network Functions Virtualization* (NFV), SDN, SON, redes móveis (para fundamentação da aplicação que será proposta no trabalho), de auto-configuração de redes e alguns protocolos e implementações atualmente utilizados, além dos trabalhos correlatos já disponíveis na literatura especializada mundial, que descrevem os esforços já realizados na investigação de questões similares. Em especial, irá também apresentar a estrutura completa do *framework* SONAr e as funcionalidades de auto-configuração genéricas já desenvolvidas nessa arquitetura (*bootstrapping* e *plug-and-play*).

Logo após, o Capítulo "Módulo de Auto-configuração Especializada" irá mostrar a proposta do trabalho em si, apresentando como a auto-configuração de rede para elementos de borda é implementada na prática. Para isso, serão descritos os módulos, modelos e interfaces desenvolvidos, além da forma em que a funcionalidade insere-se no contexto do SONAr.

O Capítulo "Experimentos e Análise dos Resultados" focará na implementação prática do ambiente para experimentações, além dos próprios experimentos realizados e a investigação da eficácia e eficiência do sistema de auto-configuração. Será feita a apresentação do cenário escolhido para implementação prática e dos métodos utilizados para as experimentações, bem como uma avaliação dos resultados alcançados.

Finalmente, o Capítulo "Conclusão" apresentará as principais contribuições obtidas no capítulo anterior de forma sumarizada, além de apresentar sugestões para a evolução do trabalho em futuras investigações e a produção bibliográfica resultante do presente trabalho.

# Capítulo 2

# Fundamentação Teórica

Nesse capítulo serão apresentados alguns conceitos básicos necessários para plena compreensão do trabalho, bem como um levantamento de trabalhos correlatos que já foram desenvolvidos sobre o tema. A partir deste capítulo serão mantidas as terminologias específicas em inglês, utilizando-se "*Self-Configuration*" ao invés de "Auto-Configuração" ou "*Virtual Network Function*" (VNF) ao invés de "Função Virtual de Rede", por exemplo. O motivo para tal é facilitar a correlação de termos já bem conhecidos na literatura internacional, uma vez que a tradução livre desses termos pode suscitar dúvidas sobre a verdadeira correspondência com seus respectivos originais.

## 2.1 Conceitos Básicos

Para que o projeto aqui apresentado possa ser plenamente compreendido, alguns conceitos básicos devem ser apresentados. São eles: NFV, SDN, protocolo OpenFlow, switch virtual *Open vSwitch* (OvS), SON, auto-configuração de redes, SONAr e redes móveis.

### 2.1.1 Network Functions Virtualization (NFV)

O conceito de virtualização de funções de rede, ou NFV, nasceu de um *white paper* publicado em outubro de 2012 no "*SDN and OpenFlow World Congress*", em Darmstadt, Alemanha (CHIOSI et al., 2012). Esse documento é de autoria de representantes das maiores operadoras de telecomunicações mundiais, incluindo AT&T, British Telecom, China Mobile, Deutsche Telekom, NTT, Orange, Telefonica, Telstra e Verizon, que formaram o grupo de especificação *Industry Specification Group* (ISG) dentro do *European Telecommunications Standards Institute* (ETSI) (YI et al., 2018). Nele, é colocada a dificuldade de se trabalhar com uma variedade de hardwares proprietários, e a necessidade de se transformar a maneira com que as operadoras fazem a arquitetura de suas redes. A proposta era que essa nova arquitetura fosse implementada com a substituição dos hardwares proprietários por funções de rede virtualizadas que rodassem sobre uma infraestrutura de

servidores padrões da indústria, os chamados *Commercial off-the-shelf x86* (COTS x86), localizados em *datacenters*, nós de rede ou nas premissas dos usuários finais. Desta forma, o lançamento de novas funções de rede poderia ser feita com a instanciação de máquinas virtuais que rodariam nesses servidores. No mesmo documento, os autores apontam nove grandes benefícios dessa nova arquitetura, que vão desde a redução de custos de equipamentos, melhoria do *time to market* dos serviços, até redução do consumo de energia. Por fim, também já colocam pontos importantes a serem endereçados na nova arquitetura – como a necessidade de uma entidade de gerenciamento e orquestração e da automação da estrutura (CHIOSI et al., 2012).

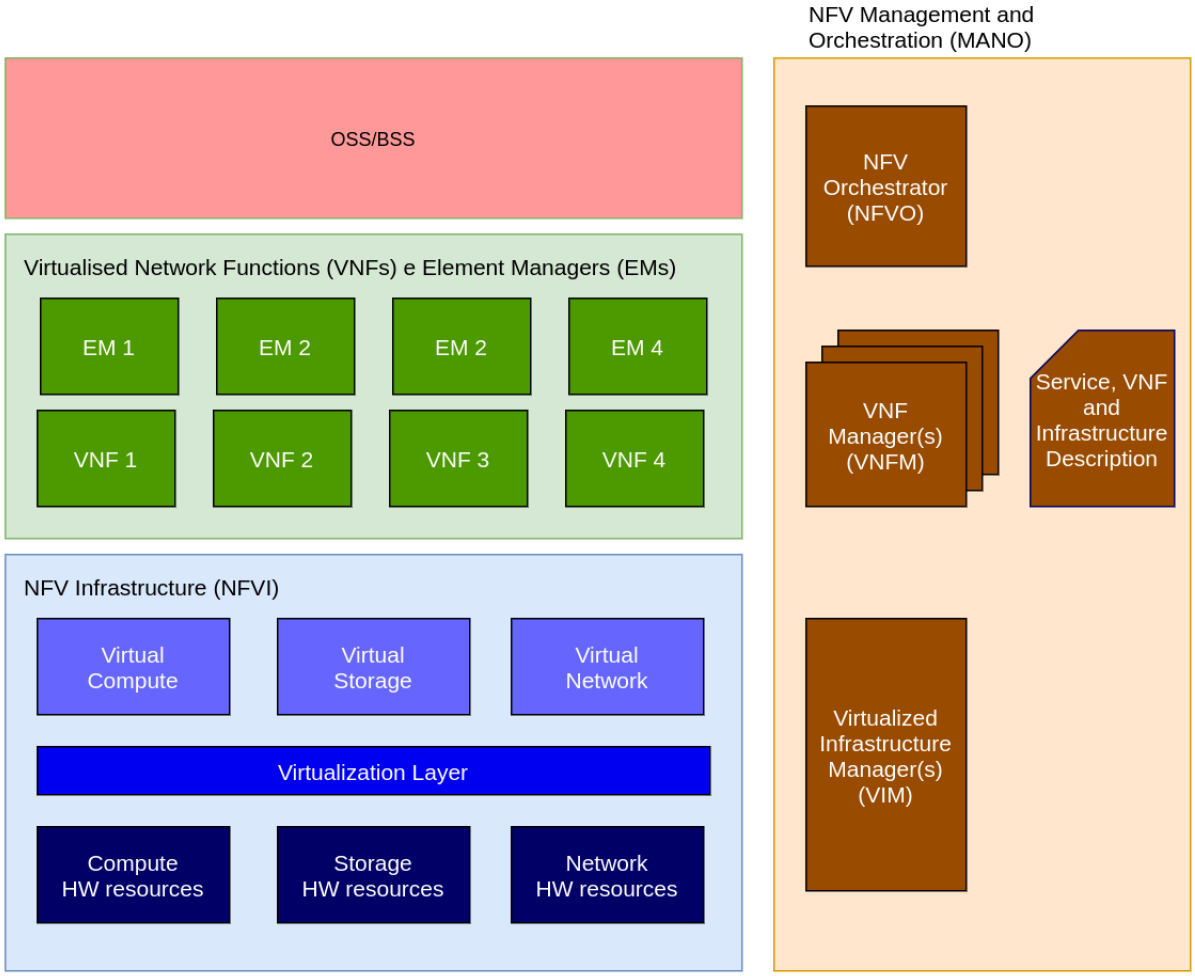
No contexto NFV, é necessário que se defina alguns termos (YI et al., 2018):

- *Physical Network Function* (PNF): É a implementação de um bloco de função especializada com interfaces e comportamento externo bem definidos, podendo ser, por exemplo, um nó de rede ou um *appliance* físico.
- *Network Function Virtualization Infrastructure* (NFVI): A NFVI provê um ambiente de rede composto tanto de componentes de hardware quanto de software, nas quais as *Virtual Network Function*s (VNFs) possam ser implementadas, gerenciadas e executadas.
- *Element Management System* (EMS): É um conjunto de gerenciadores de elementos individuais, chamados *Element Manager*s (EMs), que são responsáveis pelo gerenciamento das VNFs em termos de instanciação, execução e lançamento durante seus ciclos de vida.
- *Management and Orchestration* (MANO): Como a NFV introduz algumas novas capacidade às redes, o MANO é o elemento que gerencia e acomoda essas novas capaciddaes. É dividido em *Virtualized Infrastructure Manager* (VIM), *Virtualized Network Function Manager* (VNFM) e *Network Function Virtualization Orchestrator* (NFVO), que são responsáveis pelo gerenciamento da NFVI, alocação de recursos, virtualização de funções, etc.
- *Virtual Network Function* (VNF): É a implementação de software de uma PNF, que deve prover as mesmas funcionalidades e interfaces de uma PNF. Podem ser constituídas de um únco componente, lançado em uma única *Virtual Machine* (VM), ou de vários componentes, cada um deles em uma VM distinta.

A arquitetura proposta pelo ISG no ETSI pode ser representada como mostrado na Figura 1. Para efeito de simplificação, a representação das interfaces foi omitida.

A *Virtualization Layer* situa-se entre a infraestrutura física e a infraestrutura virtual e é uma camada que implementa as VMs, gerenciando os recursos físicos da camada inferior com os recursos lógicos da camada superior. Há várias opções de implementações

Figura 1 – Arquitetura Básica do Framework NFV.




Fonte: adaptada de (ETSI, 2014)

disponíveis, como o *Microsoft Hyper-V, Citrix Xen, Linux KVM, VMware ESXi*, etc. A *Virtualization Layer* pode consistir de *hypervisors*, quando cada VM carrega também seu próprio Sistema Operacional (SO) ou de *containers*, quando a VM não necessita um SO separado para suas aplicações VNF (YI et al., 2018).

A última camada implementa as funções *Operations Support System* (OSS) e *Business Support System* (BSS), que compõem os sistemas de suporte da operação e do negócio de uma operadora de rede. Mais detalhes sobre OSS/BSS serão discutidos na Subseção 2.1.7.

A relevância do *framework* NFV para o projeto aqui apresentado é devido à natureza dos sistemas em que ele se aplica: o conceito de funções virtualizadas de rede tem sido muito adotado na prática pelas operadoras e suas infraestruturas e funções de rede. No 5G, por exemplo, da arquitetura do sistema é definida para incorporar o NFV, tanto no core de rede quanto no acesso, inclusive suportando diversos cenários de lançamentos virtualizados (3GPP, 2019a). Na demonstração da implementação do projeto, serão utilizadas switches

virtuais – a serem introduzidas na Subseção 2.1.4 – além de outros elementos de rede virtualizados, como será demonstrado nas próximas seções.

### 2.1.2 Software Defined Networking (SDN)

O SDN é um conceito relativamente novo no âmbito das redes de computadores, e é definido como uma arquitetura emergente onde o controle da rede é desacoplado do encaminhamento e é diretamente programável (ONF, 2012). Essa migração do controle – que no *status quo* é realizado nos dispositivos individuais de rede – para unidades computacionais acessíveis denominadas *Software-Defined Network Controller*s (SDNCs) permite que a infraestrutura seja abstraída para aplicações e serviços de rede, de forma que possa ser tratada como uma entidade lógica ou virtual (ONF, 2012). Essencialmente, o conceito SDN provê uma abstração de alto nível que permite o lançamento de serviços como uma combinação de fluxos e funções (GONÇALVES et al., 2020).

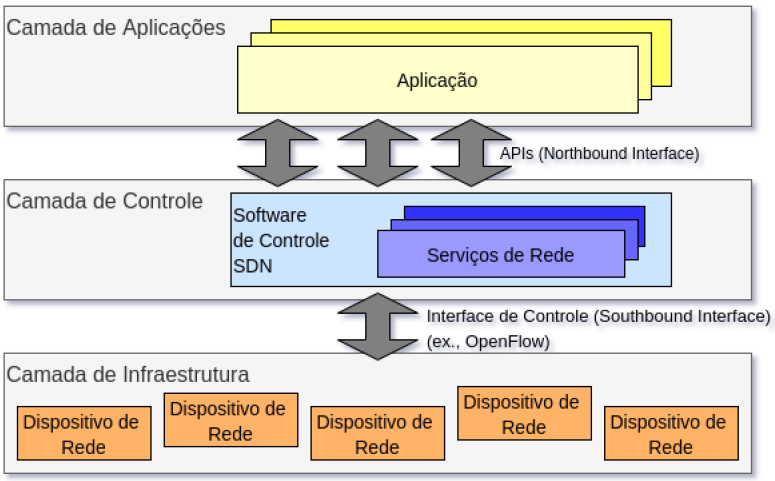
Vários esforços nesse sentido foram realizados por diversas entidades padronizadoras ou instituições acadêmicas, mas falharam em estabelecer-se como padrões *de facto*. As primeiras propostas de uso de software aberto (*open source*) para definição de uma arquitetura com separação entre planos de dados e de controle remontam a 2007, como o projeto Ethane, da Universidade de Stanford (CASADO et al., 2007), levando à criação do protocolo OpenFlow (MCKEOWN et al., 2008) em 2008. Em 2011 foi fundada a ONF, um consórcio sem fins lucrativos que agrega diversas instituições acadêmicas, operadoras e indústria, e tem como missão realizar a transformação dos modelos de infraestrutura de redes e modelos de negócios de suas operadoras (ONF, 2011). Na prática, a ONF é responsável pela especificação e padronização das normas técnicas da SDN, incluindo o protocolo OpenFlow.

A arquitetura básica do modelo SDN pode ser melhor compreendida visualizando-se a Figura 2, que será descrita a seguir.

A camada de infraestrutura é composta de todos os elementos da rede em si, incluindo os dispositivos finais dos usuários, switches, roteadores, servidores, etc. Para que estejam em um contexto SDN, estes elementos devem ser compatíveis com tal, ou seja, devem conseguir receber, interpretar e executar comandos de plano de controle através da interface *Southbound Interface* (SBI) que, no caso do padrão ONF, é implementada pelo protocolo OpenFlow. Esse protocolo em específico será abordado na próxima subseção. Uma implementação de switches SDN é a *Open vSwitch* (OvS), uma switch virtual de código aberto que também suporta interfaces de gerenciamento e protocolos padrões, como o *NetFlow*, *IP Flow Information Export* (IPFIX), *Remote Switched Port Analyzer* (RSPAN), etc. Também suporta distribuição sob múltiplos servidores físicos (LINUX FOUNDATION, 2016b). A OvS também será apresentada especificamente na Subseção 2.1.4.

A camada de controle inclui todos os controladores da rede. Esses controladores são implementados em servidores, normalmente máquinas virtuais em ambiente *in cloud*, com

Figura 2 – Arquitetura Básica do Modelo SDN.




Fonte: adaptada de (ONF, 2012)

um sistema operacional adequado ao controle da rede. Alguns desses sistemas operacionais dedicados a redes baseados em software aberto já estão disponíveis, como o *Open Networking Operating System* (ONOS), um projeto da ONF (ONOS PROJECT, 2020) ou o OpenDaylight (OPENDAYLIGHT PROJECT, 2018). Como já mencionado, os SDNCs comunicam-se com a camada de infraestrutura através da interface SBI, com *Application Programming Interfaces* (APIs) distintas para cada tarefa. Para configuração, pode ser usado o NETCONF YANG, por exemplo. Para monitoração, o *Simple Network Management Protocol* (SNMP) ou o próprio OpenFlow são suportados. Na função específica de controle de encaminhamento, o OpenFlow é o mais utilizado. Dependendo da abordagem, mais de um SDNC pode controlar uma rede, em modelo de *cluster* (CAMPANELLA; DANDOUSH; MANASSAKIS, 2017). Neste trabalho, o ONOS será utilizado como controlador.

Por fim, a camada de aplicação permite que sejam implementadas funções externas de monitoramento e engenharia de tráfego, além da virtualização da rede com *data centers multi-tenant*, dentre outras funcionalidades. As aplicações comunicam-se com o controlador através da interface *Northbound Interface* (NBI), que, como a interface SBI, pode ser implementada utilizando-se vários protocolos diferentes, normalmente na forma de uma API *RESTful*. Essa interface permite a integração com sistemas que podem fazer a automação da rede, o que é essencial em uma rede SDN. A Google, por exemplo, utiliza uma aplicação de sistema de engenharia de tráfego *Wide Area Network* (WAN) baseada em SDN, que permite uma melhor eficiência de tráfego e rápida convergência (VAHDAT; CLARK; REXFORD, 2015). As implementações de SON também são feitas nesta camada, de maneira que possam tanto ter a visibilidade do que acontece nos controladores e na rede em si (também através dos controladores), quanto atuar no sistema. Esta camada pode ser (e normalmente é) dividida em sub-camadas ela mesma: enquanto

as sub-camadas inferiores implementam as APIs de comunicação com os controladores, as superiores implementam funções de mais alto nível, como os processos de *analytics* dos dados recebidos da rede, determinação de atuação na mesma e disponibilização de *dashboards* de desempenho e interfaces gráficas de intervenção para usuários finais. Na Subseção 2.1.5 os sistemas de SON serão melhor detalhados.

Não há dúvidas quanto à necessidade de aderência das redes do futuro ao paradigma SDN, para que possam suportar, as funcionalidades inerentes ao 5G, como o *Network Slicing* (NS) e o *Mobile Edge Computing* (MEC), que irão possibilitar tanto o melhor gerenciamento de garantia do serviço quanto a baixa latência do mesmo, respectivamente. Porém, grandes desafios ainda terão de ser superados até que essa solução seja tecnicamente viável, notadamente seis pontos em específico (ZAIDI et al., 2018):

- *Fronthaul*: heterogeneidade dos meios de transmissão, hardware e protocolos de transporte.
- Latência de plataformas de propósito geral: a virtualização de elementos de rede móvel, como unidades de banda-base de estações rádio-base, impõem uma restrição na latência muitas vezes incompatível com as necessidades da sinalização de rede.
- Compatibilidade reversa: durante o período de transição do modelo tradicional ao *full* SDN, desafios de compatibilidade entre elementos de rede legados e novos terão que ser superados.
- Implantação: as várias abordagens na estratégia de implantação do SDN (de evolutiva a revolucionária) terão que ser analisadas a cada caso de rede.
- Segurança: ainda há questões importantes relativas a segurança para serem abordadas e resolvidas no contexto SDN.
- *Business Case*: os altos custos de migração de uma rede tradicional ao modelo SDN deverão ser economicamente viabilizados pela redução de investimentos e custos operacionais, além de receitas adicionais advindas de novos casos de uso.

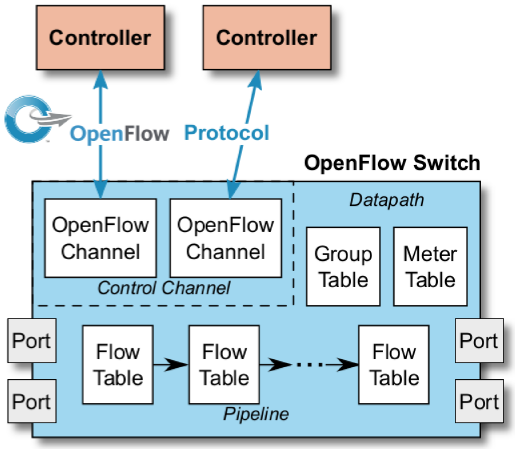
Porém, todas essas questões relativas à migração do paradigma atual de redes ao SDN são mais referentes a "quando?" do que "por que?", uma vez que esse modelo será essencial para aplicações baseadas em Internet do futuro (ZAIDI et al., 2018).

### 2.1.3 Protocolo OpenFlow

Para implementação de uma rede SDN, deve existir protocolo estruturado com qual o controlador da rede possa se comunicar com os elementos de rede presentes na camada de infraestrutura. Um desses protocolos foi proposto em 2008 e denominado OpenFlow, em um artigo na revista *ACM SIGCOMM Computer Communication Review* (MCKEOWN

et al., 2008). Atualmente, o protocolo OpenFlow está em sua versão 1.5.1 (março de 2015) e sua especificação pode ser encontrada em (ONF, 2015). Como já comentado, o protocolo OpenFlow permite que o controlador comunique-se com os elementos de rede na camada de infraestrutura. Para que isso aconteça, esses elementos (chamados de switches, independente da camada OSI em que atue, podendo ser inclusive roteadores) têm que ser compatíveis com o OpenFlow, tendo os seguintes componentes mostrados na Figura 3.

Figura 3 – Componentes de uma switch OpenFlow.




Fonte: (ONF, 2015)

Ainda segundo (ONF, 2015), uma switch lógica OpenFlow consiste de uma ou mais *flow tables* (tabela de fluxos) e uma *group table* (tabela de grupo), que realiza o encaminhamento de pacotes e um ou mais *OpenFlow channels* (canais OpenFlow) utilizados para comunicação entre a switch e o controlador através do protocolo OpenFlow. Através deste, o controlador pode adicionar, atualizar ou deletar as chamadas *flow entries*[1] nas *flow tables*, de modo reativo (em resposta a pacotes) ou proativo. Cada *flow table* contém uma série de *flow entries*, cada uma delas, por sua vez, consistindo de:

- *Match Fields*: também chamados de "*criteria*", são os parâmetros usados para se realizar uma correspondência com os pacotes, consistindo de portas de entrada, IPs de origem e destino e cabeçalhos de pacotes, por exemplo.
- *Priority*: representa a prioridade de precedência de uma *flow entry*. É levada em conta quando há uma correspondência a duas *flow entries* simultaneamente. Quando isso acontecer, a de maior prioridade é executada em detrimento à de menor prioridade.
- *Counters*: são atualizados a cada correspondência de pacote.

[1] As *flow entries* são também chamadas de *flow rules*, dependendo da referência. Em (ONF, 2015) usa-se *flow entry*, então essa forma será mantida nesta seção, mas o termo *flow rule* também será usado neste trabalho, com o mesmo significado.

- *Instructions*: conjunto de ações que ocorre quando há uma correspondência do pacote com os *match fields*.
- *Timeouts*: período máximo de tempo de vida total ou de ociosidade de uma *flow entry*, após o qual ela torna-se inválida.
- *Cookie*: Não utilizado no processamento dos pacotes, são dados "opacos" que podem ser usados pelo controlador para filtrar as *flow entries* afetadas por estatísticas, modificações ou solicitações de exclusão.
- *Flags*: alteram a maneira que as *flow entries* são gerenciadas.

O processamento dos pacotes é feito através de uma série de *flow tables*, podendo esses serem encaminhados, modificados ou processados por cada uma delas. Este processamento em *pipeline* permite que os pacotes sejam encaminhados a subsequentes tabelas para processamentos sucessivos e que informações, na forma de metadados, possam ser passadas de uma tabela a outra. O processamento em *pipeline* pára quando o conjunto de instruções associados a uma *flow entry* não especifica uma próxima tabela. Neste ponto, normalmente o pacote é modificado e encaminhado a uma porta (ONF, 2015).

Essa porta muitas vezes é uma porta física (na forma, por exemplo, de uma conexão ethernet), mas pode também ser uma porta lógica definida pela switch ou uma porta reservada pela especificação (por exemplo, uma porta do tipo *loopback*). As ações associadas às *flow entries* também pode direcionar os pacotes a um grupo, que especifica processamento adicional. Os grupos representam conjuntos de ações para *flooding* ou semânticas mais complexas de encaminhamento, como multi-caminhos ou agregação de links. A *group table* contém *group entries*, cada uma delas contem uma lista de ações com semântica específica dependendo do tipo do grupo, sendo aplicadas a pacotes direcionados a esse grupo (ONF, 2015).

Desta forma, as switches que suportam o protocolo OpenFlow tornam-se muito versáteis, como será visto na Subseção 2.1.4. Pode-se configurar ações a serem aplicadas sobre os pacotes baseadas em correspondências de camada física ("se um pacote vier da porta 2, encaminhe o mesmo à porta 5"), camada de enlace ("se o endereço MAC de destino do pacote for 00:0A:83:B1:C0:8E, encaminhe o mesmo à porta 5"), camada de rede ("se o endereço IP de destino do pacote for 192.168.0.100, mude sua marcação de DSCP e encaminhe o mesmo à porta 5") ou mesmo camada de transporte ("se o protocolo de transporte do pacote for TCP com porta 80, encaminhe o mesmo à porta 5"). Isso faz com que, como já comentado, as switches não sejam elementos de processamento exclusivo de uma camada específica, mas tenham a versatilidade de trabalhar em quaisquer camadas que sejam mais apropriadas para a funcionalidade da rede em questão.

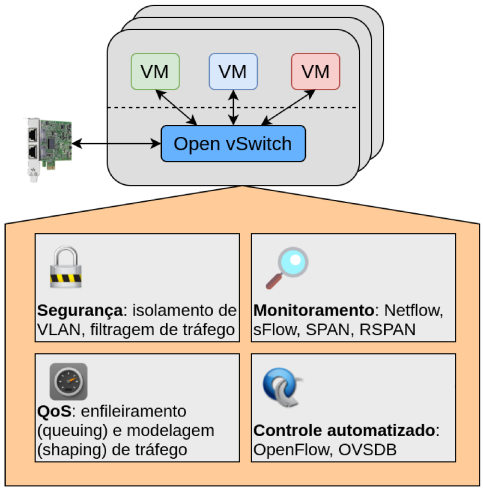
Existem outros protocolos que implementam a comunicação entre controladores e os elementos de rede, porém o OpenFlow será o utilizado neste trabalho.

### 2.1.4 Open vSwitch

A *Open vSwitch* (também chamada pelo acrônimo OvS) é uma switch virtual multi-camadas, licenciada sob a licença de código aberto Apache 2 e utilizada principalmente em ambientes virtualizados. Também suporta distribuição em diversos servidores físicos. É uma das mais populares implementações que suportam o OpenFlow. A maior parte de seu código foi escrito em C independente da plataforma, fazendo-o facilmente portável a outros ambientes (LINUX FOUNDATION, 2016a).

A Figura 4 ilustra conceitualmente a *Open vSwitch*, com algumas funcionalidades permitidas em sua implementação.

Figura 4 – Funcionalidades com implementação na Open vSwitch.




Fonte: adaptada de (LINUX FOUNDATION, 2016a)

Na distribuição atual, seus componentes principais são (LINUX FOUNDATION, 2016a):

- ovs-vswitchd: uma *daemon* que implementa a switch, junto com o módulo kernel Linux que a acompanha.
- ovsdb-server: um servidor de base de dados leve de onde a ovs-vswitchd busca sua configuração.
- ovs-dpctl: ferramenta de configuração do módulo kernel da switch.
- ovs-vsctl: um utilitário para consulta e atualização da configuração da ovs-vswitchd.
- ovs-appctl: um utilitário para enviar comandos aos *daemons* que estão rodando.
- ovs-ofctl: um utiliário para consulta e controle de controladores e switches OpenFlow.
- ovs-pki: um utilitário para se criar e gerenciar infraestrutura de chaves públicas em switches OpenFlow.

Nos experimentos realizados nesse projeto, vários elementos de rede foram implementados utilizando-se *Open vSwitch*.

### 2.1.5 Self-Organizing Networks (SON)

Como já mencionado, SON é o acrônimo de *Self-Organizing Networks*. A IBM propôs o conceito de "sistemas autonômicos" em (GANEK; CORBI, 2003), definindo quatro funcionalidades básicas para esse tipo de sistema, que foram denominadas propriedades *self-**: *self-configuration*, *self-healing*, *self-optimization* e *self-protection*. Futuramente essas propriedades foram incorporadas no conceito SON.

De uma maneira geral, o termo SON foi popularizado no contexto de automação do gerenciamento de redes móveis, tendo sido padronizado pelo *3rd Generation Partnership Project* (3GPP) como solução para a administração da crescente quantidade e complexidade de parâmetros de rede a partir da Release 8 (3GPP, 2018b) (3GPP, 2018a). O objetivo de sua introdução pode ser dividido em três pontos principais: 1) trazer inteligência e adaptabilidade autônoma para as redes móveis; 2) otimização de investimentos e custos operacionais e; 3) melhoria de desempenho de rede em termos de capacidade, cobertura, eficiência espectral e experiência final (MOYSEN; GIUPPONI, 2018).

As propriedades SON nas redes móveis permitem uma minimização da intervenção humana em determinados aspectos do gerenciamento da mesma. Porém, mesmo nesse contexto, possui uma abrangência limitada a *self-configuration*, *self-healing* e *self-optimization* e, ainda assim, com poucas funcionalidades padronizadas dentro dessas propriedades (MOYSEN; GIUPPONI, 2018). Além do mais, as funcionalidades SON nas redes móveis dizem respeito à parametrização de configurações de redes de acesso, ou *Radio Access Network* (RAN), e não se estendem aos elementos de *core* de rede (TELUS; HUAWEI, 2016).

Como será mencionado na Seção "Trabalhos Correlatos", não existem muitos trabalhos práticos fora do contexto RAN da rede móvel. Alguns modelos de arquitetura já foram propostos (NEVES et al., 2016), porém uma abordagem prática ainda foi pouco explorada. A propriedade *self-configuration* das redes ainda é um assunto com muito a ser desenvolvido.

### 2.1.6 A Funcionalidade de Self-Configuration

A propriedade *self-configuration* é uma das quatro principais funcionalidades de rede propostas inicialmente pela IBM no conceito de sistemas autonômicos (GANEK; CORBI, 2003) e sua importância nesse contexto é óbvia: sistemas autonômicos devem ser capazes de se auto-configurar, de maneira a simplificar a etapa de *set up* de rede.

Segundo (3GPP, 2018a) e (MOYSEN; GIUPPONI, 2018), a *self-configuration* é o processo de colocar um elemento de rede em serviço com a mínima intervenção humana.

Essa definição, embora aplicada originalmente a sistemas celulares, pode ser estendida a todos os elementos não pertencentes ao contexto de comunicações móveis.

De fato, a *self-configuration* pode ser vista sob duas diferentes abordagens: a do *bootstrapping* de rede, quando a mesma é "ligada" pela primeira vez e todos seus elementos – denominados *Network Element*s (NEs) em inglês – comunicam-se entre si e com o controlador, de modo que sua topologia seja corretamente mapeada e percebida pelo mesmo; e da *self-configuration* quando um novo NE é inserido ou retirado da rede, o que vai exigir uma reconfiguração da mesma para adequação à esse novo elemento, sendo esse processo chamado *plug-and-play* (GONÇALVES et al., 2020).

Essa última abordagem pode ser subdividida em algumas sub-funcionalidades, porém o trabalho aqui apresentado diz respeito à *self-configuration* da rede de forma especializada em resposta à inclusão de um novo elemento de borda, ou seja, a ativação de um novo NE que será a interface final entre o dispositivo do cliente e a rede em si. É importante ressaltar que, no contexto deste trabalho, o elemento de borda não restringe-se a equipamentos celulares, como eNodeBs, mas pode incluir diversos tipos de NEs, incluindo de redes fixas.

### 2.1.7 Self-Organizing Networks Architecture (SONAr)

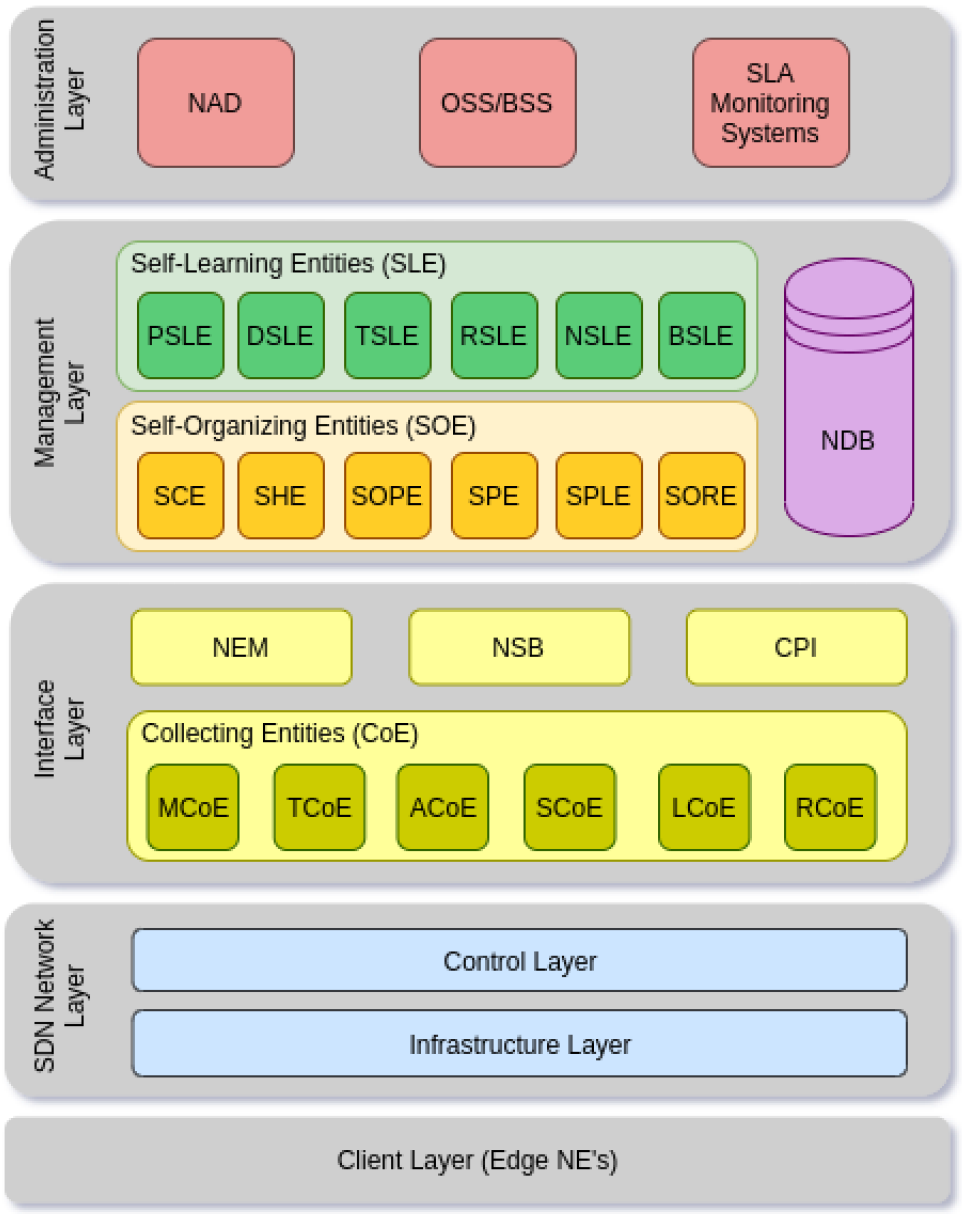
Um sistema SON deve rodar sobre uma arquitetura bem definida. Embora existam alguns *frameworks* que tentam padronizar e integrar funcionalidades SDN e NFV, como proposto em (NEVES et al., 2016), a implementação prática de um sistema SON ainda possui poucas alternativas desenvolvidas. Uma arquitetura e implementação prática foram propostas em (GONÇALVES et al., 2020), e denominada *Self-Organizing Networks Architecture* (SONAr). Esse sistema está em desenvolvimento por uma equipe de pesquisadores da Universidade Federal de Uberlândia, mas já possui resultados práticos publicados (GONÇALVES et al., 2020) (SOUZA NETO. et al., 2020) (OLIVEIRA et al., 2020).

A arquitetura SONAr é mostrada na Figura 5. Ela é composta de cinco camadas que possuem funções e funcionalidades específicas, que serão descritas a seguir. As descrições dos componentes são baseadas no trabalho de (GONÇALVES et al., 2020), exceto quando mencionada outra referência. Para manutenção de um padrão internacional e coerência com os termos usados nas referências, serão mantidas as grafias em inglês.

#### 2.1.7.1 Client Layer

Nesta camada situam-se os elementos que serão incluídos na rede e que demandarão uma configuração de forma automática de modo a permitir o pleno funcionamento na rede, processo o qual é o objeto de estudo deste trabalho. Trata-se de elementos de borda (*edge NEs*), que podem ser de diversos tipos, sendo que cada um deles necessita a configuração

Figura 5 – Diagrama simplificado em camadas da arquitetura SONAr.




Fonte: adaptada de (GONÇALVES et al., 2020)

da camada superior (*SDN Network Layer* e, em especial, a *Infrastructure Layer*) de uma forma específica.

#### 2.1.7.2 SDN Network Layer

Essa camada pode ser logicamente dividida em duas: a inferior representa uma sub-camada de infraestrutura, composta dos elementos de rede em si, como as switches, os roteadores, bridges, etc, sendo eles elementos físicos ou virtuais. A arquitetura SONAr permite que estes elementos sejam ou não compatíveis com os protocolos e funcionalidades SDN, embora a incompatibilidade com o protocolo OpenFlow naturalmente traga grandes

restrições nas funcionalidades do sistema para tais elementos.

A sub-camada superior, por sua vez, compreende os controladores SDN (SDNCs) que, centralizando o plano de controle, detém a "inteligência" primária da rede. Em uma mesma rede pode haver um ou mais controladores, que, no caso do SONAr, foram implementados na plataforma ONOS. Os SDNCs comunicam-se com os NEs através da SBI utilizando protocolo OpenFlow. Conforme já mencionado, podem ser implementados vários SDNC para uma mesma rede, no modelo de cluster (CAMPANELLA; DANDOUSH; MANASSAKIS, 2017).

#### 2.1.7.3 Interface Layer

Na *Interface Layer*, são implementados alguns componentes que fazem a interface entre a *Management Layer* e a *SDN Network Layer*. Essa camada compreende as *Collecting Entities* (CoEs) e outros componentes que serão apresentados a seguir.

As CoEs são compostas de diversos módulos que, juntos, realizam as coletas de métricas, topologia, alarmes e demais dados que são gerados durante a operação da rede. Seus componentes são a *Metrics Collecting Entity* (MCoE), a *Topology Collecting Entity* (TCoE), a *Alarms Collecting Entitiess* (ACoEs), a *Samples Collecting Entity* (SCoE), a *Logs Collecting Entity* (LCoE) e a *Results Collecting Entity* (RCoE). Todos esses componentes são implementados como microsserviços e têm, como seus próprios nomes indicam, funções de coletas específicas.

Os demais componentes da *Interface Layer* são:

- *Network Element Manager* (NEM): o SONAr é uma arquitetura completamente orientada a eventos, logo, o NEM desempenha um papel importante na conexão de seus componentes. O NEM é um provedor de publicação e subscrição que gerencia eventos, que, por sua vez, são categorizados em tópicos. Um componente – tipicamente uma *Self-Organizing Entity* (SOE) ou uma *Self-Learning Entity* (SLE) – pode assinar um tópico específico e receber notificações relativas a esse tópico. O processamento dos eventos é feito de forma assíncrona, através de uma solução de fila de mensagens, que é gerenciada pelo NEM. Na implementação atual do SONAr, o NEM foi implementado com o *message broker open source* RabbitMQ.

- *Network Service Broker* (NSB): o NSB é usado para configuração de serviços baseados em intenções expressas em linguagem natural ou formal pelos clientes ou administradores do sistema. Uma vez que a intenção seja expressa, o NSB interpreta a mesma e solicita a uma SOE que realize uma ação correta na rede.

- *Control Primitives Interceptor* (CPI): esse componente é um dos mais importantes na arquitetura SONAr. Ele situa-se na SBI, atuando como um proxy entre a rede e o controlador e fazendo a captura e análise de todas as primitivas de controle

que passam por essa interface. Caso seja detectada semântica de gerenciamento (i.e. primitivas relativas a métricas ou notificações), o CPI a envia tanto aos componentes SONAr quanto ao SDNC, caso contrário, a mesma é enviada apenas ao SDNC. Quando enviada ao SONAr – através do NEM –, o CPI realiza um *parse* da primitiva, transformando-a em um evento. Isso faz com que uma grande base de informações relativas ao gerenciamento da rede seja armazenada e reportada aos diversos componentes do SONAr, ou analisadas em conjunto pelas SLEs de modo a poder-se inferir sobre a rede.

#### 2.1.7.4 Management Layer

A camada de gerenciamento (*Management Layer*) pode ser dividida em duas subcamadas: a que integra as SOEs e a que reúne as SLEs.

As SOEs são responsáveis por realizar os algoritmos básicos relacionados aos fundamentos da computação autonômica. Como já mencionado, o NEM recebe eventos de todos os componentes da arquitetura. As SOEs assinam tópicos específicos do NEM e recebem os eventos relativos às propriedades *self-**, podendo tomar decisões de configuração, cura, otimização ou proteção da rede, atuando sobre a mesma.

Os componentes da sub-camada SOE são:

- *Self-Configuration Entity* (SCE): A SCE atua, basicamente, em dois momentos: no processo de *bootstrapping* da rede, quando os *flows* da rede inteira é configurada, e quando uma nova NE é plugada na mesma, no processo chamado *plug-and-play*. No caso de elementos convencionais, como switches SDN, esse último processo é mais simples, pois segue um plano já bem conhecido e determinado. No caso das NEs de borda, como, por exemplo, uma estação rádio-base de um sistema móvel celular, um processo próprio e específico é necessário. Neste caso em particular, há necessidade de implementar-se diferentes VLANs com diferentes características de QoS para cada tipo de pacote (no caso, de gerência, de sincronismo, de controle e de dados). Portanto, o caso *plug-and-play* pode ainda ser dividido em *self-configuration* de casos gerais e casos específicos aplicáveis a NEs de borda especializados (OLIVEIRA et al., 2020), o que será o objeto de investigação deste trabalho.

- *Self-Healing Entity* (SHE): a SHE implementa algoritmos que permitem prever ou detectar falhas em todos os elementos mostrados na Figura 5, provendo soluções para recuperação dos elementos que apresentarem essas falhas ou implementar alternativas para que a rede e seus componentes continuem funcionando mesmo com elas. As falhas nos elementos da subcamada de infraestrutura (*Infrastructure Layer*) já são normalmente tratadas pelos controladores SDN, porém, a SHE pode prever falhas que ainda não ocorreram nessa camada. Além disso, deve também tratar falhas na própria subcamada de controle (*Control Layer*), onde estão situados os

SDNCs e nos demais componentes do SONAr situados nas camadas superiores também são tratadas pela SHE, em especial a *management layer* (SOUZA NETO. et al., 2020). Por último, a SHE em si pode falhar e, nesse caso, deve autorecuperar-se o quanto antes possível.

- *Self-Optimization Entity* (SOPE): O objetivo da SOPE é a de intervir na rede de modo a maximizar seu desempenho, com base em estatísticas de rede coletadas e analisadas. A SOPE "assina" tópicos específicos do NEM e, assim, recebe informações de eventos e métricas de rede. Com base nessas informações, pode sugerir ou implementar diretamente melhorias através de rearranjos de fluxos ou realocação de recursos – o que é bastante natural em redes SDN/NFV – de modo a ter-se uma configuração mais inteligente da rede. Além disso, a SOPE também gerencia os processos de *network slicing* (fatiamento de rede), sendo essa uma questão crucial para garantia de QoS para cada serviço utilizado na rede. A SOPE trabalha em grande sinergia com o componente *Diagnostics Self-Learning Entity* (DSLE), que será abordado na subcamada SLE. A diferença entre eles é que, enquanto o DSLE (como os demais componentes das SLEs) utiliza técnicas de Inteligência Artificial (IA) para identificar potenciais melhorias na configuração da rede, a SOPE recebe relatórios da DSLE como input para ajustar os fluxos entre as rotas, criação de novos elementos em estrutura NFV e outras ações de otimização de redes.

- *Self-Protection Entity* (SPE): A SPE tem como função principal a de garantir a proteção de todas as camadas apresentadas na Figura 5 contra ataques internos e externos. Isso pode ser feito em três etapas: detecção do ataque, isolamento do componente e recuperação do mesmo. A detecção do ataque, a exemplo da SOPE, também é realizada pela DSLE que, baseada em análises realizadas com ajuda de algoritmos de IA, identifica um padrão de ataque, notificando a SPE. Então, a SPE primeiro isola o elemento atacado, para previnir que o ataque alcance outras instâncias na rede. Em seguida, recupera o elemento atacado, "limpando" os resquícios do ataque e o reinsere na estrutura. Há várias técnicas e mecanismos bem descritos na literatura sobre como evitar e se recuperar de ataques na camada de infraestrutura, de forma que o maior desafio da SPE é a de proteger as demais camadas da rede, incluindo todos os componentes SONAr. É importante sobretudo a proteção dos SDNC, uma vez que, a posse do acesso a esses elementos dá ao atacante o controle de toda infraestrutura.

- *Self-Planning Entity* (SPLE): Trabalhando em conjunto com todas as SOEs, a SPLE organiza, controla e agenda todas as ações sobre a rede advindas das demais entidades dessa camada, garantindo inclusive que ações repetidas que possam vir de entidades diferentes sejam descartadas. Essa funcionalidade é importante no

contexto das SOEs, uma vez que, por serem entidades separadas e independentes, pode haver ações de duas ou mais que causem conflitos ou inconsistências na rede.

- *Self-Orchestration Entity* (SORE): Por fim, a SORE tem o papel de receber todas as ações das demais entidades – inclusive a SPLE – e organizar as mesmas em filas, priorizando essas ações. A determinação das prioridades será feita pelo administrador ou operador da rede, através da *Network Administration Dashboard* (NAD), da camada de administração.

Acima da subcamada das SOEs, fica a subcamada das SLEs. O objetivo desses componentes é a de prover mecanismos baseados em IA para suplantar a ausência de intervenção humana. Para isso, usa base de conhecimentos adquiridas pela própria rede através de uma análise baseada em ação e consequência usando como base algoritmos de aprendizado de máquina, bases manualmente inseridas por operadores ou mesmo importadas de terceiros que já tenham desenvolvido estratégias próprias, sendo essas alternativas não-excludentes. Em resumo, essa subcamada é constituída das seguintes entidades: *Prediction Self-Learning Entity* (PSLE), *Tuning Self-Learning Entity* (TSLE), *Diagnostics Self-Learning Entity* (DSLE), *Rating Self-Learning Entity* (RSLE), *Natural Language Processing Self-Learning Entity* (NSLE) e *Behavior Pattern Self-Learning Entity* (BSLE). Devido ao fato de estarem em uma camada acima do escopo desse trabalho, não tendo influência sobre o mesmo, essas SOE não serão discutidas nesse trabalho, porém maiores detalhes sobre as mesmas podem ser encontrados em (GONÇALVES et al., 2020).

Finalmente, a *Network Database* (NDB) implementa as bases de dados que compõem toda informação gerada e armazenada no SONAr. Para citar algumas (mas não restrito às mesmas), as seguintes informações são gravadas na NDB:

- Métricas
- Dados de topologia
- Alarmes
- Amostras e logs coletados pelo CoE
- Dados e regras de administração do NAD (*dashboard*)
- Primitivas de gerenciamento coletadas pelo CPI
- Eventos enviados ao NEM
- Eventos criados por quaisquer entidades SOE
- Predições e informações de análise gerados por quaisquer entidades SLE

É importante ressaltar que as SLEs utilizam os dados de eventos enviados ao NEM para compôr sua base para análise em aprendizado de máquina. Por isso, esses eventos, após serem usados pelas SOEs, não são descartados, mas armazenados na NDB, que também gravam os resultados gerados pelas SLEs.

Uma vez que os dados armazenados na NDB são vitais para o funcionamento de toda rede, ela foi projetada em arquitetura distribuída para garantir uma alta disponibilidade e escalabilidade. Para tal, na atual configuração, foi implementada utilizando-se a base de dado NoSQL Cassandra.

#### 2.1.7.5 Administration Layer

A camada de administração, ou *Administration Layer*, compõe o último nível da arquitetura SONAr. Além dos três módulos indicados na Figura 5, pode-se ter outras aplicações de alto nível desenvolvidas para que todo sistema adeque-se a contextos específicos. De um modo geral, essa camada deve necessariamente implementar o já citado NAD, onde os administradores da rede podem ter acesso aos dados do SONAr e visualizar topologias, métricas, alarmes, logs e eventos, além de gerenciar regras e intervir manualmente na rede. Porém alguns outros componentes são importantes para compôr uma solução completa de arquitetura.

Os OSS e BSS são bem conhecidos no contexto de telecomunicações, tratando-se dos sistemas que suportam a operação da rede, usado por principalmente por operadores e *designers* para supervisionar, operar e planejar a mesma (OSS), além de atividades relativas a clientes, como bilhetagem, relacionamento, automação de *call centers*, dentre outros (BSS).

Os Sistemas de Monitoramento de *Service Level Agreement* (SLA) (ou Acordo de Nível de Serviço) são automações que permitem o monitoramento, geração de alarme ou intervenção na rede em respeito a contratos de nível de serviço com clientes ou serviços em específico (os chamados SLAs). Um serviço com SLA que exija requerimentos mais restritos normalmente deve ser provisionado com mais recursos, seja largura de banda garantida, menor *downtime*, menor latência, etc. O SONAr pode realizar esse gerenciamento, predizendo quando os parâmetros do SLA deixarão de ser cumpridos ou mesmo realizando alterações automáticas na rede de modo a garantir esse cumprimento.

Outros módulos podem ser incluídos para que, como comentado, a arquitetura SONAr adeque-se a necessidades específicas de cada negócio.

#### 2.1.7.6 Status Quo e Futuros Desenvolvimentos

O SONAr é um esforço conjunto de uma equipe de pesquisadores e um trabalho em constante evolução. Toda sua especificação encontra-se completa, embora apenas uma breve descrição das camadas e componentes tenha sido apresentada neste trabalho. Muitos componentes já foram implementados, sobretudo os da *Interface Layer*, como boa

parte das CoEs, NEM, NSB e CPI, e da *Management Layer*, como algumas SOEs e o NDB. Em específico, a parte de *bootstrapping* e o *plug-and-play* de elementos genéricos da SCE já estão prontos e validados, como demonstrado em (GONÇALVES et al., 2020), enquanto que o *plug-and-play* para elementos de borda (auto-configuração especializada) é objeto deste trabalho. A SHE é apresentada em (SOUZA NETO. et al., 2020) e está em avançado estágio de desenvolvimento. Algumas SLEs estão implementadas de maneira simplificada (por exemplo, utilizando-se técnicas simples de aprendizado de máquina, como regressões lineares), enquanto que outras ainda se apresentam como oportunidades de investigações futuras. Na *Administration Layer*, o componente NAD também está parcialmente desenvolvido, embora algumas melhorias ainda devam ser implementadas.

De uma forma geral, o esforço para implementação completa de uma arquitetura no nível proposto pelo SONAr é bastante considerável, e é um trabalho em constante desenvolvimento, uma vez que melhorias nos componentes são sempre possíveis e inovações tecnológicas constantemente exigirão evolução na arquitetura em si.

### 2.1.8 Redes Móveis

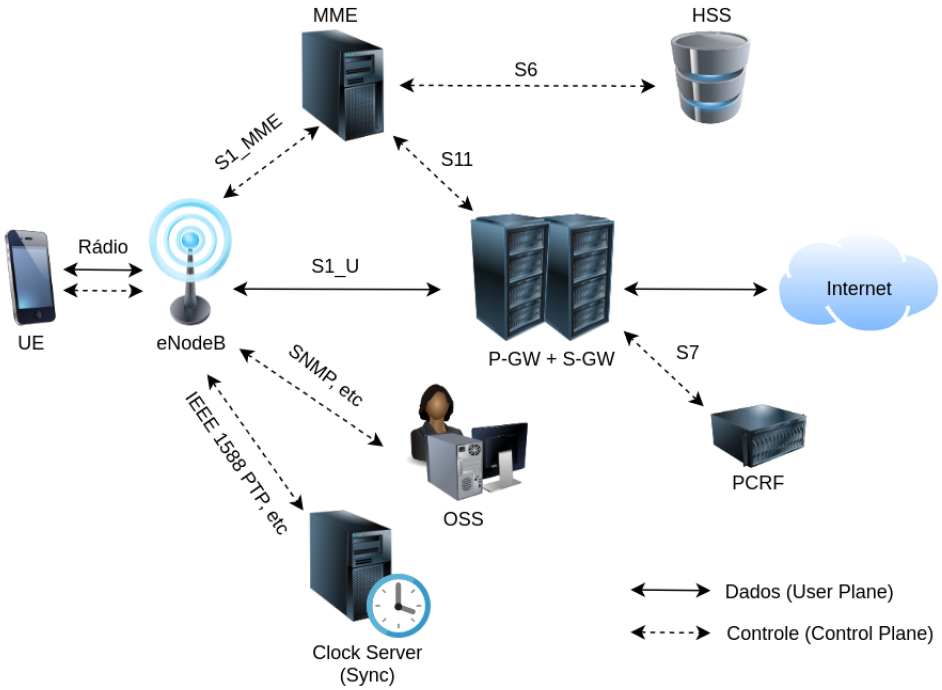
Como estudo de caso de auto-configuração especializada neste trabalho, serão realizadas simulações da implantação de uma eNodeB (estação rádio-base 4G) em uma rede móvel. Desta forma, para compreensão deste contexto, uma breve discussão sobre a arquitetura básica de uma rede celular será aqui apresentada.

A Figura 6 representa a topologia básica de uma rede móvel celular de quarta geração, denominada *Long Term Evolution* (LTE) ou *Long Term Evolution Advanced* (LTE-A), no contexto 3GPP. Ela mostra os principais elementos de rede envolvidos no processo básico de acesso a dados desde o usuário final até a Internet em si. Para efeito de simplificação, foi dado um foco prioritário nas conexões da eNodeB, sendo que algumas interfaces foram omitidas (como, por exemplo, entre o *Clock* (ou *Sync*) *Server* com os demais elementos da rede).

Os elementos básicos da topologia são os seguintes (HOLMA; TOSKALA, 2012):

- *User Equipment* (UE): representa o equipamento do usuário final, tipicamente um *smartphone*, *tablet* ou computador com modem 4G.
- eNodeB: é a Estação Rádio-Base (ERB), que irradia o sinal de rádio-frequência e provê a interface-rádio para acesso do UE.
- *Mobility Management Entity* (MME): esse elemento é responsável pelo rastreamento das UEs e instrui os *gateways* sobre onde os dados devem ser enviados.
- *Packet Data Network Gateway* (P-GW) e *Serving Gateway* (S-GW): os dois elementos trabalham em conjunto, servindo de gateway para a internet. Enquanto o

Figura 6 – Diagrama Simplificado da Arquitetura de um Sistema Móvel Celular LTE.




Fonte: adaptada de (HOLMA; TOSKALA, 2012)

P-GW converte os protocolos 3GPP, adequando as conexões aos protocolos genéricos de Internet, o S-GW faz efetivamente a interface com a rede em si. O conjunto P-GW e S-GW também é comumente denominado *System Architecture Evolution Gateway* (SAE-GW).

- OSS: implementa a gerência da rede, comunicando-se com a eNodeB via protocolo SNMP ou quaisquer outros protocolos de gerência (inclusive os proprietários, dependendo do fabricante dos elementos). É uma entidade diferente da OSS apresentada anteriormente, na Subseção *Self-Organizing Networks Architecture (SONAr)*, operando aqui no contexto da rede móvel.

- *Clock* (ou *Sync*) *Server*: fornece o sincronismo de rede para todos os elementos que compõem a rede, de modo que não haja erros por "escorregamento" da sinalização no tempo.

- *Home Subscriber Server* (HSS): é a base de dados que guarda as informações sobre o assinante (UE) da operadora da rede, incluindo funcionalidades habilitadas, quantidade de conexões simultâneas possíveis, etc.

- *Policy and Charging Resource Function* (PCRF): aplica as políticas de rede ao usuário final, como a taxa máxima de dados permitida, quantidade de dados disponível em sua franquia, tipo da cobrança, etc.

As interfaces entre os elementos são mostradas como setas inteiras ou pontilhadas. As setas inteiras representam as interfaces onde passam os dados do usuário, ou *User Plane* (UP), enquanto que as pontilhadas representam as interfaces puramente de sinalização, ou *Control Plane* (CP).

O foco do presente trabalho é a auto-configuração especializada de um elemento de borda (neste exemplo, uma eNodeB) na rede. Percebe-se, na figura, quatro elementos de rede que se conectam diretamente a esta eNodeB: o MME, o conjunto P-GW e S-GW, a plataforma de gerência da rede (OSS) e o *clock server*, para efeito de sincronismo. Essa última conexão é especialmente sensível, uma vez que requerimentos do LTE-A são de $\pm$1,5 $\mu$s a $\pm$5 $\mu$s, podendo, em alguns casos, chegar a $\pm$0,5 $\mu$s (SYMMETRICOM, 2013). Cada uma destas conexões exige uma configuração específica, em especial, uma VLAN própria, associada a uma marcação particular de QoS na camada de enlace.

A eNodeB é fisicamente ligada a todos estes elementos através do *backhaul*, normalmente um enlace DWDM ou de rádio-frequência ponto-a-ponto. Esse *backhaul* é geralmente integrado por algumas redes com topologias compostas de anéis (metro-ethernet camada 2 ou MPLS) no acesso e redes de transporte (camada 3) nos *backbones*, sendo que em cada uma dela envolve vários NEs. Uma topologia típica das redes que compõem o *backhaul* é mostrada na Figura 7.

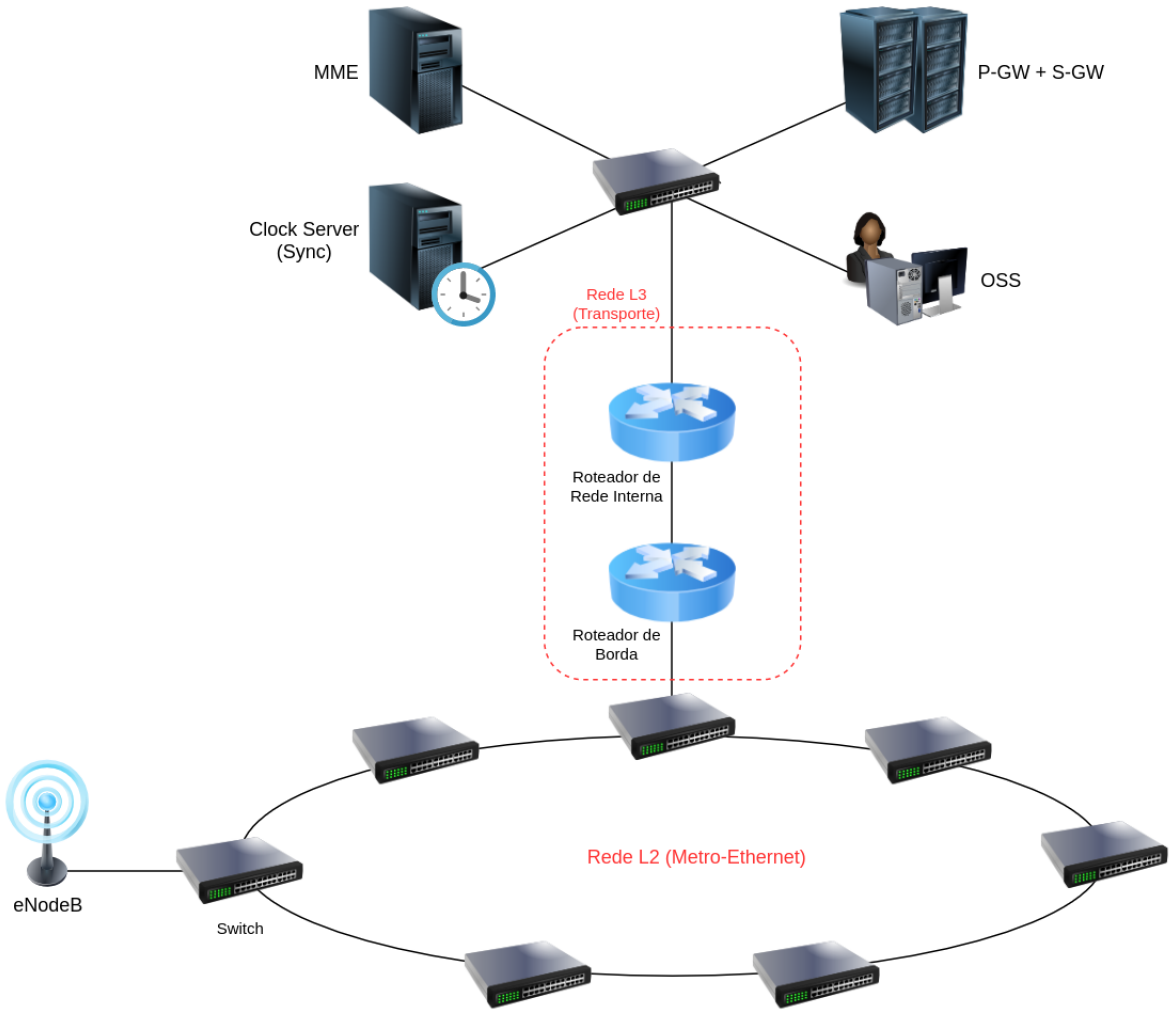
É importante reparar que, na inserção de um elemento de borda no sistema, todos os switches envolvidos no anel da rede de acesso devem ser configurados de modo a receber esse novo elemento. No caso de uma eNodeB, essa configuração envolve a criação de rotas para quatro VLANs (uma para cada elemento destino da rede) e atribuição de QoS para cada uma delas, através de marcações nos pacotes. Nos dias atuais, essa configuração é realizada manualmente, ou, no máximo, com uma semi-automatização usando-se *scripts*, mas não de forma automática.

Com o advento do 5G e uma tendência de massificação da quantidade de gNodeBs – uma vez que há a possibilidade de *femtocells* serem instaladas diretamente na cada do cliente em substituição aos pontos de acesso Wi-Fi no médio prazo – uma solução de auto-configuração especializada será não apenas desejável, mas totalmente necessária nesse contexto.

## 2.2 Trabalhos Correlatos

Como o tema é de grande relevância no contexto da indústria, alguns esforços têm sido desenvolvidos na abordagem do problema da funcionalidade de *self-configuration*. A seguir será apresentado um compilado de trabalhos já produzidos sobre SON e auto-configuração de redes.

Figura 7 – Diagrama Simplificado da Arquitetura Típica de um Backhaul.




Fonte: elaborada pelo autor

### 2.2.1 3GPP SON

Até agora, percebe-se que o SON tem uma maior evolução no contexto de redes móveis, mais especificamente nas redes de acesso de rádio, onde está relaticamente bem desenvolvido. (MOYSEN; GIUPPONI, 2018) apresenta um compilado de conceitos e taxonomias do SON dentro do contexto das especificações do 3GPP. A especificação do SON iniciou-se na Release 8 e vem evoluindo até a Release 16. As funcionalidades previstas são *self-healing*, *self-optimization* e *self-configuration*.

(HAKIRI; BERTHOU, 2015) afirmam que o suporte a um provisionamento automático de QoS e a operação e gerenciamento – *Operation, Administration and Management* (OAM) – cognitiva de redes serão aspectos importantes em uma rede 5G e essas funcionalidades só serão possíveis através da utilização dos paradigmas SDN e NFV, porém não apresenta respostas de como isso será feito na prática.

O início das especificações da automação de rede para 5G está se dando de forma

discreta na Release 16, na proposta dos facilitadores para automação de redes, com o Capítulo "*Enablers for Network Automation Architecture for 5G*" (3GPP, 2019b), porém ainda não haverá grandes propostas práticas para implementações de SON fora do escopo de RAN. O congelamento (*freezing*) da Release 16 aconteceu em julho de 2020 e o da Release 17 está previsto para setembro de 2021, segundo cronograma do 3GPP (3GPP, 2020).

Ainda segundo (MOYSEN; GIUPPONI, 2018), a funcionalidade de *self-configuration* tenta inserir um novo elemento na rede móvel (por exemplo, uma eNodeB) com um mínimo de intervenção humana. Porém esse processo restringe-se ao contexto dos protocolos e parâmetros das redes móveis, como a funcionalidade de *Automatic Neighbour Relation* (ANR) e atribuição automática de *Physical Cell Identity* (PCI). A configuração dos parâmetros da rede que envolve o *backhaul* normalmente é feita manualmente ou, no máximo, com a utilização de scripts em um processo semi-automatizado.

No geral, existem poucos estudos abrangentes sobre SON quando aplicado a redes de transporte e acesso. De fato, os raros projetos que endereçaram a auto-organização de redes na prática (com exceção da já mencionada abordagem em redes móveis) têm um foco de otimização na camada *Medium Access Control* (MAC) para LTE e Wi-Fi e gerenciamento de eficiência de energia e conectividade em áreas densas (NEVES et al., 2016).

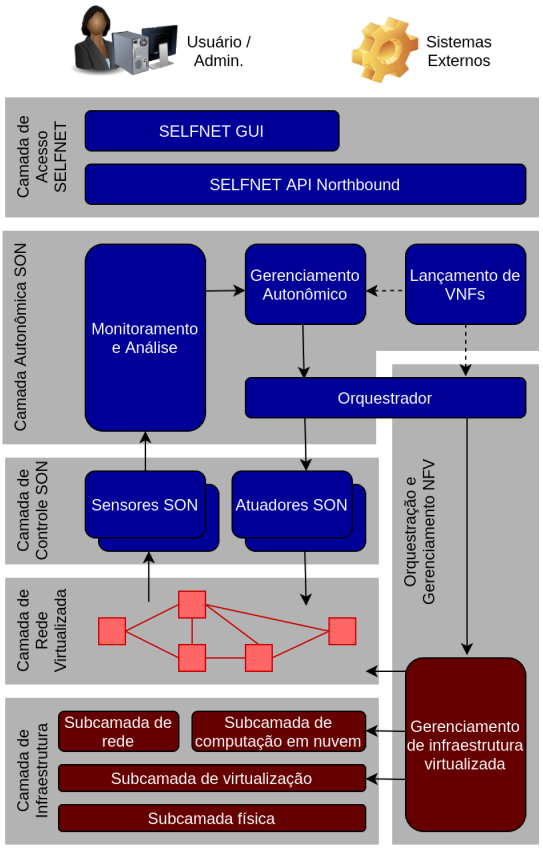
### 2.2.2 SELFNET

Uma abordagem no sentido de estabelecer-se um *framework* e uma arquitetura de referência para SON também no contexto SDN e NFV foi feita em (NEVES et al., 2016), com o projeto SELFNET. O SELFNET propõe funções de alto nível de SON, com foco em *self-healing*, *self-protection* e *self-optimization* (com bastante ênfase no último), sendo que o *self-configuration* não é tratado. Como os padrões NFV e SDN vêm de entidades de padronizações diferentes, é feita uma abordagem mista com os dois cenários, envolvendo também elementos físicos de rede não-virtualizados e não-SDN, os chamados PNFs. O modelo adotado pelo projeto SELFNET é mostrado na Figura 8.

Enquanto a Camada de Controle SON faz a coleta de dados de sensores e atua sobre a rede, a Camada Autonômica SON provê a inteligência da rede através de componentes baseados em IA. A Camada de Orquestração e Gerenciamento NFV corresponde às mesmas funcionalidades previstas para a camada MANO no modelo ETSI e, finalmente, a Camada de Acesso SELFNET provê a interface com os operadores, administradores e as APIs para sistemas externos.

O artigo em si não apresenta implementações práticas ou resultados, porém outras publicações posteriores já mostram resultados medidos, especialmente tratando de *self-optimization* (JIANG; STRUFE; SCHOTTEN, 2017).

Figura 8 – Arquitetura de Referência do Modelo SELFNET.




Fonte: adaptada de (NEVES et al., 2016)

### 2.2.3 *Superfluidity*

Outro projeto do programa *Horizon 2020*, o *Superfluidity* tem como objetivo apresentar uma arquitetura de rede flexível e adaptativa, visando principalmente aspectos importantes para o 5G, como *Cloud Radio Access Network* (CRAN), *Virtual Core* (vCore) e MEC, em um contexto de redes NFV e SDN (BIANCHI et al., 2016). Baseia-se em componentes no conceito *Reusable Function Block*s (RFBs) – entidades lógicas que realizam um conjunto de funcionalidades, com portas de entrada e saída de dados, de maneira similar a uma VNF, rodando em uma VM ou um *container*. As RFBs são descritas em uma linguagem denominada *Reusable Function Blocks Description and Composition Language* (RDCL) e o ambiente em que são executadas é chamado *Reusable Function Blocks Execution Environment* (REE). Com isso, a intenção é a de melhorar a portabilidade através da habilidade de se descrever funções de rede complexas como uma combinação de RFBs, e o objetivo geral do projeto é o de obter-se uma descrição formal e agnóstica de como as RFBs devem ser chamadas, em qual ordem, com quais dados de entrada, etc.

O trabalho é bem ambicioso, porém não apresenta nenhuma proposta prática de implementação que não o *framework* em si. Também não endereça aspectos específicos das propriedades *self-**, o que não significa que não possam ser implementadas no *framework*,

porém não há menção dessas aplicações em específico no trabalho.

### 2.2.4 CogNet

O CogNet também é um projeto dentro do programa europeu *Horizon 2020* e propõe uma arquitetura de uma rede auto-gerenciada baseada em NFV, capaz de atingir níveis de alto QoS, além de eficiência energética e operacional através de técnicas de aprendizado de máquina (XU et al., 2016). Para tal, utiliza componentes como o *CogNet Data Collector*, que coleta dados da rede NFV, o *CogNet Smart Engine*, que faz o processamento em tempo quase real dessas informações e o *CogNet Policy Manager*, que atua sobre os componentes da rede NFV, buscando um melhor desempenho em termos da QoS percebida pelo usuário final e eficiência de uso da rede.

O trabalho não faz menção a SDN ou redes legadas, atuando apenas dentro do *framework* NFV. Também não apresenta resultados práticos, mas apenas a proposta da arquitetura em uma descrição introdutória.

### 2.2.5 Auto-Configuration of SDN Switches in SDN/Non-SDN Hybrid Networks

O trabalho propõe um mecanismo de *self-configuration* para um novo switch SDN inserido em redes híbridas SDN e não-SDN (KATIYAR et al., 2015). As etapas propostas para tal são: detecção do novo elemento SDN, configuração de parâmetros da rede híbrida nesse novo elemento e garantia do serviço ao longo dos segmentos da rede híbrida. O sistema consiste de quatro componentes, a saber:

- *SDN AutoConf Server*
- *SDN AutoConf Client*
- *Intermediate Switch Configurator*
- *New Switch Locator*

O sistema foi implementado na prática e testado em um modelo simples de rede, porém não foram apresentadas métricas de tempo de resposta ou convergência. Também não está inserido em um contexto maior de SON, sendo que nem as demais propriedades *self-**, nem a manutenção de uma base de dados de parâmetros de rede não estão previstas no projeto.

### 2.2.6 A Network Management Framework for SDN

(ABDALLAH et al., 2018) faz uma investigação sobre gerenciamento de rede em SDN, com foco no modelo *Fault, Configuration, Accounting, Performance, Security* (FCAPS),

categorizando as funções de rede como uma orientação para se implementar o gerenciamento em SDN. O artigo coloca que o gerenciamento da rede não deve ficar apenas a cargo do controlador (SDNC), uma vez que sua função primordial é o de controlar os serviços da rede, então propõe a separação das funcionalidades do FCAPS entre as funções SDNC e SDN Manager, conforme mostrado na Tabela 1.

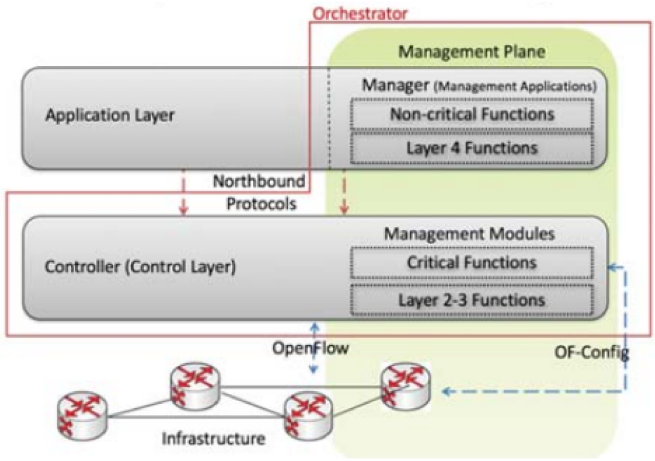
Tabela 1 – Classificação das funções de rede versus elemento responsável.

| **Nível** | **SDN Manager** | **SDN Controller** |
|---|---|---|
| Fault | | X |
| Configuration | X | X |
| Accounting | X | |
| Performance | X | X |
| Security | | X |

Fonte: adaptada de (ABDALLAH et al., 2018)

Algumas funcionalidades de *configuration* e *performance* podem ser tratadas pelo SDN *Manager*, mas funções específicas que exigem agilidade de ação devem ser tratadas no SDNC. O modelo em camadas ficaria conforme mostrado na Figura 9.

Figura 9 – Framework proposto para gerenciamento baseado em categorização de funções.




Fonte: (ABDALLAH et al., 2018)

O artigo faz então um modelamento matemático para simular a performance do gerenciamento FCAPS de uma rede com três cenários: *framework* proposto na Figura 9, todas funções no SDNC e todas as funções no SDN *Manager*. A conclusão final é que a performance no *framework* proposto é melhor do que qualquer outro cenário.

Além da divisão em camadas e da simulação matemática, o trabalho não propõe uma implementação prática nem mostra resultados comprovados em um ambiente real ou simulado, ficando restrito à simulação teórica.

### 2.2.7 Policy-based QoS Management Framework for SDN (PBNM)

O trabalho de (AL-JAWAD et al., 2018) propõe um *framework* de gerenciamento de redes SDN tendo como base o monitoramento de QoS e *Quality of Experience* (QoE) do usuário final. Se apoia na proposta de *Policy-based Network Management* (PBNM), que automatiza o processo de reconfiguração de rede através de um conjunto de regras de alto nível. O *framework* consiste das seguintes entidades:

- *Policy Repository*: armazena as regras que refletem os requerimentos dos serviços
- *Topology Tracker*: mapeia a rede física em um estrutura gráfica
- *Admission Control*: aceita ou rejeita conexões de rede baseado na disponibilidade de recursos
- *QoS Metrics Monitor*: mede as métricas de QoS sondando diretamente as switches
- *Violation Detector*: é a máquina de validação que toma as medidas para convergência da rede nos requerimentos acordados
- *Active Flows Tracker*: acompanha as rotas e estima a taxa de dados por fluxo ativo
- *Route Manager*: computa o caminho menos carregado e de menor esforço entre o cliente e a aplicação
- *Rate Limiting Manager*: configura parâmetros de limite de taxa ao longo das rotas

Estes componentes rodam em uma camada acima da camada de controle SDN e comunicam-se com os controladores via NBI. Os fluxos que apresentam violações na QoS acordada são identificados através de técnicas de IA, como redes neurais.

Para testar-se a validade do modelo, um *setup* experimental foi montado, utilizando-se um controlador OpenFlow *Floodlight*, o simulador *Mininet* para emular um plano de dados SDN e os componentes acima citados em uma camada de aplicação, que comunica-se com a camada de controle através de uma API *Representational State Transfer* (REST). Além disso, utilizou-se a rede real da *Sprint*, com 11 nós *Internet Service Provider*s (ISPs) distribuídos em várias cidades dos EUA. Nos testes, o modelo baseado em PBNM teve uma resposta significativamente melhor tanto em termos de perdas de pacote quanto em latência em períodos onde a rede apresentou maior carga de vídeo.

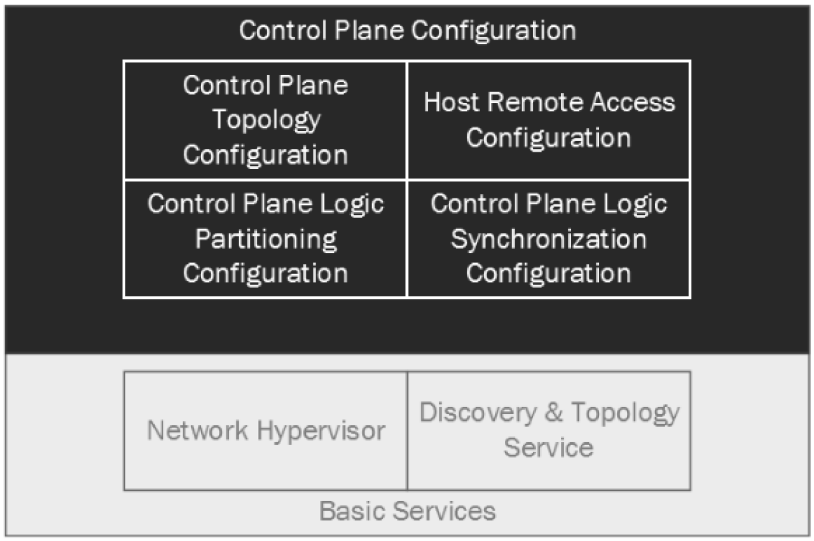
Percebe-se que o projeto tem um foco na funcionalidade de *self-optimization* dentro do contexto SON, e as configurações mencionadas são, na verdade, reconfigurações de fluxos através do ajuste das *flow tables* SDN, não tratando-se de um processo de *self-configuration*.

### 2.2.8 InitSDN

O initSDN é uma proposta de uma *self-configuration* de SDNCs em uma rede de forma automática e com flexibilidade e confiabilidade (PATIL; GOKHALE; HAKIRI, 2015). Essa *self-configuration* tem como foco o *bootstraping* da rede, ou seja, o processo de configuração inicial da mesma. Um problema inicial apontado foi que, embora o plano de dados de uma rede SDN possa efetivamente ser trafegado em um paradigma SDN, o mesmo não pode ser feito com o plano de controle – que flui em uma rede pré-SDN ou não-SDN – pois as regras de controle não podem ser estabelecidas antes que os controladores comuniquem-se com os NEs e entre si. Logo, todo processo de *bootstraping* é realizado através de uma comunicação em ambiente legado. Outro problema é que o processo de implantação de controladores distribuídos ainda é realizado de forma manual e estática.

A arquitetura initSDN contém seis módulos, divididos em "*Basic Services*" e "*Control Plane Configuration*", conforme mostrado na Figura 10. Por questões de simplificação, os módulos não serão apresentados aqui.

Figura 10 – Arquitetura Modular do InitSDN.




Fonte: (PATIL; GOKHALE; HAKIRI, 2015)

No processo de *bootstraping* da rede várias ações ocorrem em sequência, a saber: inicialização do initSDN (conectado estaticamente aos principais switches da rede), configuração do controlador initSDN, modelamento da topologia de rede, construção de uma topologia de plano de controle, fatiamento da rede em "*data-slice*" e "*control-slice*", instalação dos controladores em todos os *hosts* do plano de controle, configuração de acordo com a estratégia descrita pelo operador, alguns procedimentos de segurança e configuração de cada switch no plano de controle.

Foram realizadas implementações práticas em um protótipo inicial, baseado em *Mininet, Open Vswitch, Floodlight* e outros componentes, porém os resultados apresentados no

trabalho foram puramente qualitativos. Embora voltado a *self-configuration*, o initSDN limita-se ao processo de *bootstraping* da rede e apenas a controladores e não a NEs genéricos ou especializados.

### 2.2.9 SONAr

A arquitetura SONAr já foi apresentada e discutida em 2.1.7, de forma que não será detalhada nessa seção. A abordagem de *self-configuration* do SONAr é bem completa e contempla todos os itens avaliados, com exceção do caso específico de *self-configuration* especializada para NEs de borda.

### 2.2.10 Comparativo dos Trabalhos Correlatos

A Tabela 2 traz um resumo comparativo das soluções e trabalhos aqui apresentados, baseado nas funcionalidades de *self-configuration*, aderência a um *framework* com outros módulos SON e tipo de abordagem (teórica ou prática).

Tabela 2 – Comparativo das características entre os trabalhos correlatos.

| **Característica** | **SON 3GPP** | **SELF-NET** | **CogNet** | **Superfluidity** | **Network Management Framework** | **PBNM** | **InitSDN** | **Auto-Config. of SDN Switches** | **SONAr** |
|---|---|---|---|---|---|---|---|---|---|
| Está em um framework que engloba os demais itens de SON, além de auto-configuração? | Sim | Sim | Parc. | Parc. | Sim | - | - | - | Sim |
| Prevê auto-configuração de NEs SDN? | - | - | - | Parc. | Sim | - | Parc. | Sim | Sim |
| Prevê auto-configuração de NEs especializados? | Sim | - | - | Sim | - | - | - | - | - |
| Realiza auto-configuração em redes híbridas (SDN e não-SDN)? | Sim | - | - | - | - | - | Sim | Sim | Sim |
| Prevê coleta e manutenção de BD com mensagens do processo para histórico ou analytics? | Sim | Sim | Sim | N.I. | - | Sim | - | - | Sim |
| Propõe uma abordagem prática? | Sim | Sim | - | - | - | Sim | Sim | Sim | Sim |
| Apresenta resultados práticos? | Sim | Sim | - | - | - | Sim | - | - | Sim |

Legenda: SIM – Atende Plenamente ao quesito; Parc. – Atende o quesito parcialmente (de forma incompleta); N.I. – Não informado no trabalho; - – Não atende ao quesito

Fonte: elaborada pelo autor

De uma forma geral, percebe-se uma lacuna de propostas práticas que contemplem todos os itens descritos na tabela. Percebe-se também que a arquitetura SONAr é a mais completa e a mais adequada dentre as avaliadas para tomar-se como base para a implementação de *self-configuration* de redes para elementos de borda.

Capítulo 3

# Módulo de Auto-configuração Especializada

Este capítulo descreve a proposta para implementação do módulo de auto-configuração especializada – denominado *Specialized Self-Configuration Module* (SSCM) – no contexto da arquitetura SONAr. O capítulo está dividido da seguinte forma: em "Características Gerais", será apresentada uma breve descrição das características gerais do módulo SSCM, bem como seu objetivo dentro do *framework* SONAr; e em "Arquitetura do Módulo SSCM", será apresentado um detalhamento de seu funcionamento, incluindo as alterações dos fluxos de trabalho do *Self-Configuration Entity* (SCE), as funcionalidades cujas implementações são possíveis no módulo e o algoritmo de sua operação.

## 3.1 Características Gerais

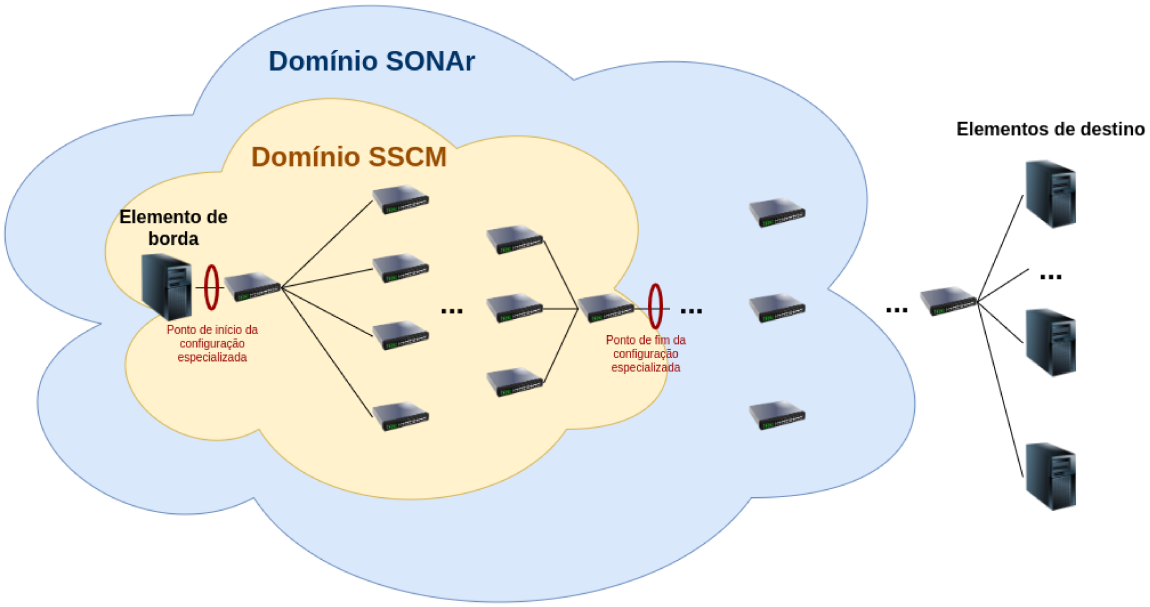
A proposta deste trabalho é a implementação de um módulo dentro do SCE no sistema SONAr, denominado *Specialized Self-Configuration Module* (SSCM), conforme será mostrado, dentro da Seção 3.2, na Figura 12.

O objetivo do SSCM é o de realizar a uma configuração específica de um determinado segmento de rede em resposta à inserção de um novo elemento de rede que exija um nível especializado de configuração, como, por exemplo, um elemento de borda que necessite implementações específicas de rede para seu funcionamento. O módulo deve identificar esse novo NE (tipicamente um elemento de borda com implementações específicas para funções de telecomunicações), verificar se há configurações específicas descritas para o mesmo e, se houver, implementá-las na rede, agindo como um *plug-and-play* especializado. Ele deve funcionar tanto para o caso específico que será demonstrado – que será o da implementação de uma nova eNodeB em uma rede celular – mas também deve ser flexível para utilização em quaisquer casos práticos, isto é, suportar a configuração para outros tipos de elementos de borda e outras configurações de topologia.

A aplicabilidade do funcionamento do SSCM pressupõe que haja dois pontos bem

definidos para início e fim da configuração especializada: o primeiro ponto é interface de saída do próprio elemento de borda (*edge NE*) e o segundo, a interface de saída do elemento onde a rede converge para distribuição dos elementos destino, normalmente um roteador *gateway* ou uma switch agregadora em um core de rede. Pressupõe-se ainda que todos os elementos dentro do domínio de *self-configuration* do SSCM estejam também sobre a esfera de atuação do SONAr, ou seja, todos os elementos estejam "visíveis" aos componentes que integram a arquitetura (NEM, SCE, CPI, SDNC, etc). Os elementos de destino final dos pacotes enviados pelo elemento de borda podem estar dentro ou fora do domínio do SONAr e podem ser um ou vários, porém o elemento de borda em si deve ser único (identificável por sua interface de saída) e deve estar necessariamente dentro do domínio SONAr. Esse domínio de atuação é representado na Figura 11.

Figura 11 – Requerimentos de topologia para aplicabilidade do SSCM.




Fonte: elaborada pelo autor

O SSCM é um módulo dentro do SCE que atua apenas quando, dentro do processo de *plug-and-play*, for identificado um elemento cujo endereço MAC esteja relacionado a uma *Specialized Self-Configuration Description* (SSCD), que é um registro em uma *keyspace* específica no NDB, denominada *NEBootConfiguration*. Caso não esteja, o processo de *plug-and-play* ocorre normalmente, considerando que o elemento conectado seja um NE padrão e com configurações idem. Mas, caso o endereço físico da interface conectada pertença a um elemento constante nesta tabela do NDB, uma alteração no fluxo do SCE será realizada, de modo que a configuração específica seja feita. Esse SSCD na *keyspace NEBootConfiguration* é descrito em formato *JavaScript Object Notation* (JSON) e contém as características de configuração da rede que irá receber o elemento. A especificação desse formato será apresentado com mais detalhes na Subseção 3.2.3. O SSCD é incluído

no NDB através de módulo NAD, na *Administration Layer*, que fornece uma interface amigável com o operador e/ou administrador do sistema. No entanto, a implementação da interface geradora do SSCD no NAD não faz parte do escopo deste trabalho e não será discutida ou apresentada aqui.

Ressalta-se aqui um papel adicional de segurança do SSCD: ele faz com que um componente necessariamente deva estar especificado no NDB para que sua configuração especializada seja realizada. Com isso, evita-se que NEs não autorizadas sejam automaticamente configuradas e, em teoria, entrem em funcionamento de forma irregular na rede. Para que o elemento seja corretamente adicionado à topologia de rede, ele deve ser previamente cadastrado via NAD por um operador autorizado e habilitado para tal. Porém, embora cumpra esse papel adicional de segurança, o SSCD não garante que o sistema esteja totalmente protegido de ataques como no caso de MAC *spoofing*, por exemplo. Essa garantia é, em última instância, responsabilidade da SPE.

A implementação do SSCM foi feita em Java 8, seguindo o padrão para todo sistema SONAr. As especificações de todos os softwares e versões utilizados na implementação serão apresentados no Capítulo 4.

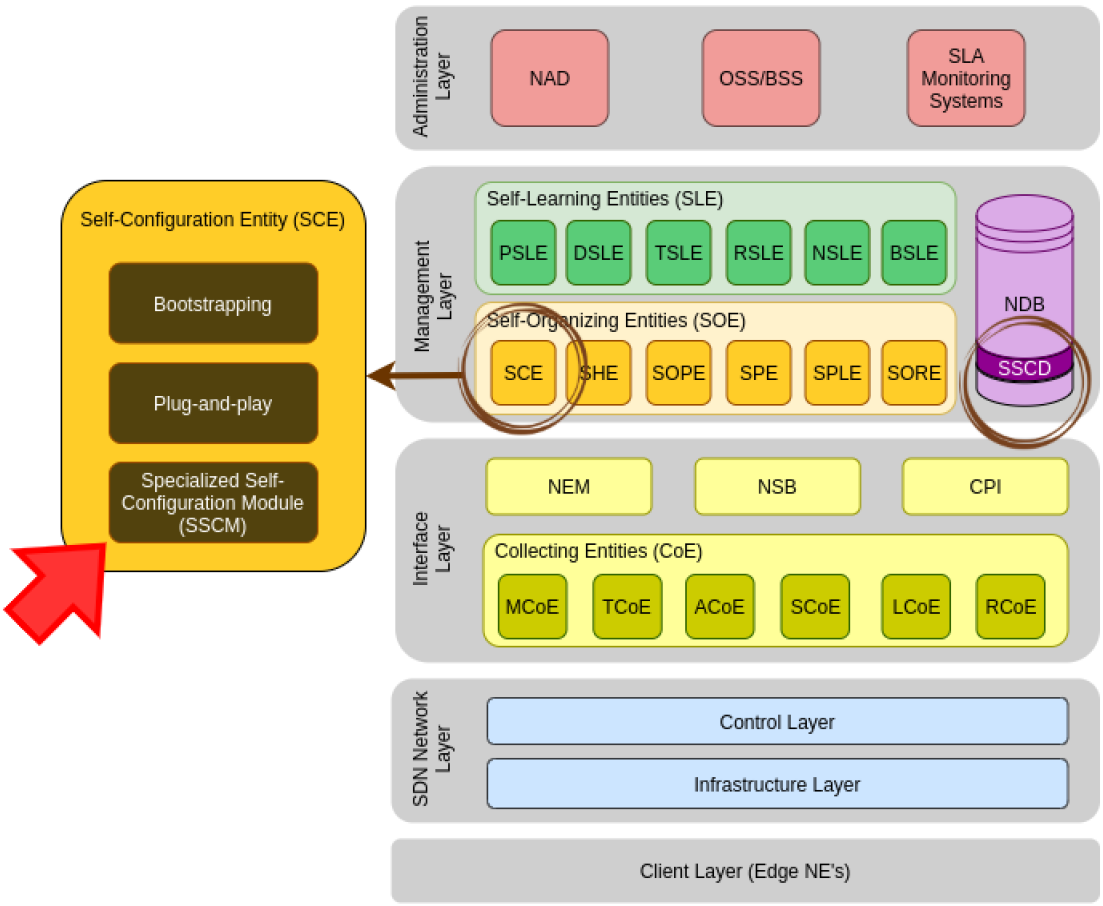
## 3.2 Arquitetura do Módulo SSCM

A arquitetura do sistema SONAr pode ser revista de modo a incluir agora o módulo SSCM e a tabela com os SSCDs, conforme mostrado na Figura 12. Esta figura complementa a Figura 5 (*Diagrama simplificado em camadas da arquitetura SONAr*) apresentada na Seção 2.1.7, com a identificação e localização dos novos módulos dentro das camadas que compõem a arquitetura SONAr.

Conforme já mencionado, o SSCM está incluído dentro da *Management Layer*, *Self-Organizing Entities sub-layer* e inserido dentro da *Self-Configuration Entity*. Os demais módulos que compõem essa entidade são o módulo de *bootstrapping*, responsável pela configuração inicial da rede uma vez que ela seja colocada em funcionamento, e o *plug-and-play*, que implementa a funcionalidade de *self-configuration* de elementos inseridos em uma rede já em funcionamento. Já o SSCD é um registro em uma tabela específica dentro do NDB, cujo formato será detalhado na Seção 3.2.3.

O SSCM é um serviço complementar ao *plug-and-play*: uma vez ativa, quando essa última funcionalidade detecta que o elemento inserido consta no NDB demandando uma configuração específica, o SSCM é iniciado para realizar esse processo. A chave que dispara esse processo é o endereço MAC do elemento de borda adicionado, sendo esse também o campo-chave na *keyspace NEBootConfiguration* da NDB, podendo corresponder a um SSCD. Enquanto o *plug-and-play* continua desempenhando seu papel, fazendo as configurações de rotas naturais entre os elementos – incluindo habilitação dos fluxos de *Address Resolution Protocol* (ARP), *Link Layer Discovery Protocol* (LLDP) e demais

Figura 12 – Diagrama simplificado em camadas da arquitetura SONAr mostrando módulo SSCM e a base SSCD.




Fonte: adaptada de (GONÇALVES et al., 2020)

protocolos – o SSCM encarrega-se de configurações mais especializadas, como a marcação de pacotes com cabeçalho IEEE 802.1Q (VLAN), priorizações em *layer* 2 (PCP), *layer* 3 (*DiffServ*), encaminhamentos específicos e outras ações, dependendo da especificação inserida na NDB. Essas configurações são realizadas em todos os elementos de rede no caminho entre o elemento inicial (NE de borda) e o elemento final identificado no SSCD (que não necessariamente é o elemento de destino final dos pacotes), isto é, são implementados os fluxos (*flow rules*) específicos em todos os elementos no caminho entre estes dois NEs.

Dessa forma, o SSCM pressupõe que o *bootstrapping* já tenha sido executado (e, de fato, esse último é realizado antes dos demais módulos) e as configurações básicas do *plug-and-play* já tenham sido implementadas, sendo ele o último módulo a ser executado dentro da SCE.

### 3.2.1 Alteração do Fluxo no SONAr

O início do processo de qualquer *self-configuration* no contexto do *plug-and-play* na arquitetura SONAr, seja especializado ou genérico, inicia-se na solicitação de um IP da rede pelo dispositivo, através de uma mensagem de *Dynamic Host Configuration Protocol* (DHCP) *Discovery*. Essa mensagem em *broadcast* chega ao DHCP *server* do SONAr, denominado *Address Provider* (AP). O AP faz a consulta dos IPs disponíveis na NDB, oferece um endereço ao novo NE, seguindo a troca de mensagens normais do protocolo DHCP. A principal diferença entre o AP e um servidor DHCP padrão é que o AP, além de fornecer e gerenciar os endereços, também publica um evento "*device identified*" no NEM.

O TCoE assina esse tópico em específico no NEM e recebe o evento "*device identified*". Em resposta, realiza um processo inicial de *discovery* para encontrar o ponto de conexão do novo elemento. Uma vez descoberto, o TCoE publica o evento "*device found*" no NEM.

Por sua vez, o SCE assina esse último tópico no NEM, então irá receber esse evento "*device found*", o que disparará uma nova sequência de ações: ela irá calcular as rotas de acesso entre esse novo dispositivo e os componentes SONAr, adicionar essas as rotas na infraestrutura e publicar o evento "*device accessible*" também no NEM.

Uma vez que o dispositivo esteja acessível pelos componentes SONAr, é a vez do TCoE – que assina o tópico e recebe a mensagem "*device accessible*" – iniciar o segundo estágio do processo de *discovery*, autenticando o mesmo e notificando os demais componentes de sua disponibilidade, com a publicação do evento "*device discovered*".

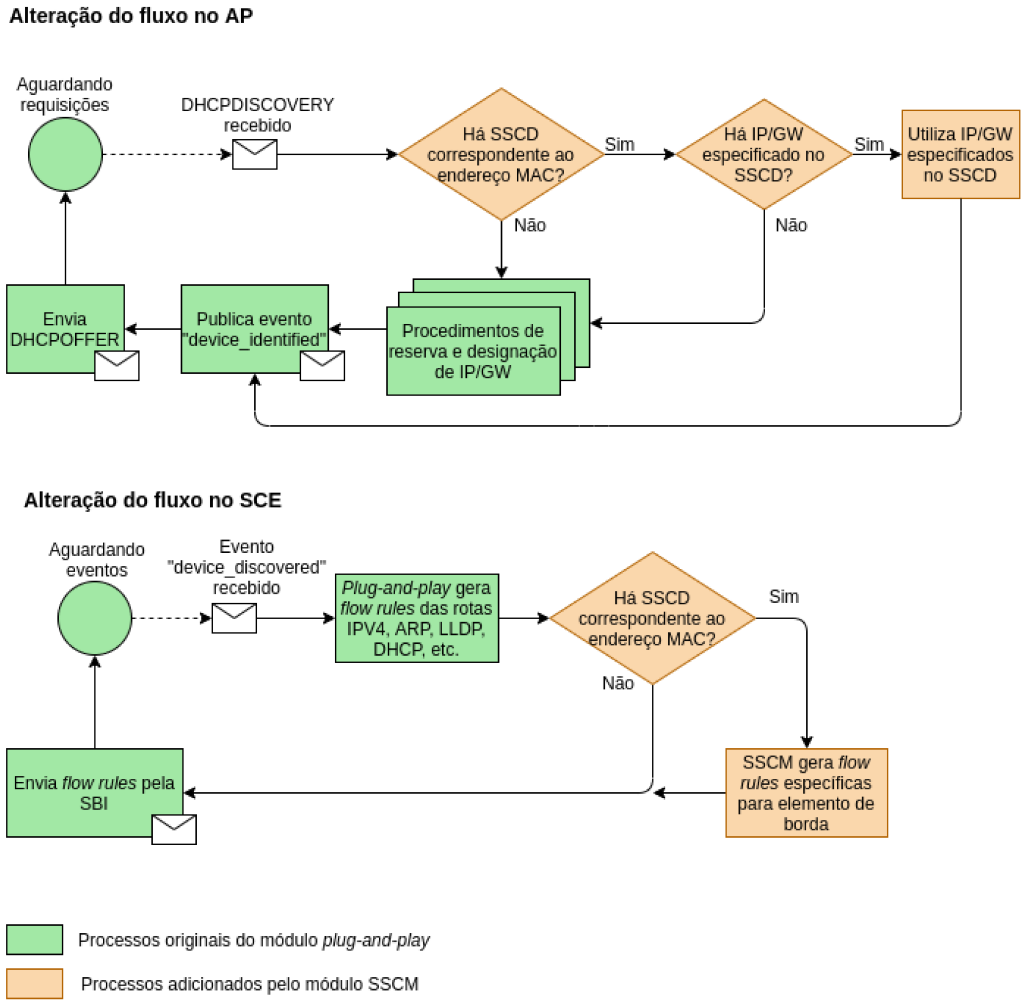
Só então, com o recebimento do evento "*device discovered*", o SCE realiza o processo final do *plug-and-play*: envia a configuração de acesso ao controlador (SDNC) ao elemento, aguarda sua conexão e gera as *flow rules* das rotas padrões IPV4 e de controle(já calculadas pelo TCoE). Essas rotas permitem tanto um roteamento genérico quanto a troca de mensagens padrões dos protocolos de camada 2 e camada 3: ARP, LLDP e DHCP (*proxy* e *broadcast* de entrada e saída). Finalmente, envia todas as rotas ao controlador através da NBI, que as encaminha aos elementos da rede via SBI.

Com a inserção do módulo SSCM, duas etapas do processo de *self-configuration* são alteradas. Essas alterações são mostradas na Figura 13 e descritas a seguir.

A primeira alteração é no AP: ao receber a mensagem de *DHCPDISCOVERY*, o AP consulta antes o NDB para verificar se existe um SSCD associado ao endereço MAC constante na mensagem enviada em *broadcast*. Se houver, na mensagem *DHCPOFFER* ele oferece o endereço IP e o endereço do *gateway* especificados no SSCD, ao invés de buscar por um IP livre no NDB. Isso garante que o NE de borda possa ter seu IP e *gateway* definidos pelo engenheiro ou operador da rede antes mesmo de ser plugado à rede.

A segunda alteração ocorre no SCE: após gerar as *flow rules* com as rotas *default*, o SCE também verifica o endereço MAC da interface do elemento inserido. Caso ele

Figura 13 – Alteração de fluxo nos elementos SONAr.




Fonte: elaborada pelo autor

conste na base de SSCDs no NDB, ele inicia um processo de configuração especializada, gerando *flow rules* específicas para o elemento de borda. Esse processo permite que, além de ações genéricas, como simples configurações de rotas *default*, o SCE faça configurações mais elaboradas na topologia, de acordo com o especificado no SSCD. Após finalizadas as gerações de todas as *flow rules*, elas são enviadas ao controlador via NBI. O detalhamento do processo de geração das configurações especializadas será descrito na Seção 3.2.3.

### 3.2.2 Funcionalidades Disponíveis

O processo de *self-configuration* do SSCM foi implementado utilizando-se as funcionalidades disponíveis no controlador ONOS, através de comandos via API REST na NBI, as denominadas *flow rules* ou *flow entries*. Neste capítulo, o termo *flow rule* será utilizado.

A *flow rule* é normalmente descrita em formato JSON e é composta de um número de identificação da *flow rule* e tabela, prioridade na execução (caso mais de uma *flow rule*

atenda aos critérios do pacote), identificação do dispositivo em que irá ser aplicada, de ações a serem executadas caso haja correspondência de um critério e dos próprios critérios que identificam as condições às quais ela se aplica. Normalmente segue um formato como o abaixo indicado (alguns atributos menos relevantes foram omitidos):

```
"id": "...",
"tableId": "...",
(...)
"priority": ...,
(...)
"deviceId": "...",
"treatment": {
  "instructions": { ... }
},
  "selector": {
    "criteria": { ... }
  }
}
```

Ou seja, quando um pacote chega à switch, ela verifica se o mesmo atende ao critério descrito no atributo *criteria* de alguma *flow rule* instalada. Caso atenda a mais de um *flow rule*, o que tiver maior prioridade descrita em *priority* irá ditar as ações a serem realizadas, especificadas no atributo *instructions*. A *flow rule* enviada na NBI será instalada no dispositivo discriminado em *deviceId*, que corresponde ao número do dispositivo Openflow usado pelo SDNC.

Para melhor compreensão, um exemplo de *flow rule* em formato JSON para envio via na NBI é mostrado abaixo. Ele exemplifica um fluxo real implementado em uma switch durante os testes realizados com o SSCM.

```
"id": "53480248152484957",
"tableId": "0",
"appId": "org.onosproject.rest",
"groupId": 0,
"priority": 6000,
"timeout": 0,
"isPermanent": true,
"deviceId": "of:0000f67755a2284f",
"state": "PENDING_ADD",
"life": 0,
"packets": 0,
"bytes": 0,
"liveType": "UNKNOWN",
"lastSeen": 1600907143567,
"treatment": {
  "instructions": [
    {
```

```
        "type": "L2MODIFICATION",
        "subtype": "VLAN_POP"
      },
      {
        "type": "OUTPUT",
        "port": "14"
      }
    ],
    "deferred": []
},
    "selector": {
      "criteria": [
        {
          "type": "ETH_TYPE",
          "ethType": "0x800"
        },
        {
          "type": "IPV4_SRC",
          "ip": "192.168.0.200/32"
        },
        {
          "type": "IPV4_DST",
          "ip": "10.1.1.201/32"
        }
      ]
    }
}
```


Conforme já comentado, esta *flow rule* é constituída de um critério de *match* (atributo *criteria*), que dá a condição para que uma ação seja tomada sobre o pacote, e a instrução (atributo *instruction*), que diz ao switch quais as ações devem ser realizadas se um pacote corresponde aos critérios de *match* citados. Ela é enviada via NBI para o SDNC que, por sua vez, a envia à switch-alvo designada acima pelo atributo *deviceId* via protocolo OpenFlow. No exemplo anterior, a *flow rule criteria* é composta das seguintes condições:

- O *EtherType* (ETH_TYPE) do pacote deve ser 0x800 (ou seja, um pacote do tipo IPV4);
- O IP de origem (IPV4_SRC) do pacote tem que ser 192.688.0.200;
- O IP de destino (IPV4_DST) do pacote tem que ser 10.1.1.201.

Caso os critérios acima sejam cumpridos, a seguinte relação de instruções deve ser realizada:

- A *tag* VLAN (cabeçalho IEEE 802.1Q) deve ser retirada do pacote;

- O pacote deve ser encaminhado para a porta 14 (seguindo a designação OpenFlow de portas na OvS).

Trata-se, portanto de um conjunto de ações específicas a serem realizadas quando um determinado conjunto de critérios são atendidos.

Na versão do ONOS utilizada, os conjuntos de critérios e instruções disponíveis para serem usados como *flow criteria* e *flow instructions* são mostrados nas Tabelas 3 e 4, respectivamente (ONOS PROJECT, 2019). Algumas outras instruções foram implementadas recentemente no Projeto ONOS porém não constam da documentação oficial. No escopo deste trabalho serão utilizadas apenas os critérios e instruções documentados na página oficial do Projeto ONOS e necessários para a implementação do caso prático escolhido, como será mostrado no Capítulo 4.

Tabela 3 – Relação de *Flow Criteria* especificados no Projeto ONOS.

| **Type (Criteria)** | **Descrição** |
|---|---|
| ETH_TYPE | Tipo de frame ethernet |
| ETH_DST | Endereço MAC de destino |
| ETH_SRC | Endereço MAC de origem |
| IN_PORT | Porta de entrada |
| IN_PHY_PORT | Porta de entrada (física) |
| METADATA | Metadados passados entre tabelas |
| VLAN_VID | VLAN Id |
| VLAN_PCP | Prioridade VLAN |
| INNER_VLAN_VID | VLAN Id (inner) |
| INNER_VLAN_PCP | Prioridade VLAN (inner) |
| IP_DSCP | DSCP (6 bits no campo ToS/DiffServ) |
| IP_ECN | ECN (2 bits no campo ToS/DiffServ) |
| IP_PROTO | Protocolo IP |
| IPV4_SRC | IP de origem |
| IPV4_DST | IP de destion |
| TCP_SRC | Endereço de origem TCP |
| TCP_DST | Endereço de destino TCP |
| UDP_SRC | Endereço de origem UCP |
| UDP_DST | Endereço de destino UDP |
| SCTP_SRC | Endereço de origem SCTP |
| SCTP_DST | Endereço de destino SCTP |
| ICMPV4_TYPE | Internet Control Message Protocol for IPV4 type (RFC0792) |
| ICMPV4_CODE | Internet Control Message Protocol for IPV4 code (RFC0792) |
| IPV6_SRC | IPV6 de origem |
| IPV6_SRC | IPV6 de destino |
| IPV6_FLABEL | IPV6 Flow Label (RFC 6437) |
| ICMPV6_TYPE | Internet Control Message Protocol for IPV6 type (RFC2463) |
| ICMPV6_CODE | Internet Control Message Protocol for IPV6 code (RFC2463) |
| IPV6_ND_TARGET | IPV6 Neighbor discovery target address |
| IPV6_ND_SLL | IPV6 Neighbor discovery MAC address |
| IPV6_ND_TTL | IPV6 Neighbor discovery TTL MAC address |
| MPLS_LABEL | Label MPLS |
| IPV6_EXTHDR | Cabeçalho de extensão IPV6 |
| OCH_SIGID | Abstração de comprimento de onda |
| GRID_TYPE | Tipo de grid de comprimento de onda |
| CHANNEL_SPACING | Espaçamento de canal óptico |
| SPACING_MULIPLIER | Multiplicador de espaçamento de canal óptico |
| OCH_SIGID | Identificador de sinal de canal óptico |
| TUNNEL_ID | Tunnel ID |
| OCH_SIGID | Typo de sinal de canal óptico |
| ODU_SIGID | ODU (Optical channel Data Unit) signal ID |
| tributaryPortNumber | OPU (Optical channel Payload Unit) port number |
| tributarySlotBitmap | OPU (Optical channel Payload Unit) slot length |
| tributarySlotLen | OPU (Optical channel Payload Unit) slot bitmap |
| ODU_SIGTYPE | ODU (Optical channel Data Unit) signal type |

Fonte: adaptada de (ONOS PROJECT, 2019)

Tabela 4 – Relação de *Flow Instructions* especificados no Projeto ONOS.

| **Type (Instruction)** | **Sub-Type** | **Descrição** |
|---|---|---|
| OUTPUT | - | Define porta de saída do pacote |
| TABLE | - | Passa o tráfego para outra tabela |
| METER | - | O tráfego deve ser medido de acordo com uma métrica |
| QUEUE | - | O tráfego deve ser colocado em outra fila (queue) em uma porta |
| L0MODIFICATION | LAMBDA | Modifica em L0 com um comprimento de onda específico |
| L0MODIFICATION | OCH | Modifica em L0 com um Och (Optical Channel) específico |
| L1MODIFICATION | ODU_SIGID | Modifica em L1 o sinal ODU em rede óptica |
| L2MODIFICATION | VLAN_POP | Retira o cabeçalho L2 IEEE 802.1Q (marcação de VLAN) |
| L2MODIFICATION | VLAN_PUSH | Insere em L2 a marcação VLAN Id |
| L2MODIFICATION | VLAN_PCP | Insere em L2 a marcação VLAN PCP (priorização em L2) |
| L2MODIFICATION | ETH_SRC | Modifica em L2 o endereço ethernet de origem do pacote |
| L2MODIFICATION | ETH_DST | Modifica em L2 o endereço ethernet de destino do pacote |
| L2MODIFICATION | MPLS_LABEL | Insere label MPLS no pacote |
| L2MODIFICATION | MPLS_PUSH | Insere cabeçalho MPLS no pacote |
| L2MODIFICATION | TUNNEL_ID | Insere tunnel Id MPLS no pacote |
| L3MODIFICATION | IPV4_SRC | Modifica em L3 o IP de origem |
| L3MODIFICATION | IPV4_DST | Modifica em L3 o IP de destino |
| L3MODIFICATION | IPV6_SRC | Modifica em L3 o IPV6 de origem |
| L3MODIFICATION | IPV6_DST | Modifica em L3 o IPV6 de destino |
| L3MODIFICATION | IPV6_FLABEL | Modifica em L3 o campo Flow Label no cabeçalho IPV6 |
| L4MODIFICATION | TCP_SRC | Modifica em L4 a porta de origem (source port) no protocolo TCP |
| L4MODIFICATION | UDP_SRC | Modifica em L4 a porta de origem (source port) no protocolo UDP |

Fonte: adaptada de (ONOS PROJECT, 2019)

### 3.2.3 Aspectos Funcionais

Todo o processo de *self-configuration* no SSCM inicia-se pela especificação da configuração especializada a ser feita na rede para que a mesma esteja preparada para conectar um determinado elemento de borda a seu destino. Essa especificação é realizada através de um objeto em formato JSON contendo os atributos que descrevem as especificidades da rede em questão, denominado SSCD. Este SSCD é um registro na *keyspace NEBoot-Configuration*, no NDB. Um exemplo de uma entrada SSCD é mostrada abaixo:

```
{
  "id": {
    "mac_address": "00:00:00:00:00:01",
    "description": "Edge NE Description Here",
    "responsible" : "John Doe",
    "ip": "192.168.0.100",
    "gateway": "192.168.0.200"
  },
  "config_end_node": {
    "mac_address": "00:00:00:00:00:02",
    "ip": "192.168.0.200",
    "mask_bits": 24,
    "config_end_node_description": "Last NE to Configure"
  },
  "dst_node": [
    {
      "description": "First Destination",
      "ip": "10.1.1.1",
      "mask_bits": 24,
      "priority": 5000,
      "vlan": 10,
      "pcp_cos": 1
    },
    {
      "description": "Second Destination",
      "ip": "10.1.1.2",
      "mask_bits": 24,
      "priority": 5000,
      "vlan": 20,
      "pcp_cos": 2
    },
    {
    ...
    }
  ]
}
```

Os atributos deste objeto SSCD são interpretados da seguinte forma:

- *id*: É o identificador do elemento de borda a ser automaticamente inserido na rede e o que marca o ponto inicial de configuração da rede. Nele especifica-se o endereço MAC, que é o principal identificador do elemento (e que vai ser usado para disparar o processo do SSCM) e outras informações complementares, como a descrição do elemento, o responsável pela entrada do registro na SSCD, o IP e o *gateway* do elemento. Caso não existam os identificadores de IP e *gateway*, os mesmos serão definidos e atribuídos pelo AP, caso contrário, o AP irá ler estes identificadores e enviá-los na mensagem *DHCPOFFER* ao NE de borda. Os identificadores de descrição do elemento e do responsável pela entrada do registro cumprem apenas papéis de documentação, portanto também são opcionais e não têm função real dentro do processo de *self-configuration*.

- *config_end_node*: Identifica o elemento final do escopo de *self-configuration* da rede, isto é, o processo de *self-configuration* é realizado entre o elemento de borda e o elemento indicado nesse campo. Esse elemento é tipicamente um *gateway* ou um elemento agregador. O principal identificador desse elemento também é o endereço MAC da porta de saída para o próximo elemento (que não será configurado automaticamente). Os demais campos também não necessariamente são usados e apenas cumprem a função de documentação.

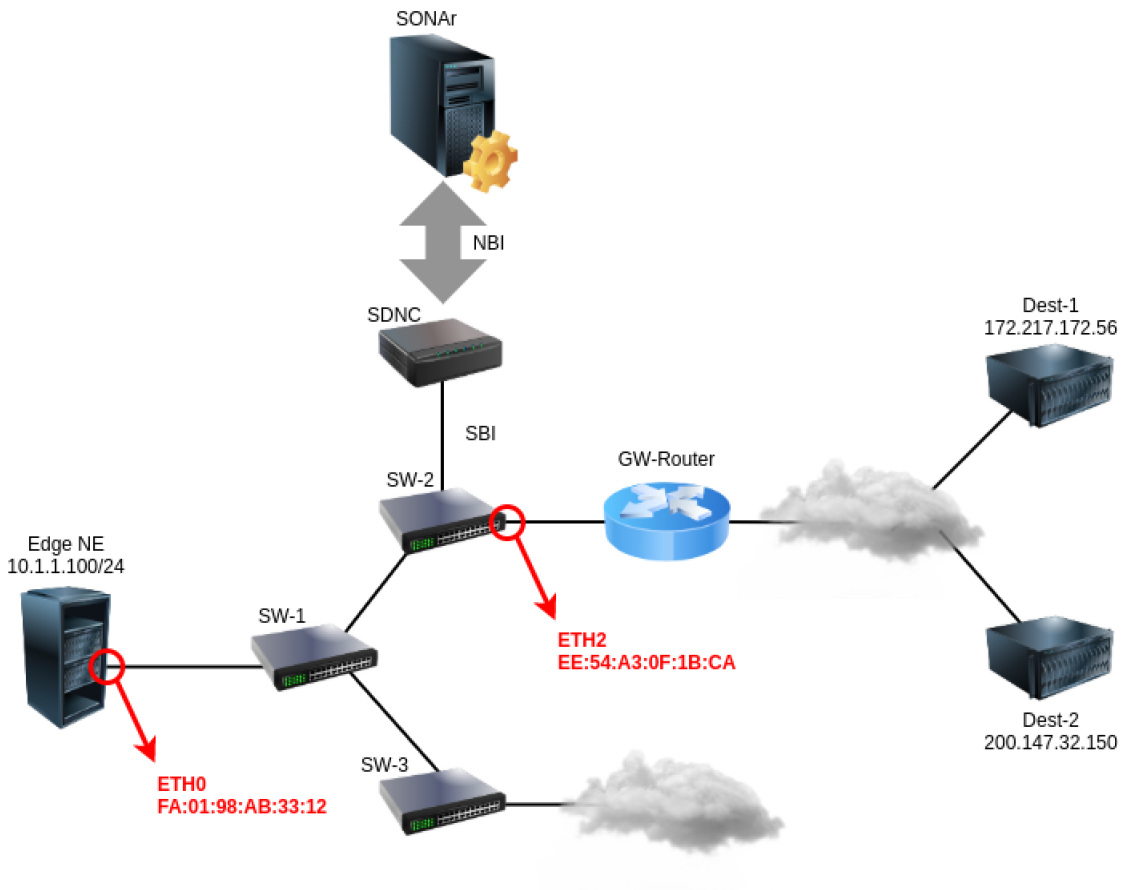
- *dst_node*: É um *array* que contém os destinos possíveis dos pacotes enviados pelo elemento de borda e a especificação do que se fazer com eles, devendo estar além do *config_end_node* e podendo estar fora do domínio SONAr. É aqui que se caracterizam as ações a serem implementadas nas configurações da rede. Por exemplo, um pacote enviado do elemento de borda a um servidor (identificado pelo IP) deve trafegar em uma VLAN específica com um determinado *Class of Service* (CoS). O campo *priority* refere-se à prioridade do fluxo dentro do contexto SDN e não à priorização do pacote na rede, sendo que é recomendado que essa prioridade seja maior do que as das *flow rules default* inseridas pelo *plug-and-play* que no padrão SONAr é 4.000. A lista de ações possíveis a serem implementadas é dada pela Tabela 4 mostrada anteriormente. Esse *array* pode conter tantos elementos de destino com ações específicas quanto for necessário, dependendo do tipo de aplicação. Como exemplo, uma eNodeB celular usa quatro destinos finais (dados, controle, gerência e sincronismo), enquanto que uma *Optical Line Terminal* (OLT) utiliza três destinos (dados, voz e vídeo), em ambos os elementos com níveis de QoS específicos para cada um desses destinos.

O processo de *self-configuration* pode ser exemplificado através de uma rede como o da Figura 14. O elemento de borda inserido é o *Edge NE*. Ele é conectado à rede através da interface ETH0 ao switch SW-1. Como é um elemento especializado, ele deverá enviar pacotes a um servidor *Dest-1* em uma VLAN específica, por exemplo, uma

*VLAN Id* (VID) 115, com uma certa marcação de QoS em *layer* 2 (por exemplo, CoS 3) e a um servidor *Dest-2* em outra VLAN (VID 120), com uma outra marcação de QoS em *layer* 2 (CoS 1).

O domínio da rede sobre a gerência do SONAr é delimitado pelo elemento *GW-Router*, isto é, a rede além deste elemento (ele incluído) não é "enxergada" nem está sob controle do SONAr. Logo, é importante estabelecer-se o ponto inicial e o ponto final de onde as configurações devem ser realizadas que são, respectivamente, a porta ETH0 da *Edge NE* e a porta ETH2 da SW-2.

Figura 14 – Exemplo de topologia de rede.




Fonte: elaborada pelo autor

Neste caso, os elementos a serem configurados pelo SONAr são os switches SW-1 e SW-2, pois estão no caminho mais curto (e, neste caso, o único) entre a *Edge NE* e o *GW-Router*. Para esse exemplo, o objeto SSCD seria:

```
{
  "id": {
    "mac_address": "fa:01:98:ab:33:12",
    "description": "Edge NE",
    "responsible" : "Administrador 01",
    "ip": "10.1.1.100",
```

```
    "gateway": "10.1.1.200"
  },
  "config_end_node": {
    "mac_address": "ee:54:a3:0f:1b:ca",
    "config_end_node_description": "Access Switch SW-2"
  },
  "dst_node": [
    {
      "description": "Dest-1",
      "ip": "172.217.172.56",
      "priority": 5000,
      "vlan": 115,
      "pcp_cos": 3
    },
    {
      "description": "Dest-2",
      "ip": "200.147.32.150",
      "priority": 5000,
      "vlan": 120,
      "pcp_cos": 1
    }
  ]
}
```

Uma vez conectado o *Edge NE*, o AP receberá uma mensagem de solicitação de endereço IP desse elemento (mensagem *DHCPDISCOVERY*), identificará a correspondência de seu endereço MAC e um SSCD, designará o IP e *gateway* especificados nesse último (mensagem *DHCPOFFER*) e publicará um evento "*device identified*" no NEM. Esse evento dispara as ações do TCoE e do SCE, conforme já descrito em na Subseção 3.2.1. No processo de *self-configuration* efetuado pelo SCE, uma vez identificado que há uma entrada SSCD no NDB correspondente ao endereço MAC desse elemento, o SSCM determina o caminho identificado entre o *Edge NE* e o *config_end_node* (no caso, o *GW-Router*) e a montagem das *flow rules* específicas para cada elemento com esses parâmetros. Esse caminho já foi calculado anteriormente pelo TCoE, utilizando-se o algoritmo de Dijkstra para determinação da menor distância entre os dois elementos, sendo que esse processo não faz parte do escopo deste trabalho.

No algoritmo utilizado no SSCM, a elaboração das *flow rules* inicia-se no último elemento – no exemplo anterior, a SW-2. Os fluxos implementados nessa switch são:

- Para pacotes com IP de origem 10.1.1.100 e IP de destino 172.217.172.56:
  - Retirar cabeçalho 802.1Q (tag VLAN)
  - Encaminhar para ETH2
  - Marcar a *flow rule* com prioridade 5000

- ❏ Para pacotes com IP de origem 10.1.1.100 e IP de destino 200.147.32.150:
  - Retirar cabeçalho 802.1Q (tag VLAN)
  - Encaminhar para ETH2
  - Marcar a *flow rule* com prioridade 5000
- ❏ Para pacotes com IP de origem 172.217.172.56 e IP de destino 10.1.1.100:
  - Inserir cabeçalho 802.1Q com VLAN Id 115 e CoS 3
  - Encaminhar para ETH1
  - Marcar a *flow rule* com prioridade 5000
- ❏ Para pacotes com IP de origem 200.147.32.150 e IP de destino 10.1.1.100:
  - Inserir cabeçalho 802.1Q com VLAN Id 120 e CoS 1
  - Encaminhar para ETH1
  - Marcar a *flow rule* com prioridade 5000

A seguir, o algoritmo passa para o cálculo das *flow rules* do SW-1, que serão similares (porém invertidas) às da SW-2. Por fim, o SCE envia todas as *flow rules* ao SDNC via API REST na NBI.

De uma maneira geral, o algoritmo pode ser expressado da seguinte forma:

**Algoritmo 1** Algoritmo do módulo SSCM

```
i ← 1
if path.size() == 1 then
    no sentido edge_NE para dest_node: gera flow rule que encaminha pacote
    no sentido dest_node para edge_NE: gera flow rule que encaminha pacote
else
    for (i < path.size(); i++) do
        if network_element(i) == endElement then
            j ← 1
            for (j < dest_node.arraySize(); j++) do
                no sentido edge_NE para dest_node: gera flow rule que retira todas marcações e encaminha pacote
                no sentido dest_node para edge_NE: gera flow rule que insere marcações e encaminha pacote
            end
            else if network_element(i) != startElement && network_element(i) != endElement then
                j ← 1
                for (j < dest_node.arraySize(); j++) do
                    no sentido edge_NE para dest_node: gera flow rule que encaminha pacote
                    no sentido dest_node para edge_NE: gera flow rule que encaminha pacote
                end
                else if network_element(i) == startElement then
                    j ← 1
                    for (j < dest_node.arraySize(); j++) do
                        no sentido edge_NE para dest_node: gera flow rule que insere marcações e encaminha pacote
                        no sentido dest_node para edge_NE: gera flow rule que retira todas marcações e encaminha pacote
                    end
                end
            end
        end
    end
end
onosManager.configureFlows()
```

Pelo algoritmo usado pelo TCoE, o primeiro elemento a ser configurado no caminho é o último elemento, devido ao método utilizado para esse traçado. O algoritmo do SSCM percorre elemento de rede por elemento de rede, verificando se ele é o elemento final, o inicial ou um dos intermediários no caminho identificado pelo TCoE. Quanto é um elemento inicial ou final, ele fará as marcações indicadas no SSCD para cada um dos *dest_nodes*. Essas marcações podem ser quaisquer disponíveis na Tabela 4, mas, na prática, serão tipicamente referentes a VLAN, QoS ou marcações *Multiprotocol Label Switching* (MPLS).

Importante reparar que essas marcações ocorrem quando o pacote está entrando na rede de domínio do SONAr, sendo retiradas no sentido contrário. O intuito disso é manter-se a integridade dos metadados originais dos pacotes quando estiverem em redes não controladas ou gerenciadas pelo sistema SONAr, deixando as decisões dessa rede transparente às redes externas.

Quando o elemento é intermediário, é instruído apenas a encaminhar o pacote, sem alterar seus metadados. O mesmo acontece quando há apenas uma única switch no caminho, indicado pela primeira condição "*If*" no algoritmo: se for o caso, ou seja se o *path.size()* contiver apenas um elemento, apenas os encaminhamentos devem ser feitos.

Esse é um comportamento típico de redes de telecomunicações, onde os pacotes percorrem redes de acesso em VLANs distintas, com diferentes priorizações dependendo de sua função. Como será visto no exemplo apresentado no Capítulo 4, pacotes de sincronismo de rede têm que possuir uma altíssima prioridade na entrega quando comparado com outros tipos, pois se houver perda de sincronismo, os demais planos de usuário e controle deixarão também de funcionar. Essa é o principal propósito do módulo SSCM, a *self-configuration* de redes de aplicações específicas, como as que suportam sistemas de telecomunicações.

Capítulo 4

# Experimentos e Análise dos Resultados

Neste capítulo serão apresentados os temas pertinentes à implantação prática do módulo e experimentos realizados, incluindo uma descrição do ambiente utilizado na implementação do projeto e do cenário escolhido para avaliação, além do detalhamento dos experimentos executados e a avaliação dos resultados obtidos.

## 4.1 Método para a Avaliação

Nesta seção serão descritos o ambiente de implementação do projeto, além do cenário escolhido para avaliação prática do sistema e realização dos experimentos.

### 4.1.1 Ambiente de Implementação

A primeira etapa para implementação prática do projeto e subsequente avaliação experimental do mesmo é a definição de um ambiente para tal. Uma parte substancial do trabalho foi a montagem de um ambiente adequado para uma implementação fidedigna do projeto e realização dos experimentos. A seguir serão apresentados os componentes de *hardware* e *software* utilizados neste ambiente, juntamente com suas versões

#### 4.1.1.1 Servidor

O *hardware* utilizado foi o um servidor da Faculdade de Computação da Universidade Federal de Uberlândia (FACOM/UFU), virtualizado utilizando-se VMware, com as seguintes configurações:

- Sistema operacional Ubuntu 18.10
- 16 GBytes de RAM
- 60 GBytes de HD
- Processador com 8 cores operando a 2 GHz

#### 4.1.1.2 Emulação de Rede

Para emulação do ambiente de redes, foi escolhido o software *Graphical Network Simulator-3* (GNS3), versão 2.1.16. Ele foi escolhido por diversos motivos, mas o principal é o fato de ele não ser um simulador, mas um ambiente que permite o lançamento de elementos em *containers* ou VMs com imagens de reais desses elementos, isto é, é um ambiente que permite uma emulação muito próxima à situação real sem requerer os elementos físicos de hardware. De fato, o GNS3 é utilizado por grandes corporações como AT&T, Google, Chevron, NASA, Intel, IBM, Verizon, Chevron, dentre outras (GALAXY TECHNOLOGIES LLC, 2020).

Dentro do GNS3 utilizou-se também o analisador de protocolos *Wireshark* versão 2.6.10 para análise dos protocolos e avaliação da efetividade das *flow rules* implementadas.

#### 4.1.1.3 SONAr Server

O conjunto SONAr *Server* é constituído de todos os componentes do sistema SONAr e o controlador SDN, conforme descritos na Seção 2.1.7 e Capítulo 3. Ele foi implementado no servidor já mencionado com VM *VirtualBox* 6.0, sobre um SO Ubuntu Server 16.04 e desenvolvido utilizando-se *Java Development Kit* (JDK) 1.8.0-151. Utiliza também net-snmp 5.7.3 e lldpd 1.0.3 como soluções de protocolos padrões e Docker 18.06.2-ce para solução de *containers*. Os componentes foram implementados com as seguintes ferramentas:

- ❏ Controlador SDN (SDNC): ONOS 2.1.0
- ❏ NDB: Cassandra 3.11.4
- ❏ NEM: RabbitMQ 3.8.0
- ❏ AP: DHCP customizado para interação com os demais componentes
- ❏ SOEs, SLEs e CoEs (incluindo SCE e SSCM): Java 8 e Spring Boot 2.1.2

O ONOS já havia sido a opção de controlador escolhido para suportar a arquitetura SONAr devido a várias características de estabilidade, documentação e pelo fato de ser um projeto *open source*, distribuído sob a licença Apache 2.0 (ONF, 2020a).

Esse conjunto de componentes na configuração indicada foi denominado SONAr 0.1.0 e constitui a primeira versão funcional desse sistema.

#### 4.1.1.4 Elementos de Borda

Os elementos de borda, tanto a eNodeB quanto os demais elementos de *core* de rede (MME, SAE-GW, *sync server* e *management server*), que serão apresentados com detalhes na Seção 4.1.2, foram implementados utilizando-se a solução de *containers* Docker

18.06.2-ce, com SO Ubuntu 19.04, e com aplicativos net-snmp 5.7.3, lldpd 1.0.3, net-tools, tcpdump, telnet, traceroute, curl, iperf4 e nmap.

Uma vez que o objetivo não é avaliar o funcionamento de uma rede celular em si, mas apenas o da rede que suporta suas conexões fim-a-fim e suas particularidades, não foram implementadas aplicações específicas destes elementos de redes móveis nos servidores. Essas aplicações já estão disponíveis em licença aberta nos projetos *O-RAN Alliance* (ORAN ALLIANCE, 2019) e *NextEPC*, (NEXTEPC INC., 2020), porém suas implantações implicariam em não só um grau de complexidade muito maior, mas também na necessidade de um ambiente muito mais robusto para suportar aplicações RAN e *core*. A conexão pôde ser testada apenas utilizando-se os aplicativos acima citados, sem a utilização de elementos da rede celular em si.

#### 4.1.1.5 Open vSwitch

Os principais elementos da rede a ser configurada automaticamente são as switches SDN *Open vSwitch*, e que já foram apresentadas na Subseção 2.1.4. As switches utilizadas na montagem da rede no GNS3 foram baseadas na versão OvS 2.10.1 e implementadas sobre *containers* Docker 18.06.2-ce, com SO Linux Alpine 3.9. Além disso, foram instalados net-snmp 5.7.3, lldpd 1.0.3 e isc-dhclient 4.4.1 para suporte dos protocolos de rede utilizados.

#### 4.1.1.6 Roteadores *Appliance*

Para melhor simulação de uma rede real, foram utilizados também roteadores do tipo *appliance*, ou seja, elementos que não suportam o padrão SDN ou os protocolos associados ao mesmo, como o OpenFlow. Conforme será mostrado na Seção 4.1.2, esses roteadores foram utilizados como *gateway* para acesso a outros domínios de rede e possibilitar um cenário mais próximo ao hoje usando no contexto de telecomunicações.

Para implementação desses elementos, foi utilizada a imagem do roteador Cisco 7206VXR (rodando sobre Dynamips versão 0.2.20) com *engines* de processamento de rede NPE-400 (*revision* A) e 512 MiB e 512 KiB de memória RAM e NVRAM, respectivamente, CPU R7000 operando a 150MHz, *Implementation* 39, Rev 2.1, cache 256KB L2 e 512KB L3.

Ambos os roteadores utilizados na topologia foram virtualmente equipados com uma interface *FastEthernet* e duas interfaces *Gigabit Ethernet*, e usaram o SO Cisco IOS 7200 (C7200-ADVIPSERVICESK9-M), versão 12.4(24)T8, *Release Software* (fc1).

### 4.1.2 Cenário Avaliado

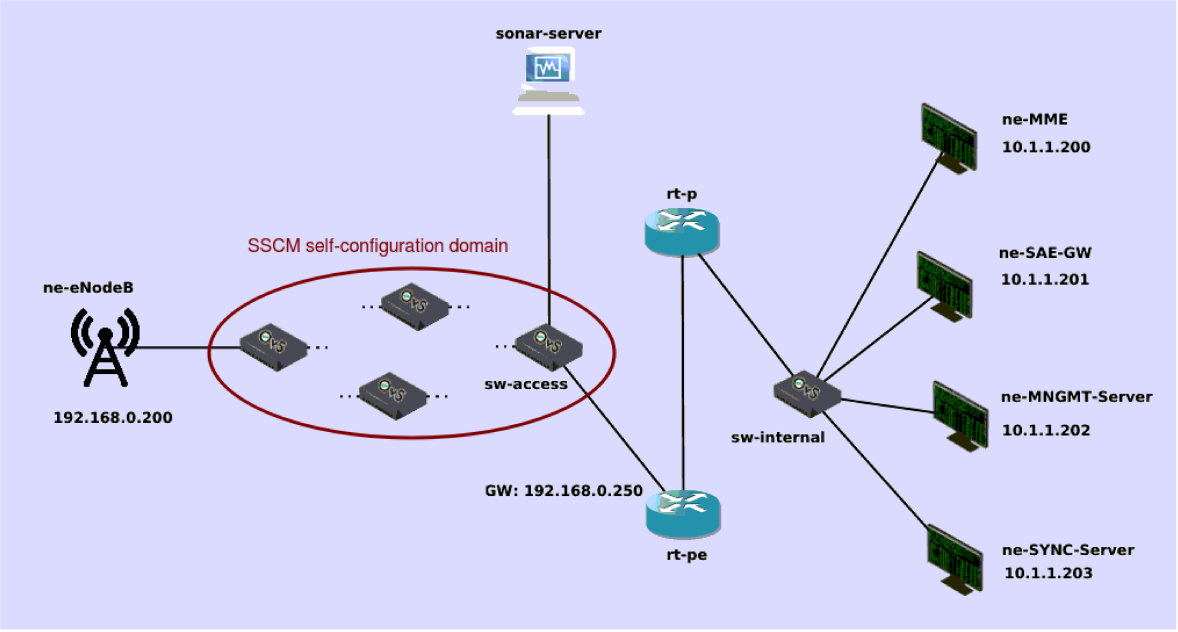
Para avaliação da efetividade e eficiência da solução de configuração especializada, foi escolhido o cenário típico de uma rede celular 4G. Embora o módulo de configuração

especializada possa ser utilizável no caso geral, um ambiente de redes móveis foi preferido devido à grande quantidade de elementos de borda que são normalmente implementados, em especial quando considera-se a demanda incipiente motivada pelo 5G. No entanto, o caso de uso aplica-se para diferentes tipos de elementos de borda, como a já citada OLT, o DSLAM, ou mesmo para as demais gerações de ERBs, como as BTSs (2G), *NodeB* (3G), *eNodeB* (4G) ou *gNodeB* (5G).

O caso 4G foi preferido em detrimento ao caso 5G por questões práticas: no atual estágio de implementação, o 5G ainda opera em modo *non-standalone*, ou seja, a *gNodeB* 5G carrega apenas o tráfego de *user plane* (dados efetivamente enviados e recebidos pela aplicação do usuário), enquanto ainda depende de uma *eNodeB* 4G em operação conjunta – e em outra frequência – para carregar o tráfego de sinalização, gerência e sincronismo. Dessa forma, o caso 5G seria equivalente à configuração especializada de dois elementos de rede distintos e mais simples – *eNodeB* e *gNodeB* –, exigindo dois SSCDs distintos. Deste modo, a demonstração do caso 4G torna-se ao mesmo tempo mais completa (por contemplar todos os planos de usuário e controle) e mais simples (por exigir a auto-configuração de um único elemento) no contexto do trabalho.

A Figura 15 mostra a topologia básica usada nos experimentos, e as Tabelas 5 e 6 mostram as configurações usadas.

Figura 15 – Topologia básica para experimentos.




Fonte: elaborada pelo autor

Essa topologia foi retirada de um exemplo real de uma operadora de redes móvel e fixa. O domínio do *self-configuration* é restrito às denominadas redes de acesso: normalmente são redes que operam em *layer* 2 ou MPLS que são distribuídas pela área de atendimento da operadora e nas quais são conectados os elementos de borda (*edge* NEs) e fazem a

Tabela 5 – Parâmetros de rede da topologia usada nos experimentos.

| **Element** | **Network** | **Gateway** | **Broadcast** | **eth0** | **eth1** | **eth2** |
|---|---|---|---|---|---|---|
| sw-access | 192.168.1.0/24 | 192.168.1.250 | 192.168.1.255 | - | - | - |
| sw-1 (Metro Ethernet) | 192.168.1.0/24 | 192.168.1.250 | 192.168.1.255 | - | - | - |
| ... | 192.168.1.0/24 | 192.168.1.250 | 192.168.1.255 | - | - | - |
| sw-n (Metro Ethernet) | 192.168.1.0/24 | 192.168.1.250 | 192.168.1.255 | - | - | - |
| sw-internal | 10.1.1.0/24 | 10.1.1.250 | 10.1.1.255 | - | - | - |
| rt-pe (Provider Edge) | - | - | - | - | 192.168.1.250 | 10.1.2.200 |
| rt-p (Provider) | - | - | - | - | 10.1.2.201 | 10.1.1.250 |
| eNodeB | 192.168.1.0/24 | 192.168.1.250 | 192.168.1.255 | 192.168.1.200 | - | - |
| ne-MME | 10.1.1.0/24 | 10.1.1.250 | 10.1.1.255 | 10.1.1.200 | - | - |
| ne-SAE-GW | 10.1.1.0/24 | 10.1.1.250 | 10.1.1.255 | 10.1.1.201 | - | - |
| ne-MNGMT-Server | 10.1.1.0/24 | 10.1.1.250 | 10.1.1.255 | 10.1.1.202 | - | - |
| ne-SYNC-Server | 10.1.1.0/24 | 10.1.1.250 | 10.1.1.255 | 10.1.1.203 | - | - |

Fonte: elaborada pelo autor

Tabela 6 – Parâmetros de VLANs na topologia usada nos experimentos.

| **VLAN** | **Description** | **nw_dst (IP)** | **VLAN ID** | **CoS/PCP** | **Type** | **Description** |
|---|---|---|---|---|---|---|
| VLAN CP | Control plane or MME (C-Plane) | 10.1.1.200 | 3661 | 6 | IN | Internetwork Control |
| VLAN UP | User/Data plane or SAE-GW (U-Plane) | 10.1.1.201 | 3660 | 1 | BE | Best Effort |
| VLAN MNG | Management or NE-Management (M-Plane) | 10.1.1.202 | 3662 | 2 | EE | Excellent Effort |
| VLAN SYNC | Sync or ClockServer (S-Plane) | 10.1.1.203 | 3663 | 7 | NC | Network Control |

Fonte: elaborada pelo autor

conexão com o *core* da rede, onde situam-se os elementos que atuam como *gateways* para o mundo externo, fazem o controle, sincronismo e gerência da rede, além de outras plataformas, como *caching*, *media gateways* VoIP, etc.

Essas redes de acesso conectam-se ao *core* de rede através de um switch de acesso que, por sua vez, estabelece a conexão com o roteador denominado *Provider Edge* (PE). Os roteadores do tipo PE são interconectados entre si e também conectam-se aos roteadores do tipo *Provider* (P). A função dos roteadores do tipo P é de estabelecer a interface diretamente com os elementos do *core* da rede.

Por fim, os elementos internos do *core* são conectados aos roteadores P por switches internas, que distribuem os pacotes aos elementos finais.

Desta forma, para esta topologia simplificada, tem-se três domínios de rede distintos: a rede de acesso (192.168.0.0/24), uma rede intermediária para transporte (10.1.2.0/24) e uma rede interna de *core* (10.1.1.0/24), sendo que, na implementação de um elemento de borda, seja uma nova eNodeB ou um novo elemento de acesso OLT, apenas a rede de acesso é configurada pelos engenheiros. As redes intermediárias e interna, por motivo de segurança, não podem sofrer alterações com a implementação de novos elementos de acesso. Desta forma, na prática, apenas os elementos da rede de acesso são configurados pelos engenheiros de rede para receber um novo elemento de borda, sendo este cenário o escolhido para a demonstração da funcionalidade de *self-configuration* aqui apresentada.

Na rede de acesso, os pacotes são isolados em VLANs distintas, e a cada VLAN *tag*, é associada uma classe de serviço específica. No caso das redes móveis, dentro do caso

levantado e escolhido, são configuradas quatro VLANs para quatro tipos de pacotes diferentes, identificados pelo seu elemento de destino, conforme já foi explicado na Subseção 2.1.8. Isso garante que pacotes de sincronismo e controle tenham prioridade no tráfego da rede de acesso, pois são mais sensíveis a latência e perda de pacotes. A relação entre tipos de pacotes com sua prioridade, ou CoS, foi mostrada na Tabela 6. Ao entrarem na rede de acesso, tanto do lado do elemento de borda (eNodeB) quanto do lado do *core* da rede, uma VLAN *tag* deve ser inserida, juntamente com o CoS associado e, em sua saída, essa *tag* deve ser retirada.

Reforça-se aqui que tanto a determinação da topologia de rede quanto as ações realizadas para configurar-se a mesma em resposta à inclusão de um novo elemento de borda foram baseados no caso real de uma operadora de serviços fixos e móveis. Este caso-base foi levantado através de diversas entrevistas com engenheiros desta operadora e estudos das topologias usadas na prática. Embora apresentado de uma maneira simplificada, o caso adequa-se perfeitamente ao uso na rede da operadora em questão e também deve adequar-se a outros casos distintos, observando-se que as condições descritas na Seção 3.1 tenham sido satisfeitas.

## 4.2 Experimentos

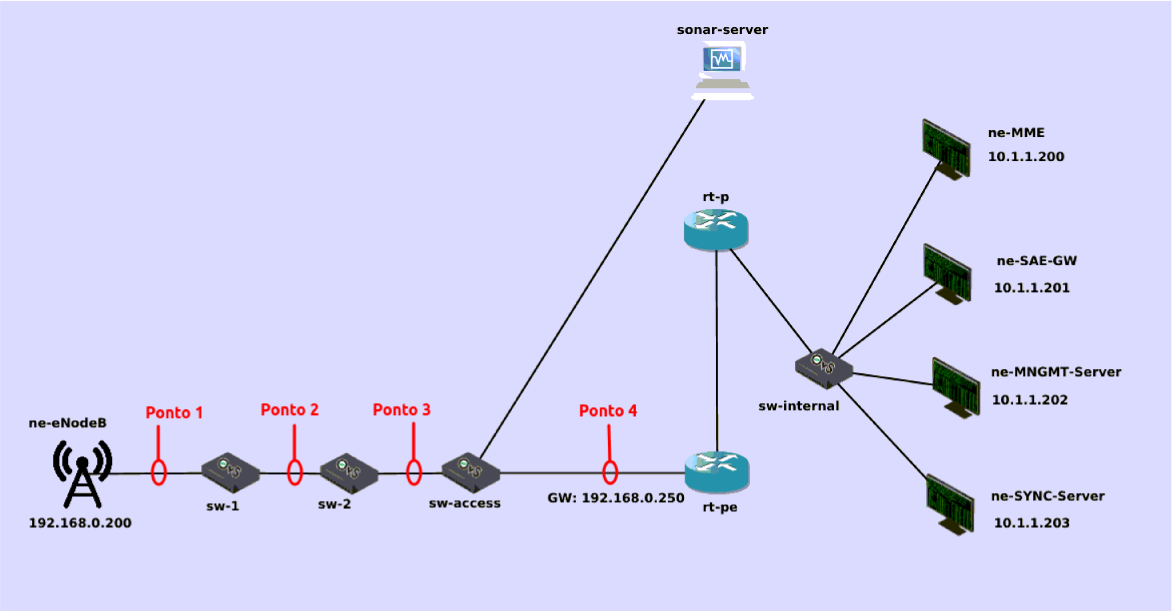
Nesta seção serão descritos os experimentos realizados no intuito de avaliar na prática a eficácia e eficiência da solução proposta para auto-configuração especializada de rede. Os experimentos são divididos em três etapas: a Subseção 4.2.1 expõe a comprovação do funcionamento em um cenário mais básico de rede, a Subseção 4.2.2 aborda cenários mais complexos, a relação desta complexidade com o tempo de convergência da solução e o impacto no tempo total de *bootstrapping* da rede e, por fim, a Subseção 4.2.3 descreve o comportamento em topologias com mais de um caminho disponível para o roteamento dos pacotes.

### 4.2.1 Experimento 1 - Efetividade da Solução

O primeiro experimento realizado foi a de comprovação prática do funcionamento correto da solução proposta para a auto-configuração especializada de rede. Não foi objetivo nesta etapa o de avaliar a performance do SSCM, o que será feito em na Subseção 4.2.2, mas apenas o de verificar se, uma vez que um elemento de borda é inserido na rede sob o domínio do SONAr e havendo um SSCD correspondente a este elemento no NDB, a rede será configurada com sucesso de acordo com o descrito no SSCD e se os pacotes que trafegam na rede de acesso estão corretamente marcados.

Para isso, foi utilizada a topologia mostrada na Figura 16 implementada no GNS3 conforme descrito na Subseção 4.1.1, com três switches na rede de acesso. Este é o caso mais simples que permite uma investigação completa do funcionamento do sistema.

Figura 16 – Topologia para experimento 1.




Fonte: elaborada pelo autor

A investigação foi realizada em duas etapas diferentes: a primeira, com a topologia conectada e ligada, porém antes de se conectar o servidor SONAr à mesma, isto é, a rede em estado *default*. Para que houvesse condição de testes nesta etapa, um IP foi manualmente atribuído à eNodeB, uma vez que o AP não está em funcionamento e não consegue responder ao DHCP *discovery*. A segunda etapa foi a conexão do servidor SONAr à rede e, após o processo de *self-configuration* completo, com a rede em estado *started*. Para cada uma destas etapas, foram verificados:

- ❏ *Flow rules* instaladas nas switches sw-1, sw-2 e sw-access;
- ❏ Solicitação e resposta do *ping* – que usa o protocolo *Internet Control Message Protocol* (ICMP) – da eNodeB até cada um dos elementos de core de rede;
- ❏ Análise dos protocolos L2 nos pacotes referentes ao ICMP *echo request* e ICMP *echo reply* nos pontos 1, 2, 3 e 4, mostrados na Figura 16.

Este cenário foi implementado e executado e os resultados serão apresentados a seguir.

#### 4.2.1.1 Flow Rules Instaladas

Com os elementos ligados e ainda sem a execução dos processos SONAr, as *flow rules* observadas foram as padrões quando as switches OvSs são ativadas e são listadas no anexo A.1, por questões de organização e espaço. Logo após, o SSCD a seguir foi incluído na NDB:

```
{
   "id":{
      "mac_address":"62:05:4b:7e:76:b2",
      "description":"eNodeB-ULA-Jaragua",
      "responsible":"Daniel Ricardo",
      "ip":"192.168.0.200"
   },
   "config_end_node":{
      "mac_address":"be:d9:04:08:4e:a6",
      "ip":"192.168.0.250",
      "mask_bits":24,
      "config_end_node_description":"PE-Router (Gateway)"
   },
   "dst_node":[
      {
         "description":"MME (C-Plane)",
         "ip":"10.1.1.200",
         "mask_bits":24,
         "priority":5000,
         "vlan":3661,
         "pcp_cos":6
      },
      {
         "description":"SAE-GW (U-Plane)",
         "ip":"10.1.1.201",
         "mask_bits":24,
         "priority":5000,
         "vlan":3660,
         "pcp_cos":1
      },
      {
         "description":"NE-MNGT (M-Plane)",
         "ip":"10.1.1.202",
         "mask_bits":24,
         "priority":5000,
         "vlan":3662,
         "pcp_cos":2
      },
      {
         "description":"SYNC (S-Plane)",
         "ip":"10.1.1.203",
         "mask_bits":24,
         "priority":5000,
         "vlan":3663,
         "pcp_cos":7
      }
   ]
```

```
}
```

Então o SONAr *server* foi ativado e a rede foi configurada automaticamente com os processos já mencionados. Repare-se que, no processo de *self-configuration*, o SONAr apaga as *flow rules* padrões que haviam sido automaticamente geradas pelas OvSs por questão de consistência e segurança. Ao final da convergência ds processo de *self-configuration*, observou-se as *flow rules* instaladas nas switches mostradas no anexo A.2.

Conforme esperado, as *flow rules* instaladas fazem com que os pacotes sejam marcados na switch sw-1 com as VLANs e prioridades específicas de acordo com cada elemento de destino e então encaminhadas para a switch sw-2. No sentido inverso, são retirados os cabeçalhos 802.1Q (com a ação *strip_vlan*) dos pacotes recebidos da switch sw-2 e então encaminhados para a eNodeB. O mesmo acontece, em ordem inversa, na switch sw-access.

Por fim, a switch sw-2 simplesmente encaminha os pacotes para a conforme os critérios de IP de origem e destino dos mesmos, sem alterar as marcações VLAN.

Repara-se também que as *flow rules* instaladas pelo SSCM têm uma prioridade 5.000, ao contrário das *flow rules* de propósitos gerais (ARP, LLDP, comunicação com controlador e outras switches, etc), instaladas pelo módulo de *plug-and-play* do SCE, que têm uma prioridade *default* 4.000. Isso faz com que os pacotes com origem e destino indicados no SSCD obedeçam prioritariamente as *flow rules* especializadas, instaladas pelo SSCM, caso haja mais de uma regra com critérios satisfeitos.

#### 4.2.1.2 Efetividade da Conexão

Também antes e após a execução da *self-configuration*, foram realizados testes de *ping* para testar-se a conectividade entre a eNodeB, os elementos de *core* e o SONAr *server* em si.

Uma vez que antes da *self-configuration* as *flow rules* padrões ("*actions=NORMAL*") mostradas no anexo A.1 estavam instaladas nas switches, esperava-se que efetivamente houvesse conectividade entre os elementos, o que foi demonstrado nas listagens mostradas no anexo B.1.

Após a *self-configuration* era também necessário confirmar que a conectividade manteria-se efetiva com as novas *flow rules* instaladas, uma vez que as *default* foram removidas pelo SONAr. A listagem mostrada no anexo B.2 demonstra que a conectividade se manteve e os pacotes conseguiram percorrer os caminhos de ida e volta na rede utilizando as novas *flow rules* mais específicas e apropriadas.

#### 4.2.1.3 Análise dos Pacotes

Por fim, resta a análise da efetividade das *flow rules* na inserção dos cabeçalhos 802.1Q nos pacotes enviados da eNodeB pela switch sw-1 e retirada dos mesmos após a saída da

switch sw-access para o *core* da rede. Isso foi feito abrindo-se os protocolos de camada 2 nos pontos 1, 2, 3 e 4 da rede, conforme indicado na Figura 16 com o aplicativo *Wireshark*.

Na etapa antes da *self-configuration*, os pacotes foram encaminhados normalmente, porém sem qualquer marcação ou separação por VLANs, pois as *flow rules* instaladas (anexo A.1) não contém nenhuma instrução neste sentido.

Na etapa após a *self-configuration*, o tráfego de ICMP *request* foi coletado e analisado nos pontos indicados (interfaces entre as switches). O resultado é mostrado na Figura 17, que contém as saídas do *Wireshark* para cada ponto analisado.

Neste experimento, foi enviado um *ping* com múltiplos ICMP *requests* e *replies* da ne-eNodeB (192.168.1.200) para o servidor de sincronismo (ne-SYNC-Server). Pela Tabela 6, percebe-se que a VLAN para esse elemento em específico possui uma VLAN Id 3663 e o tipo de tráfego é *Network Control*, especificado pelo *Priority Point Code* (PCP) 7.

No ponto 1, entre a eNodeB e a switch sw-1, verifica-se que não há marcação do pacote. De fato, nos elementos de rede reais, essa marcação normalmente não é feita no elemento de borda ou, caso seja, é considerada uma boa prática a retirada da marcação e inclusão de uma nova marcação pela switch na qual o elemento conecta-se à rede. Esta prática é para que os próprios engenheiros de rede de acesso possam garantir que essa marcação será feita de forma correta, não precisando confiar na configuração realizada por outros profissionais no elemento de borda.

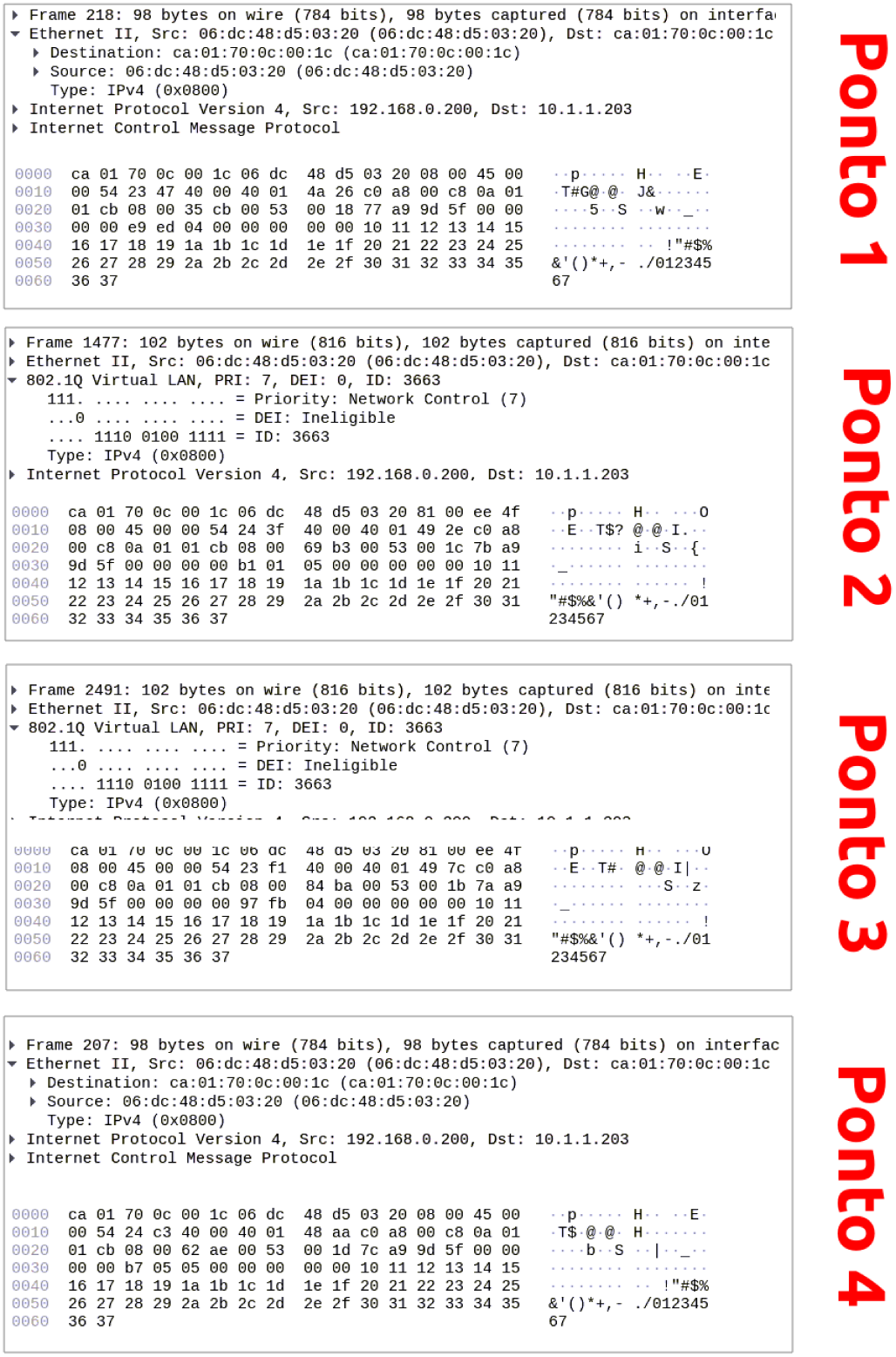
No ponto 2, já nota-se o cabeçalho 802.1Q, marcando o pacote corretamente com VLAN Id 3663 e PCP 7. O mesmo acontece também no ponto 3, entre a switch sw-2 e a switch sw-access. Essas marcações correspondem às descritas na Tabela 6 para pacotes com origem ou destino do ne-SYNC-Server.

Por fim, no ponto 4, percebe-se que já não há um cabeçalho 802.1Q: ele foi retirado por essa switch, conforme *flow rule* específica observada no anexo A.2. A retirada deste cabeçalho é importante para manter-se o pacote em sua forma original quando o mesmo sair do domínio de rede SONAr, garantido-se assim que a integridade das configurações nas redes legadas ou não pertencentes ao domínio SONAr seja mantida.

#### 4.2.1.4 Conclusões do Experimento 1

De acordo com o que foi mostrado nesta subseção, verifica-se a efetividade do módulo SSCM na configuração especializada de uma rede com três switches. Embora, por razões de simplificação não seja monstrado aqui, os cenários com várias switches (até a quantidade de 18, conforme irá ser demonstrado na Subseção 4.2.2) também foram investigados e o mesmo comportamento foi evidenciado nos experimentos. Logo, assume-se que o algoritmo utilizado aplica-se satisfatoriamente para quaisquer quantidades de switches na rede em questão.

Figura 17 – Análise de camada 2 nos quatro pontos da rede.

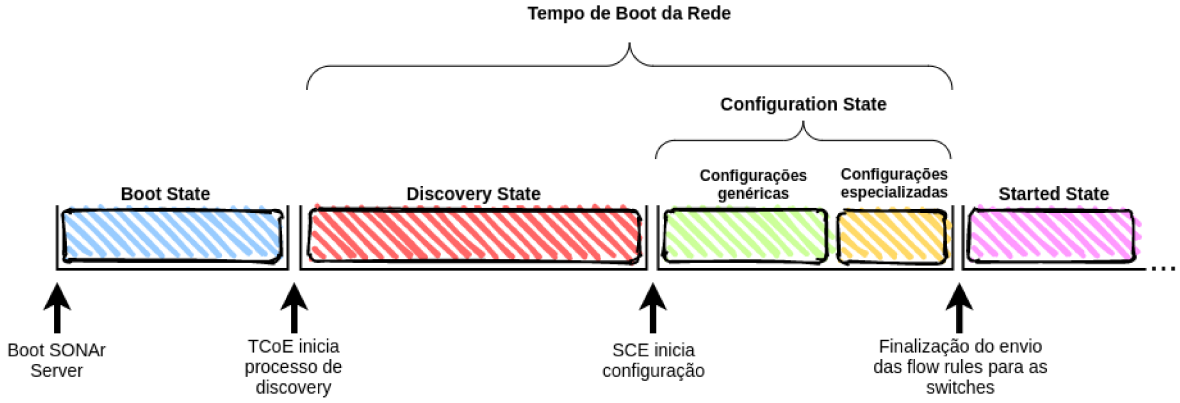



Fonte: elaborada pelo autor

### 4.2.2 Experimento 2 - Tempo de Convergência do SSCM e Impacto no Bootstrapping

Nesta subseção, serão discutidos os experimentos que avaliaram o comportamento do SSCM em redes com diferentes quantidades de elementos na rede de acesso. Especificamente, foram avaliados os cenários com 3, 6, 9, 12, 15 e 18 switches em linha, de forma similar à topologia usada em 4.2.1. Todos esses cenários também foram simulados partindo-se do *bootstrapping* da rede para que seja avaliado o impacto do SSCM no tempo total de convergência, iniciando do *discovery* até a configuração final da rede. A Figura 18 mostra de maneira simplificada as etapas de *bootstrapping* e *self-configuration* da rede.

Figura 18 – Processo de bootstrapping da rede.




Fonte: elaborada pelo autor

O primeiro estado (*boot state*) representa o tempo em que os próprios componentes SONAr estão sendo ativados. Este estado é finalizado com o início do processo de *discovery* da rede pelo TCoE, onde essa entidade irá descobrir os elementos no domínio do SONAr, estabelecer uma topologia na forma de um grafo, gravar esta topologia no NDB e notificar aos demais componentes da finalização do processo publicando um evento no NEM. Esse estado é denominado *discovery state*.

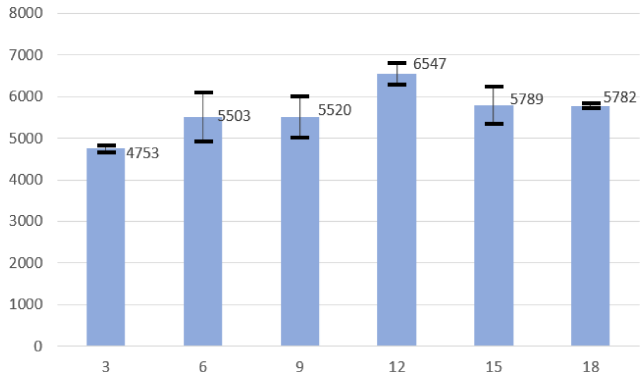
A etapa de configuração, que caracteriza o *configuration state*, inicia quando o SCE recebe o evento de finalização do *discovery* e inicia o processo de configuração, que envolve o cálculo do menor caminho entre o servidor SONAr e os demais componentes da rede e a geração das *flow rules* que permitirão a comunicação dos elementos através dos protocolos de controle e sinalização (ARP, DHCP, LLDP, etc). Finalizada esta sub-etapa (denominada configurações genéricas), uma outra sub-etapa inicia-se, caso haja elementos descobertos com um SSCD correspondente no NDB: as configurações especializadas através do SSCM, conforme já visto em 3.2.1. O *configuration state* termina com o envio das *flow rules* para os elementos via SBI, e a rede entra no *started state*.

Para este experimento, cada cenário foi simulado cinco vezes na modalidade de *bootstrapping* e os tempos de cada etapa mostrada na Figura 18 foram marcados em mi-

lissegundos por uma classe específica, denominada *SONAr Results Extractor* (SRE), que registra os intervalos entre os eventos recebidos no NEM. Com isso, tem-se uma medição bem precisa da duração dos diversos estados da rede. Após a medição dos tempos, foram calculadas as médias das amostras de cada intervalo e para cada cenário, e então determinados os intervalos de confiança com níveis de 95%. Os gráficos apresentados mostram a média junto com os limites inferior e superior desse intervalo.

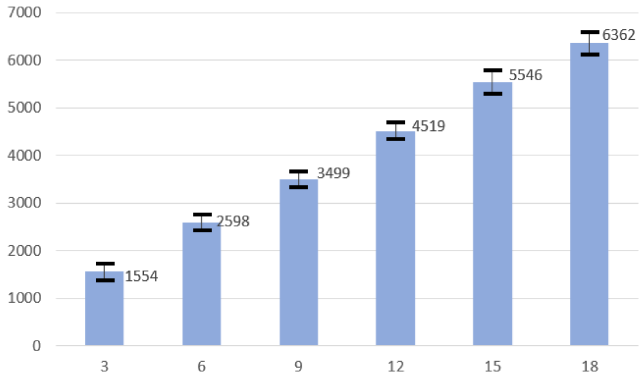
A Figura 19 representa o tempo de convergência do SSCM, em milissegundos, enquanto as Figuras 20, 21 e 22 representam, respectivamente, o tempo de *discovery* da rede (*discovery state*), o tempo de configurações genéricas e o tempo total de *boot* da rede, todos esses também em milissegundos. Os valores indicados nas barras em todos os gráficos correspondem à média das amostras, enquanto que os valores dos limites inferior e superior dos intervalos de confiança não são mostrados.

Figura 19 – Tempo de convergência (em ms) do SSCM com relação à quantidade de switches.



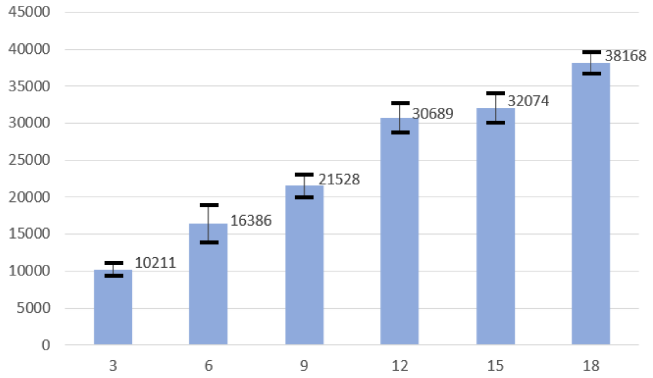

Fonte: elaborada pelo autor

Figura 20 – Tempo de discovery de rede (em ms) com relação à quantidade de switches.



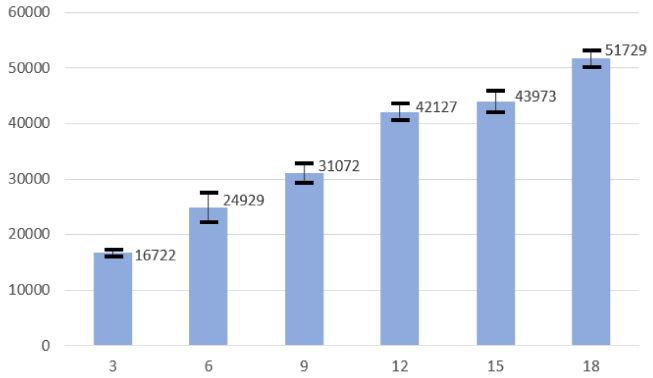

Fonte: elaborada pelo autor

Figura 21 – Tempo de configuração genérica (em ms) com relação à quantidade de switches.




Fonte: elaborada pelo autor

Figura 22 – Tempo total de boot da rede (em ms) com relação à quantidade de switches.




Fonte: elaborada pelo autor

Observando-se somente a Figura 19, verifica-se que o tempo de convergência do SSCM não varia substancialmente com relação à quantidade de switches que estão no caminho entre o elemento de borda e o roteador *provider edge*. Isso explica-se pelo fato de, quando a etapa de configuração especializada inicia-se, todas os elementos já foram identificados, todas as rotas já foram calculadas e todos as configurações genéricas já foram realizadas, de forma que o *SONAr* já possui pleno conhecimento da estrutura da rede. Há, naturalmente, um aumento no tempo envolvido no cálculo das *flow rules* específicas (quanto maior a quantidade de elementos, maior o tempo deste cálculo), porém a duração desse processo é muito pequena quando comparado ao tempo total de *self-configuration*, ficando na casa das dezenas de milissegundos. Desta forma, embora haja efetivamente uma correlação da quantidade de switches em linha com o tempo de finalização do processo SSCM, em casos práticos esta correlação é fraca e não evidente, como mostrado no experimento. Em uma configuração com três switches, o tempo médio de convergência observado foi de 4.753 ms ± 143 ms, enquanto que o mesmo tempo para 18 switches foi de 5.782 ms ± 96 ms,

ambos considerando intervalo de confiança de 95%.

Já nos tempos de *discovery* e de configuração genérica da rede esta relação é bem mais óbvia. Em ambos os casos, tanto o *discovery* quanto a configuração são feitos elemento por elemento em sequência, ou seja, é necessário que um esteja efetuado para que o outro se inicie. Isso faz com que haja uma relação direta e visualmente evidente entre o tempo de convergência de ambas as etapas com a quantidade de elementos no caminho entre o início e fim da rede.

A soma das durações das etapas de *discovery* e configurações genéricas e especializadas são aproximadamente iguais ao tempo medido total de *boot* da rede para cada caso, conforme mostrado na Figura 22. As somas dos tempos são, na verdade, ligeiramente menores do que o tempo efetivamente medido para o *boot*, devido ao fato de todo o processo ser assíncrono e orientado a eventos. A diferença corresponde sobretudo ao atraso dos envios, publicações e recebimento dos eventos no NEM.

De uma forma geral, o tempo do processo de configuração especializada – o que efetivamente está sendo avaliado neste estudo – ficou entre 4 e 7 segundos para todas as amostras obtidas, de forma que a resposta do SONAr à inclusão de um elemento de borda na rede é suficientemente rápida para utilização na prática, superando em várias ordens de grandeza o tempo de configuração manual de uma rede, mesmo utilizando-se processos semi-automatizados (como a configuração com *scripts*, por exemplo). Este processo em específico correspondeu, no pior caso (3 switches), a 28,4% do tempo total de *bootstrapping* e *self-configuration* e, no melhor caso (18 switches), a 11,2% do mesmo tempo.

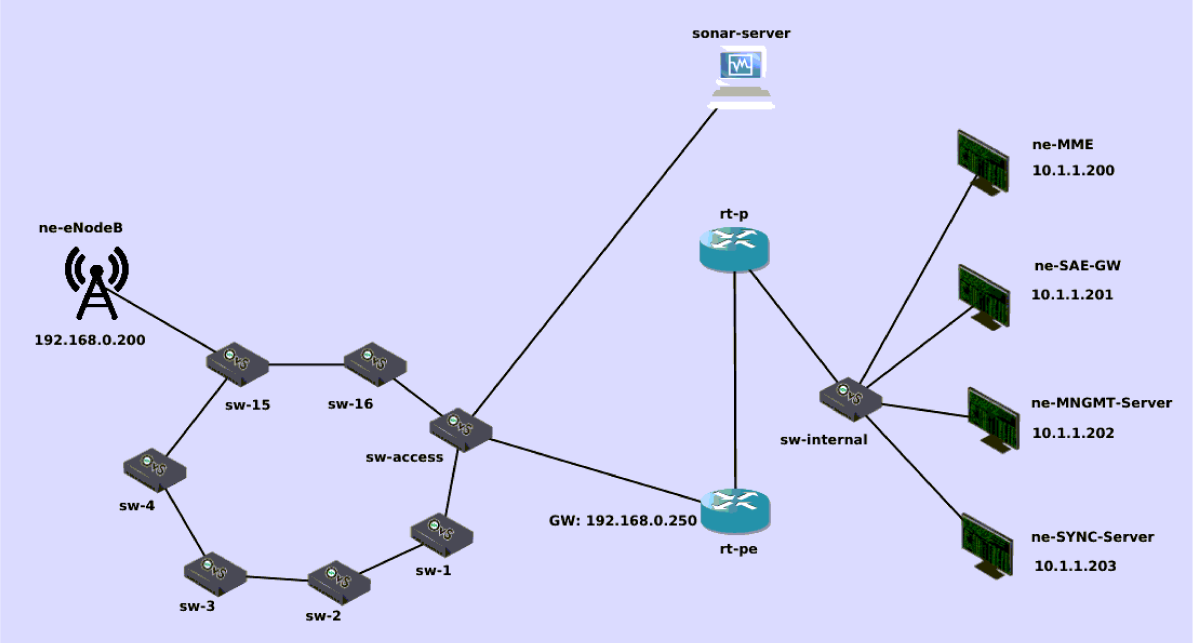
### 4.2.3 Experimento 3 - Efetividade em Topologia em Anel

Para topologia em anel, foi habilitado o protocolo 802.1w, denominado *Rapid Spanning Tree Protocol* (RSTP) – que é uma melhoria do protocolo 802.1D *Spanning Tree Protocol* (STP) – nas switches OvSs. Os experimentos de efetividade em topologia em anel foram realizados utilizando a configuração mostrada na Figura 23.

O protocolo RSTP atua logo que a rede é iniciada e seu tempo de resposta é rápido, correspondendo a três vezes o tempo de *Hello*, que, por default, é de dois segundos (CISCO SYSTEMS, 2017), resultando em um tempo de convergência típico de seis segundos. Seu funcionamento baseia-se na desabilitação das portas que não estão no caminho definido para o roteamento do pacote, ficando estas portas nos estados "*Discarding*" ou "*Learning*" (CISCO SYSTEMS, 2017), processo que ocorre *antes* do *bootstrap* da rede. Em ambos os casos de estados das portas, os caminhos que utilizam as mesmas não são considerados no processo de *bootstrap* e o caso da topologia em anel resume-se ao de topologia em linha, similares aos abordados nos experimentos 1 e 2.

Nos experimentos em topologia em anel, o caminho definido foi o ne-eNodeB ↔ sw-15 ↔ sw-16 ↔ sw-access ↔ rt-pe, tornando-se o caso de três switches em linha similar

Figura 23 – Topologia para experimentos em anel.




Fonte: elaborada pelo autor

ao simulado no Experimento 1 (Seção 4.2.1). De fato, os tempos de convergência do SSCM observados nos experimentos em topologia em anel foram similares ao caso de três switches em linha e ficaram próximos ao intervalo observado na Figura 19 em 4.688 ms ± 112 ms (média de cinco amostras considerando intervalo de confiança de 95%).

Uma importante observação é referente ao caso da falha na rota ativa: no protocolo RSTP a rota é rapidamente comutada para uma rota alternativa em caso de falhas. O escopo deste trabalho não prevê a reconfiguração especializada desta rota alternativa, pois este desenvolvimento envolveria processos das entidades de *self-healing* (SHE), que não fazem parte desta investigação. O desenvolvimento e implementação de uma técnica para reconfiguração de rotas alternativas fica em aberto e como sugestão de tema para trabalhos futuros.

## 4.3 Avaliação dos Resultados

Após a realização dos experimentos e análise dos resultados, constatou-se a efetividade do módulo SSCM dentro da arquitetura SONAr. Também constatou-se que o tempo de convergência da solução é bastante adequado às aplicações práticas e o impacto do SSCM no desempenho geral no processo de *self-configuration* varia com relação à quantidade de switches, sendo menos perceptível quanto maior for a quantidade de switches que serão configuradas (uma vez que o tempo de convergência do SSCM aumenta muito pouco com relação a essa quantidade). Além disso, mostrou-se que, uma vez que o

próprio SONAr estabelece uma rota entre o elemento de borda e o elemento final do domínio SSCM (com auxílio do protocolo RSTP), as funcionlidades de *self-configuration* genérica e especializada mantém-se mesmo em redes com caminhos redundantes, no caso de topologias envolvendo anéis.

Ainda assim, algumas limitações aplicam-se ao SSCM. A primeira delas é a necessidade de ter-se pontos bem definidos e únicos de início e final da configuração, conforme já exposto na Seção 3.1. A solução não adequa-se a redes que necessitem a funcionalidade de *self-configuration* com mais de uma ramificação inicial ou final. Porém, conforme foi determinado através de entrevistas e estudos em campo, este requerimento não impacta na aplicabilidade em casos reais, uma vez que as topologias utilizadas na prática normalmente convergem no lado do *core* de rede para um *gateway* ou switch concentrador.

Outra limitação também já abordada é a impossibilidade de reconfiguração da rede em casos de falhas, o que não foi tratado no escopo deste trabalho. Uma melhor investigação neste aspecto deve ser realizada com o envolvimento de outras funcionalidades, em especial a de *self-healing*. Na implementação das demais funcionalidades (*self-optimization* e *self-protection*), o processo de SSCM também deverá ser tratado.

Finalmente, em comparação com os trabalhos já desenvolvidos no mesmo tema e mostrados na Tabela 2, pode-se incluir o projeto SONAr SSCM na mesma análise, conforme apresentado na Tabela 7. Nela percebe-se a aderência da abordagem adotada com os as principais características relevantes levantadas neste estudo.

Tabela 7 – Comparativo das características entre os trabalhos correlatos e solução SONAr SSCM.

| **Característica** | **SON 3GPP** | **SELF-NET** | **CogNet** | **Superfluidity** | **Network Management Framework** | **PBNM** | **InitSDN** | **Auto-Config. of SDN Switches** | **SONAr** | **SONAr SSCM** |
|---|---|---|---|---|---|---|---|---|---|---|
| Está em um framework que engloba os demais itens de SON, além de auto-configuração? | Sim | Sim | Parc. | Parc. | Sim | - | - | - | Sim | Sim |
| Prevê auto-configuração de NEs SDN? | - | - | - | Parc. | Sim | - | Parc. | Sim | Sim | Sim |
| Prevê auto-configuração de NEs especializados? | Sim | - | - | Sim | - | - | - | - | - | Sim |
| Realiza auto-configuração em redes híbridas (SDN e não-SDN)? | Sim | - | - | - | - | - | Sim | Sim | Sim | Sim |
| Prevê coleta e manutenção de BD com mensagens do processo para histórico ou analytics? | Sim | Sim | Sim | N.I. | - | Sim | - | - | Sim | Sim |
| Propõe uma abordagem prática? | Sim | Sim | - | - | - | Sim | Sim | Sim | Sim | Sim |
| Apresenta resultados práticos? | Sim | Sim | - | - | - | Sim | - | - | Sim | Sim |

Legenda: SIM – Atende Plenamente ao quesito; Parc. – Atende o quesito parcialmente (de forma incompleta); N.I. – Não informado no trabalho; - – Não atende ao quesito

Fonte: elaborada pelo autor

Capítulo 5

# Conclusão

Os trabalhos na área de *Self-Organizing Networks* ainda são relativamente escassos, embora venham tornando-se cada vez mais evidentes com a adoção dos conceitos NFV e SDN na redes em produção. Especialmente na área de telecomunicações, percebe-se uma alta tendência em ambas as abordagens, bem evidente com a implantação das primeiras redes 5G no mundo.

À medida que aumenta-se a complexidade das redes, elas irão demandar um nível cada vez maior de automatização para responder de forma ágil às falhas nos planos de controle e dados, ataques cibernéticos, uso eficiente de seus recursos e necessidade de crescimento e configuração. Este último quesito foi a principal motivação deste trabalho.

O módulo de auto-configuração especializada – aqui denominado SSCM – foi a solução proposta para essa situação. Na Seção 1.1, foi declarado que o principal objetivo da pesquisa seria o de investigar as alternativas para um sistema de auto-configuração de rede para elementos com requerimentos específicos, juntamente com um modelamento para se descrever esses requerimentos. Durante a investigação, foram estudadas várias abordagens para este problema, sendo que o *framework* SONAr adequou-se muito bem ao propósito. Dentro dessa arquitetura, um novo módulo foi desenhado e implementado (o SSCM) e uma forma de descrever as especificidades da configuração da rede foi proposta (o formato SSCD).

## 5.1 Principais Contribuições

O desenvolvimento do trabalho aqui apresentado contribui para complementar as já importantes funcionalidades de *self-configuration* pré-existentes no SONAr, pois faz com que a arquitetura suporte agora essa funcionalidade para um contexto mais específico, conforme a realidade da indústria, sobretudo a de telecomunicações.

Conforme demonstraram os resultados dos experimentos, tanto em termos de efetividade quanto de eficiência, o módulo de auto-configuração especializada atingiu o desafio de que toda estrutura necessária particular a cada caso de elemento de borda seja con-

figurada sem intervenção humana e de forma mais rápida do que se fosse feita por um especialista – conforme havia sido colocado na Seção 1.1. Os resultados mostrados nas Subseções 4.2.1, 4.2.2 e 4.2.3 corrobam este atingimento, especialmente no quesito de rapidez de configuração, tendo sido registrados tempos muitíssimo menores do que os atingíveis com uma configuração manual ou semi-automatizada. A hipótese apresentada na Seção 1.2 provou-se verdadeira, sendo possível desenvolver-se um modelo funcional de *self-configuration* especializado mais eficiente em termos de agilidade de implantação do que a um procedimento não-automatizado.

O modelamento utilizado, principalmente o formato do SSCD, pode ser eventualmente aproveitado em outras abordagens que tenham o mesmo objetivo. O mesmo se aplica à forma com que o SSCM foi incluído no contexto da arquitetura SONAr: uma abordagem baseada em eventos mostrou-se não só muito eficaz, mas de uma grande versatilidade na inclusão de novos módulos à esta arquitetura, como foi demonstrado neste trabalho.

## 5.2 Trabalhos Futuros

A plataforma SONAr permite uma grande variedade de novos trabalhos, uma vez que a maior parte das SLEs ainda não foram totalmente desenvolvidas e as SOEs também podem ser revistas, melhoradas e ampliadas, como foi feito neste trabalho com a SCE. Porém, mesmo falando-se especificamente sobre o SSCM, pode-se identificar várias possibilidades de trabalhos futuros envolvendo, por exemplo:

- Desenvolvimento de alternativas para a reconfiguração automática em caso de falha de rede, integrando o SSCM à SHE. Conforme já mencionado, em caso de falha de uma rota específica e alternância para uma rota redundante, o SSCM não realiza a reconfiguração especializada nesta rota alternativa. Uma investigação desta funcionalidade aumentaria bastante a aderência da solução à aplicação prática na indústria.

- Expansão do escopo de atuação a mais de um ponto de entrada e um ponto de saída de rede. Embora não tenha sido identificadas na prática aplicações que envolvam mais de um ponto convergente para a entrada e saída da rede, um estudo para ampliação da solução de modo a evitar esta restrição pode ser desenvolvido, partindo-se da solução proposta neste trabalho.

- Integração do SSCM à SOPE. Muitas vezes, a reconfiguração da topologia ou característica de tráfego de uma rede permite uma otimização na mesma. Como um exemplo simples, cite-se a comutação de uma rota em anel de um segmento a outro, em virtude do congestionamento de um elemento em específico. Em uma situação como essa, a SOPE é a responsável por otimizar a forma com que o tráfego é encaminhado. Também neste caso a reconfiguração pelo SSCM não é automática,

de maneira que uma forma de integração entre estas duas entidades ainda tem que ser desenvolvida.

- Da mesma forma, a integração do SSCM à SPE, que objetiva a proteção de forma automatizada da rede, também é um trabalho em aberto. A reconfiguração especializada de uma rede pode ser uma das ações que mitigam ou mesmo interrompem um ataque na rede. Como um exemplo simples, frente a um ataque do tipo DDoS, o SSCM poderia reconfigurar os fluxos de modo a priorizar os pacotes vindos dos elementos já conhecidos pela rede. A integração entre as funcionalidades poderia contribuir para maior confiabilidade do sistema sobre o domínio SONAr.

## 5.3 Contribuições em Produção Bibliográfica

Durante o desenvolvimento deste trabalho, alguns artigos foram produzidos e submetidos, sendo os de maior relevância listados abaixo.

Publicação como autor principal do artigo "*Network Self-configuration for Edge Elements using Self-Organizing Networks Architecture (SONAr)*" no *10$^{th}$ International Conference on Cloud Computing and Services Science* (CLOSER 2020), em maio de 2020 (OLIVEIRA et al., 2020).

Publicação como co-autor do artigo "*Bootstrapping and Plug-and-Play Operations on Software Defined Networks: A Case Study on Self-configuration using the SONAr Architecture*" no *10$^{th}$ International Conference on Cloud Computing and Services Science* (CLOSER 2020), em maio de 2020 (GONÇALVES et al., 2020).

Apresentação como autor principal desta dissertação "*Uma Proposta para Implementação Prática de Auto-configuração Especializada de Elementos de Borda em Redes SDN Utilizando a Arquitetura SONAr*" no *XIV Workshop de Teses e Dissertações em Ciência da Computação* (XIV WTDCC), em novembro de 2020.

Submissão como autor principal do artigo "*Specialized Network Self-Configuration: An Approach Using Self-Organizing Networks Architecture (SONAr)*" no *11$^{th}$ International Conference on Cloud Computing and Services Science* (CLOSER 2021), em dezembro de 2020.

# Referências


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

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

3GPP. *System Architecture for the 5G System (5GS); Stage 2 (Release 16)*. [S.l.], 2019. Disponível em: <https://portal.3gpp.org/desktopmodules/Specifications/SpecificationDetails.aspx?specificationId=3144>.

3GPP. *Technical Specification Group Services and System Aspects; Release 16 Description; Summary of Rel-16 Work Items (Release 16)*. [S.l.], 2019. Disponível em: <https://portal.3gpp.org/desktopmodules/Specifications/SpecificationDetails.aspx?specificationId=3493>.

3GPP. *3GPP Releases*. 2020. Disponível em: <https://www.3gpp.org/specifications/67-releases>.

ABDALLAH, S. et al. A Network Management Framework for SDN. In: **2018 9th IFIP International Conference on New Technologies, Mobility and Security (NTMS)**. Paris: IEEE, 2018. p. 1–4. ISBN 978-1-5386-3662-6. Disponível em: <https://doi.org/10.1109/NTMS.2018.8328672>.

AL-JAWAD, A. et al. Policy-based QoS Management Framework for Software-Defined Networks. In: **2018 International Symposium on Networks, Computers and Communications (ISNCC)**. Rome: IEEE, 2018. p. 1–6. ISBN 978-1-5386-3779-1. Disponível em: <https://doi.org/10.1109/ISNCC.2018.8530994>.

BIANCHI, G. et al. Superfluidity: a flexible functional architecture for 5G networks: G. Bianchi *et al.* **Transactions on Emerging Telecommunications Technologies**, v. 27, n. 9, p. 1178–1186, set. 2016. ISSN 21613915. Disponível em: <https://doi.org/10.1002/ett.3082>.

CAMPANELLA, A.; DANDOUSH, A.; MANASSAKIS, L. **ONOS - Open Network Operating System**. 2017. Disponível em: <https://onosproject.org/wp-content/uploads/2018/01/1-ONOS_Introduction.pdf>.

CASADO, M. et al. Ethane: Taking Control of the Enterprise. p. 12, 2007. Disponível em: <https://doi.org/10.1145/1282427.1282382>.

CHIOSI, M. et al. **Network Functions Virtualisation, An Introduction, Benefits, Enablers, Challenges & Call for Action**. 2012. Disponível em: <https://portal.etsi.org/NFV/NFV_White_Paper.pdf>.

CISCO SYSTEMS. **Understanding Rapid Spanning Tree Protocol (802.1w)**. 2017. Disponível em: <https://www.cisco.com/c/en/us/support/docs/lan-switching/spanning-tree-protocol/24062-146.html>.

ERICSSON. **Ericsson Mobility Report November 2019**. [S.l.], 2019. 36 p. Disponível em: <ericsson.com/mobility-report>.

ETSI. **ETSI_NFV_Architecture.pdf**. 2014. Disponível em: <https://www.etsi.org/deliver/etsi_gs/nfv/001_099/002/01.02.01_60/gs_nfv002v010201p.pdf>.

GALAXY TECHNOLOGIES LLC. **GNS3 Software**. 2020. Disponível em: <https://www.gns3.com/software>.

GANEK, A. G.; CORBI, T. A. The dawning of the autonomic computing era. **IBM Systems Journal**, v. 42, n. 1, p. 5–18, 2003. ISSN 0018-8670. Disponível em: <https://doi.org/10.1147/sj.421.0005>.

GONÇALVES, M. et al. Bootstrapping and plug-and-play operations on software defined networks: A case study on self-configuration using the sonar architecture. In: INSTICC. **Proceedings of the 10th International Conference on Cloud Computing and Services Science - Volume 1: CLOSER,**. SciTePress, 2020. p. 103–114. ISBN 978-989-758-424-4. Disponível em: <https://doi.org/10.5220/0009406901030114>.

HAKIRI, A.; BERTHOU, P. Leveraging SDN for The 5G Networks: Trends, Prospects and Challenges. In: **Software Defined Mobile Networks : Beyond LTE Network Architecture**. Madhusanka Liyanage (Editor), Andrei Gurtov (Editor), Mika Ylianttila (Editor), 2015, (Wiley Series on Communications Networking and Distributed Systems). p. 23. ISBN 978-1-118-90028-4. Disponível em: <https://doi.org/10.1002/9781118900253.ch5>.

HOLMA, H.; TOSKALA, A. **LTE Advanced: 3GPP Solution for IMT-Advanced**. 1. ed. John Wiley and Sons Ltd, 2012. ISBN 978-1-119-97405-5. Disponível em: <https://doi.org/10.1002/9781118399439>.

JIANG, W.; STRUFE, M.; SCHOTTEN, H. D. A SON decision-making framework for intelligent management in 5G mobile networks. In: **2017 3rd IEEE International Conference on Computer and Communications (ICCC)**. Chengdu: IEEE, 2017. p. 1158–1162. ISBN 978-1-5090-6352-9. Disponível em: <https://doi.org/10.1109/CompComm.2017.8322725>.

KATIYAR, R. et al. Auto-Configuration of SDN Switches in SDN/Non-SDN Hybrid Network. In: **Proceedings of the Asian Internet Engineering Conference - AINTEC '15**. Bangkok, Thailand: ACM Press, 2015. p. 48–53. ISBN 978-1-4503-3914-8. Disponível em: <https://doi.org/10.1145/2837030.2837037>.

LINUX FOUNDATION. **Open vSwitch Documentation**. 2016. Disponível em: <http://docs.openvswitch.org/en/latest/>.

_____. **What Is Open vSwitch? — Open vSwitch 2.13.90 documentation**. 2016. Disponível em: <http://docs.openvswitch.org/en/latest/intro/what-is-ovs/>.

MCCORRY, L. K. Physiology of the Autonomic Nervous System. **American Journal of Pharmaceutical Education**, p. 11, 2007. Disponível em: <https://doi.org/10.5688/aj710478>.

MCKEOWN, N. et al. OpenFlow: enabling innovation in campus networks. **ACM SIGCOMM Computer Communication Review**, v. 38, n. 2, p. 69, mar. 2008. ISSN 01464833. Disponível em: <https://doi.org/10.1145/1355734.1355746>.

MOYSEN, J.; GIUPPONI, L. From 4G to 5G: Self-organized network management meets machine learning. **Computer Communications**, v. 129, p. 248–268, set. 2018. ISSN 01403664. Disponível em: <https://doi.org/10.1016/j.comcom.2018.07.015>.

NEVES, P. et al. The SELFNET Approach for Autonomic Management in an NFV/SDN Networking Paradigm. **International Journal of Distributed Sensor Networks**, v. 12, n. 2, p. 2897479, fev. 2016. ISSN 1550-1477, 1550-1477. Disponível em: <https://doi.org/10.1155/2016/2897479>.

NEXTEPC INC. **NextEPC**. 2020. Disponível em: <https://nextepc.org/>.

OLIVEIRA, D. et al. Network self-configuration for edge elements using self-organizing networks architecture (sonar). In: **Proceedings of the 10th International Conference on Cloud Computing and Services Science - Volume 1: CLOSER**. [s.n.], 2020. p. 408–414. ISBN 978-989-758-424-4. Disponível em: <https://doi.org/10.5220/0009465104080414>.

ONF. **Open Networking Foundation - Our Mission**. 2011. Disponível em: <https://www.opennetworking.org/mission/>.

_____. **Software-Defined Networking: The New Norm for Networks**. 2012. Disponível em: <https://www.opennetworking.org/images/stories/downloads/sdn-resources/white-papers/wp-sdn-newnorm.pdf>.

_____. **OpenFlow Switch Specification - Version 1.5.1 (Protocol version 0x06)**. Open Networking Foundation, 2015. Disponível em: <https://www.opennetworking.org/wp-content/uploads/2014/10/openflow-switch-v1.5.1.pdf>.

_____. **Open Network Operating System (ONOS) SDN Controller for SDN/NFV Solutions**. 2020. Disponível em: <https://opennetworking.org/onos/>.

_____. **Software-Defined Networking (SDN) Definition**. 2020. Disponível em: <https://www.opennetworking.org/sdn-definition/>.

ONOS PROJECT. **Flow Rules - ONOS**. 2019. Disponível em: <https://wiki.onosproject.org/display/ONOS/Flow+Rules>.

_____. **ONOS Project Mission**. 2020. Disponível em: <https://onosproject.org/mission/>.

OPENDAYLIGHT PROJECT. **About The OpenDaylight Foundation**. 2018. Disponível em: <https://www.opendaylight.org/about>.

ORAN ALLIANCE. **ORAN Alliance**. 2019. Disponível em: <https://www.o-ran.org/>.

PATIL, P.; GOKHALE, A.; HAKIRI, A. Bootstrapping Software Defined Network for flexible and dynamic control plane management. In: **Proceedings of the 2015 1st IEEE Conference on Network Softwarization (NetSoft)**. London, United Kingdom: IEEE, 2015. p. 1–5. ISBN 978-1-4799-7899-1. Disponível em: <https://doi.org/10.1109/NETSOFT.2015.7116132>.

SOUZA NETO., N. V. de et al. A self-healing platform for the control and management planes communication in softwarized and virtualized networks. In: INSTICC. **Proceedings of the 10th International Conference on Cloud Computing and Services Science - Volume 1: CLOSER,**. SciTePress, 2020. ISBN 978-989-758-424-4. Disponível em: <https://doi.org/10.5220/0009465204150422>.

SYMMETRICOM. **Timing and Synchronization for LTE-TDD and LTE-Advanced Mobile Networks**. 2013. Disponível em: <https://www.chronos.co.uk/files/pdfs/wps/WP-Timing-Sync-LTE-SEC.pdf>.

TELUS; HUAWEI. White Paper, **Next generation son for 5g**. 2016. Disponível em: <https://www.huawei.com/en/industry-insights/outlook/mobile-broadband/insights-reports/next-generation-son-for-5g>.

VAHDAT, A.; CLARK, D.; REXFORD, J. A Purpose-built Global Network: Google's Move to SDN. **ACM Queue**, v. 13(8), p. 100–125, set. 2015. ISSN 1542-7749. Disponível em: <https://doi.org/10.1145/2838344.2856460>.

XU, L. et al. CogNet: A network management architecture featuring cognitive capabilities. In: **2016 European Conference on Networks and Communications (EuCNC)**. Athens, Greece: IEEE, 2016. p. 325–329. ISBN 978-1-5090-2893-1. Disponível em: <https://doi.org/10.1109/EuCNC.2016.7561056>.

YI, B. et al. A comprehensive survey of Network Function Virtualization. **Computer Networks**, v. 133, p. 212–262, mar. 2018. ISSN 13891286. Disponível em: <https://doi.org/10.1016/j.comnet.2018.01.021>.

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

# Anexos

ANEXO A

# Flow Rules (Experimento 1)

## A.1 Flow Rules Antes da Self-Configuration

```
Flows sw-1
/ # ovs-ofctl dump-flows br0
 cookie=0x0, duration=609.965s, table=0, n_packets=498, n_bytes=47014,
    priority=0 actions=NORMAL

Flows sw-2
/ # ovs-ofctl dump-flows br0
 cookie=0x0, duration=626.453s, table=0, n_packets=230, n_bytes=35303,
    priority=0 actions=NORMAL

Flows sw-access
/ # ovs-ofctl dump-flows br0
 cookie=0x0, duration=641.227s, table=0, n_packets=570, n_bytes=51363,
    priority=0 actions=NORMAL
```

## A.2 Flow Rules Após a Self-Configuration

```
Flows sw-1
/ # ovs-ofctl dump-flows br0
 cookie=0xbe00004b1402c6, duration=433.308s, table=0, n_packets=3,
    n_bytes=294, priority=5000,ip,nw_src=192.168.0.200,nw_dst=10.1.1.203
     actions=mod_vlan_vid:3663,mod_vlan_pcp:7,output:eth2
 cookie=0xbe00001f4aa967, duration=433.307s, table=0, n_packets=3,
    n_bytes=306, priority=5000,ip,nw_src=10.1.1.203,nw_dst=192.168.0.200
     actions=strip_vlan,output:eth1
 cookie=0xbe00000fba9e14, duration=433.303s, table=0, n_packets=3,
    n_bytes=294, priority=5000,ip,nw_src=192.168.0.200,nw_dst=10.1.1.202
     actions=mod_vlan_vid:3662,mod_vlan_pcp:2,output:eth2
```

```
 cookie=0xbe0000cb36cbe9, duration=433.302s, table=0, n_packets=3,
    n_bytes=306, priority=5000,ip,nw_src=10.1.1.202,nw_dst=192.168.0.200
     actions=strip_vlan,output:eth1
 cookie=0xbe0000b996d032, duration=433.302s, table=0, n_packets=7,
    n_bytes=686, priority=5000,ip,nw_src=192.168.0.200,nw_dst=10.1.1.201
     actions=mod_vlan_vid:3660,mod_vlan_pcp:1,output:eth2
 cookie=0xbe000023869ae9, duration=433.302s, table=0, n_packets=7,
    n_bytes=714, priority=5000,ip,nw_src=10.1.1.201,nw_dst=192.168.0.200
     actions=strip_vlan,output:eth1
 cookie=0xbe00001e05b1a5, duration=433.302s, table=0, n_packets=3,
    n_bytes=294, priority=5000,ip,nw_src=192.168.0.200,nw_dst=10.1.1.200
     actions=mod_vlan_vid:3661,mod_vlan_pcp:6,output:eth2
 cookie=0xbe0000b1569e1f, duration=433.300s, table=0, n_packets=3,
    n_bytes=306, priority=5000,ip,nw_src=10.1.1.200,nw_dst=192.168.0.200
     actions=strip_vlan,output:eth1
 cookie=0xbe00000b1cf02e, duration=444.750s, table=0, n_packets=4,
    n_bytes=204, priority=4000,arp actions=NORMAL
 cookie=0xbe00007875aac9, duration=444.742s, table=0, n_packets=45,
    n_bytes=9885, priority=4000,dl_type=0x88cc actions=NORMAL
 cookie=0xbe000028031f20, duration=444.744s, table=0, n_packets=3,
    n_bytes=186, priority=4000,ip,nw_dst=192.168.0.3 actions=NORMAL
 cookie=0xbe00006096a6c9, duration=444.742s, table=0, n_packets=6,
    n_bytes=639, priority=4000,ip,nw_dst=192.168.0.1 actions=output:eth2
 cookie=0xbe0000c88c1782, duration=444.742s, table=0, n_packets=3,
    n_bytes=294, priority=4000,ip,nw_dst=192.168.0.200 actions=output:
    eth1
 cookie=0xbe00004264e3c5, duration=444.742s, table=0, n_packets=0,
    n_bytes=0, priority=4000,udp,nw_dst=255.255.255.255,tp_dst=67
    actions=CONTROLLER:65535

Flows sw-2
/ # ovs-ofctl dump-flows br0
 cookie=0xbe000015ef16ab, duration=484.003s, table=0, n_packets=3,
    n_bytes=306, priority=5000,ip,nw_src=192.168.0.200,nw_dst=10.1.1.202
     actions=output:eth2
 cookie=0xbe000028714637, duration=483.997s, table=0, n_packets=3,
    n_bytes=306, priority=5000,ip,nw_src=10.1.1.202,nw_dst=192.168.0.200
     actions=output:eth1
 cookie=0xbe00005d07b7ba, duration=483.996s, table=0, n_packets=7,
    n_bytes=714, priority=5000,ip,nw_src=192.168.0.200,nw_dst=10.1.1.201
     actions=output:eth2
 cookie=0xbe0000b1f18212, duration=483.996s, table=0, n_packets=7,
    n_bytes=714, priority=5000,ip,nw_src=10.1.1.201,nw_dst=192.168.0.200
     actions=output:eth1
 cookie=0xbe0000ed7748af, duration=483.995s, table=0, n_packets=3,
    n_bytes=306, priority=5000,ip,nw_src=192.168.0.200,nw_dst=10.1.1.203
     actions=output:eth2
```

```
 cookie=0xbe00005e38fe80, duration=483.995s, table=0, n_packets=3,
    n_bytes=306, priority=5000,ip,nw_src=10.1.1.203,nw_dst=192.168.0.200
     actions=output:eth1
 cookie=0xbe00008f141d4f, duration=483.993s, table=0, n_packets=3,
    n_bytes=306, priority=5000,ip,nw_src=192.168.0.200,nw_dst=10.1.1.200
     actions=output:eth2
 cookie=0xbe000043390c81, duration=483.992s, table=0, n_packets=3,
    n_bytes=306, priority=5000,ip,nw_src=10.1.1.200,nw_dst=192.168.0.200
     actions=output:eth1
 cookie=0xbe00009d00e7ca, duration=493.340s, table=0, n_packets=6,
    n_bytes=533, priority=4000,ip,nw_dst=192.168.0.1 actions=output:eth2
 cookie=0xbe0000005b6f15, duration=493.337s, table=0, n_packets=2,
    n_bytes=132, priority=4000,ip,nw_dst=192.168.0.2 actions=NORMAL
 cookie=0xbe0000b133d14d, duration=493.337s, table=0, n_packets=3,
    n_bytes=186, priority=4000,ip,nw_dst=192.168.0.3 actions=output:eth1
 cookie=0xbe0000ced0462f, duration=493.326s, table=0, n_packets=3,
    n_bytes=294, priority=4000,ip,nw_dst=192.168.0.200 actions=output:
    eth1
 cookie=0xbe000037ff316e, duration=493.348s, table=0, n_packets=51,
    n_bytes=11186, priority=4000,dl_type=0x88cc actions=NORMAL
 cookie=0xbe0000e80dd038, duration=493.334s, table=0, n_packets=4,
    n_bytes=204, priority=4000,arp actions=NORMAL
 cookie=0xbe000049792391, duration=493.336s, table=0, n_packets=0,
    n_bytes=0, priority=4000,udp,nw_dst=255.255.255.255,tp_dst=67
    actions=CONTROLLER:65535

Flows sw-access
/ # ovs-ofctl dump-flows br0
 cookie=0xbe0000092e03b0, duration=519.604s, table=0, n_packets=7,
    n_bytes=714, priority=5000,ip,nw_src=192.168.0.200,nw_dst=10.1.1.201
     actions=strip_vlan,output:eth2
 cookie=0xbe0000a2adcec8, duration=519.603s, table=0, n_packets=7,
    n_bytes=686, priority=5000,ip,nw_src=10.1.1.201,nw_dst=192.168.0.200
     actions=mod_vlan_vid:3660,mod_vlan_pcp:1,output:eth1
 cookie=0xbe0000ad50e4b2, duration=519.602s, table=0, n_packets=3,
    n_bytes=306, priority=5000,ip,nw_src=192.168.0.200,nw_dst=10.1.1.202
     actions=strip_vlan,output:eth2
 cookie=0xbe000063a72e59, duration=519.601s, table=0, n_packets=3,
    n_bytes=294, priority=5000,ip,nw_src=10.1.1.202,nw_dst=192.168.0.200
     actions=mod_vlan_vid:3662,mod_vlan_pcp:2,output:eth1
 cookie=0xbe00005802ac63, duration=519.599s, table=0, n_packets=3,
    n_bytes=306, priority=5000,ip,nw_src=192.168.0.200,nw_dst=10.1.1.203
     actions=strip_vlan,output:eth2
 cookie=0xbe0000523d001f, duration=519.597s, table=0, n_packets=3,
    n_bytes=294, priority=5000,ip,nw_src=10.1.1.203,nw_dst=192.168.0.200
     actions=mod_vlan_vid:3663,mod_vlan_pcp:7,output:eth1
 cookie=0xbe00005df77779, duration=519.597s, table=0, n_packets=3,
```

```
    n_bytes=306, priority=5000,ip,nw_src=192.168.0.200,nw_dst=10.1.1.200
     actions=strip_vlan,output:eth2
 cookie=0xbe000086bbbc47, duration=519.595s, table=0, n_packets=3,
    n_bytes=294, priority=5000,ip,nw_src=10.1.1.200,nw_dst=192.168.0.200
     actions=mod_vlan_vid:3661,mod_vlan_pcp:6,output:eth1
 cookie=0xbe0000938b2533, duration=528.946s, table=0, n_packets=54,
    n_bytes=11988, priority=4000,dl_type=0x88cc actions=NORMAL
 cookie=0xbe000091f75bb7, duration=528.933s, table=0, n_packets=4,
    n_bytes=204, priority=4000,arp actions=NORMAL
 cookie=0xbe0000f6f82cb9, duration=528.937s, table=0, n_packets=3,
    n_bytes=186, priority=4000,ip,nw_dst=192.168.0.3 actions=output:eth1
 cookie=0xbe00005e3a1161, duration=528.936s, table=0, n_packets=2,
    n_bytes=132, priority=4000,ip,nw_dst=192.168.0.2 actions=output:eth1
 cookie=0xbe00004f767d83, duration=528.934s, table=0, n_packets=7,
    n_bytes=599, priority=4000,ip,nw_dst=192.168.0.1 actions=output:eth0
 cookie=0xbe0000cf94f045, duration=528.934s, table=0, n_packets=3,
    n_bytes=294, priority=4000,ip,nw_dst=192.168.0.200 actions=output:
    eth1
 cookie=0xbe000010f09384, duration=528.931s, table=0, n_packets=2,
    n_bytes=132, priority=4000,ip,nw_dst=192.168.0.4 actions=NORMAL
 cookie=0xbe000031e08922, duration=528.934s, table=0, n_packets=0,
    n_bytes=0, priority=4000,udp,nw_dst=255.255.255.255,tp_dst=67
    actions=CONTROLLER:65535
```

ANEXO **B**

# Testes de Conexão (Experimento 1)

## B.1 Demonstração da Conectividade Entre Elementos Antes da Self-Configuration

```
root@ne-eNodeB:~# ping 10.1.1.200
PING 10.1.1.200 (10.1.1.200) 56(84) bytes of data.
64 bytes from 10.1.1.200: icmp_seq=1 ttl=62 time=25.2 ms
64 bytes from 10.1.1.200: icmp_seq=2 ttl=62 time=26.9 ms
64 bytes from 10.1.1.200: icmp_seq=3 ttl=62 time=30.5 ms
^C
--- 10.1.1.200 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2002ms
rtt min/avg/max/mdev = 25.260/27.587/30.528/2.202 ms

root@ne-eNodeB:~# ping 10.1.1.201
PING 10.1.1.201 (10.1.1.201) 56(84) bytes of data.
64 bytes from 10.1.1.201: icmp_seq=1 ttl=62 time=35.1 ms
64 bytes from 10.1.1.201: icmp_seq=2 ttl=62 time=31.0 ms
64 bytes from 10.1.1.201: icmp_seq=3 ttl=62 time=26.0 ms
^C
--- 10.1.1.201 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2002ms
rtt min/avg/max/mdev = 26.005/30.745/35.187/3.759 ms

root@ne-eNodeB:~# ping 10.1.1.202
PING 10.1.1.202 (10.1.1.202) 56(84) bytes of data.
64 bytes from 10.1.1.202: icmp_seq=1 ttl=62 time=39.6 ms
64 bytes from 10.1.1.202: icmp_seq=2 ttl=62 time=24.3 ms
64 bytes from 10.1.1.202: icmp_seq=3 ttl=62 time=30.6 ms
^C
--- 10.1.1.202 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2003ms
rtt min/avg/max/mdev = 24.351/31.536/39.658/6.285 ms
```

```
root@ne-eNodeB:~# ping 10.1.1.203
PING 10.1.1.203 (10.1.1.203) 56(84) bytes of data.
64 bytes from 10.1.1.203: icmp_seq=1 ttl=62 time=31.1 ms
64 bytes from 10.1.1.203: icmp_seq=2 ttl=62 time=27.3 ms
64 bytes from 10.1.1.203: icmp_seq=3 ttl=62 time=23.6 ms
^C
--- 10.1.1.203 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2002ms
rtt min/avg/max/mdev = 23.625/27.371/31.139/3.067 ms

root@ne-eNodeB:~# ping 192.168.0.1
PING 192.168.0.1 (192.168.0.1) 56(84) bytes of data.
64 bytes from 192.168.0.1: icmp_seq=1 ttl=64 time=5.84 ms
64 bytes from 192.168.0.1: icmp_seq=2 ttl=64 time=1.09 ms
64 bytes from 192.168.0.1: icmp_seq=3 ttl=64 time=1.13 ms
64 bytes from 192.168.0.1: icmp_seq=4 ttl=64 time=1.13 ms
^C
--- 192.168.0.1 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3004ms
rtt min/avg/max/mdev = 1.095/2.302/5.844/2.045 ms
```

## B.2 Demonstração da Conectividade Entre Elementos Após a Self-Configuration

```
root@ne-eNodeB:~# ping 10.1.1.200
PING 10.1.1.200 (10.1.1.200) 56(84) bytes of data.
64 bytes from 10.1.1.200: icmp_seq=1 ttl=62 time=22.8 ms
64 bytes from 10.1.1.200: icmp_seq=2 ttl=62 time=38.0 ms
64 bytes from 10.1.1.200: icmp_seq=3 ttl=62 time=28.2 ms
^C
--- 10.1.1.200 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2003ms
rtt min/avg/max/mdev = 22.828/29.709/38.092/6.324 ms

root@ne-eNodeB:~# ping 10.1.1.201
PING 10.1.1.201 (10.1.1.201) 56(84) bytes of data.
64 bytes from 10.1.1.201: icmp_seq=1 ttl=62 time=31.6 ms
64 bytes from 10.1.1.201: icmp_seq=2 ttl=62 time=27.6 ms
64 bytes from 10.1.1.201: icmp_seq=3 ttl=62 time=24.0 ms
^C
--- 10.1.1.201 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2003ms
rtt min/avg/max/mdev = 24.054/27.790/31.645/3.106 ms
```

```
root@ne-eNodeB:~# ping 10.1.1.202
PING 10.1.1.202 (10.1.1.202) 56(84) bytes of data.
64 bytes from 10.1.1.202: icmp_seq=1 ttl=62 time=39.9 ms
64 bytes from 10.1.1.202: icmp_seq=2 ttl=62 time=26.3 ms
64 bytes from 10.1.1.202: icmp_seq=3 ttl=62 time=25.0 ms
^C
--- 10.1.1.202 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2002ms
rtt min/avg/max/mdev = 25.056/30.450/39.965/6.749 ms

root@ne-eNodeB:~# ping 10.1.1.203
PING 10.1.1.203 (10.1.1.203) 56(84) bytes of data.
64 bytes from 10.1.1.203: icmp_seq=1 ttl=62 time=22.3 ms
64 bytes from 10.1.1.203: icmp_seq=2 ttl=62 time=28.3 ms
64 bytes from 10.1.1.203: icmp_seq=3 ttl=62 time=30.0 ms
^C
--- 10.1.1.203 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2002ms
rtt min/avg/max/mdev = 22.397/26.916/30.000/3.265 ms

root@ne-eNodeB:~# ping 192.168.0.1
PING 192.168.0.1 (192.168.0.1) 56(84) bytes of data.
64 bytes from 192.168.0.1: icmp_seq=1 ttl=64 time=3.11 ms
64 bytes from 192.168.0.1: icmp_seq=2 ttl=64 time=3.22 ms
64 bytes from 192.168.0.1: icmp_seq=3 ttl=64 time=1.36 ms
^C
--- 192.168.0.1 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2002ms
rtt min/avg/max/mdev = 1.362/2.565/3.221/0.854 ms
```