UNIVERSIDADE DO VALE DO RIO DOS SINOS
UNIDADE ACADÊMICA DE GRADUAÇÃO
CURSO DE BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

JULIO RENNER

UMA ARQUITETURA DE ORQUESTRAÇÃO DE FUNÇÕES DE REDE DE ACESSO DE RÁDIO DESAGREGADAS

São Leopoldo
2020

JULIO RENNER

UMA ARQUITETURA DE ORQUESTRAÇÃO DE FUNÇÕES DE REDE DE ACESSO DE RÁDIO DESAGREGADAS

Artigo apresentado como requisito parcial para obtenção do título de Bacharel em Ciência da Computação, pelo Curso de Ciência da Computação da Universidade do Vale do Rio dos Sinos (UNISINOS)

Orientador(a): Prof. Dr. Cristiano B. Both

São Leopoldo
2020

UMA ARQUITETURA DE ORQUESTRAÇÃO DE FUNÇÕES DE REDE DE ACESSO DE RÁDIO DESAGREGADAS

Julio Renner[1]
Cristiano B. Both [2]

Resumo:

As redes de acesso por rádio da quinta geração (5G) estão passando por alterações significativas. Por exemplo, a utilização de recursos centralizados e virtualização das funções de rede que auxiliam na redução dos custos com operação e compra de equipamentos por permitir a utilização de hardware de propósito genérico. Além disso, a virtualização permite a desagregação das funções de rede de acordo com os requisitos de latência e banda, aumentando o número de funções de rede que precisam ser gerenciadas. Dessa forma, a virtualização das funções de rede impõe o desenvolvimento de novas soluções de orquestração ou a customização das já existentes para que supram requisitos específicos das redes móveis. Nos atuais ambientes em nuvem, contêineres são amplamente utilizados como forma de virtualização e implantação de software, onde vários estudos vêm sendo feitos utilizando contêineres para o fornecimento de funções de rede no cenário da 5G. Entretanto, em ambientes de grande escala, contêineres por si só não são suficientes e ferramentas de orquestração precisam ser utilizadas. A ferramenta adotada largamente na academia e na indústria é o Kubernetes (K8S), um gerenciador de infraestrutura virtual de código aberto e altamente customizável. Nesse contexto, esse trabalho apresenta uma arquitetura de customização, utilizando K8S e seguindo as definições das instituições padronizadoras, para suprir os requisitos de orquestração das funções de rede virtualizadas das redes móveis 5G. Os resultados obtidos mostram que o K8S, através de customizações, suporta os requisitos necessários, e.g., o posicionamento utilizando os operadores *RANPlacer* e *RANDeployer* desenvolvidos apresenta um número de saltos 230% menor que o posicionamento nativo do K8S.

Palavras-chave: Gerenciamento. Desagregação. RAN. Kubernetes.

Abstract: The Fifth Generation (5G) Radio Access Network will be affected by significant changes. The usage of centralized resources and virtual network functions (VNFs) helps by reducing costs with the operation and acquisition of equipment by allowing the usage of general-purpose hardware. However, the virtualization of network functions requires the development of new orchestration solutions or the customization of the existing ones in a way they meet the needs of a large and highly volatile environment with specific requirements. In today's cloud environments, containers are widely used to deploy and virtualize applications. Several studies have already been done to provide VNFs using containers in the 5G scenario. In production environments, however, containers by themselves are not enough and orchestration tools need to be used in addition. The most widely adopted tool is Kubernetes, an open-source and highly customizable container orchestrator. In this context, this paper aims to analyze Kubernetes and its customization points for the usage in the orchestration of virtual network functions in the 5G networks radio access layer, proposing an architecture that aims to supply all the specific needs in this scenario following the standards defined by the standard development organizations. The

[1]Graduando em de Ciência da Computação pela Unisinos. Email: juliorenner@edu.unisinos.br
[2]Mini-currículo do orientador. Email: cbboth@unisinos.br

results of the experiments applied demonstrated that K8S, through customizations, supports the requirements, for example, the positioning of the network functions using the *RANPlacer* and *RANDeployer* operators demonstrates an improvement of 230% compared with the native positioning of K8S.

Keywords: Management. Disaggregation. RAN. Kubernetes.

1 INTRODUÇÃO

A evolução das redes móveis de quinta geração (5G) propõe mudanças significativas na organização e arquitetura em relação as redes móveis atuais com o objetivo de suprir estritos requisitos de novos serviços. Entre os requisitos estão alta disponibilidade e baixíssima latência, além de suporte a um número massivo de dispositivos conectados a rede (MARSCH ÖMER BULAKçI, 2018). Apesar da transformação, as redes móveis mantém-se compostas por três principais componentes: (i) o núcleo da rede (CN - *Core Network*) responsável pela gestão da rede e dos serviços, (ii) as redes de acesso via rádio (RAN - *Radio Access Network*) responsáveis pela comunicação entre o CN e o equipamento do usuário (UE - *User Equipment*) via interface aérea e (iii) os UE que obtém acesso a rede e usufruem de suas funcionalidades. Um dos pontos chave das mudanças arquiteturais da 5G se trata da virtualização das funções de rede, sejam estas do CN ou da RAN. Através da virtualização é possível desacoplar as funções de hardware e software, uma vez que atualmente os dispositivos operam de forma monolítica e com hardware proprietário. A virtualização permite a flexibilização e o uso mais eficiente do hardware, acarretando em redução dos custos operacionais. Entretanto, para redes móveis, tal transformação envolve grandes desafios, principalmente devido a escala de dispositivos e restritos parâmetros de rede.

No contexto da RAN, a virtualização é desenvolvida em conjunto com a transformação da arquitetura na direção da nova-geração da RAN (NG-RAN - *Next Generation RAN*), definida pelas organizações padronizadoras 3rd Generation Partnership Project (3GPP) e International Telecommunication Union (ITU-T). A nova arquitetura flexibiliza o número de nós de rede em operação, com as opções de execução de apenas um único nó (i.e., enodeB monolítica), chamada de RAN distribuída (D-RAN - *Distribuited RAN*) e de execução de dois nós, (i.e., RRH - *Remote Radio Head* e BBU - *BaseBand Unit*) chamada de RAN em nuvem (C-RAN - *Cloud RAN*). Além de suportar os nós de rede sem desagregação (D-RAN) e com uma desagregação (C-RAN), que já eram modelos adotados em gerações anteriores, a nova arquitetura prevê o cenário com duas desagregação com composição de três nós de rede: (i) unidade central (CU - *Central Unit*), (ii) unidade distribuída (DU - *Distribuited Unit*), e (iii) unidade de rádio (RU - *Radio Unit*). Além disso, cada um desses três nós pode adotar diferentes configurações de desagregação em relação ao seus protocolos de comunicação. Essa flexibilidade tem como objetivo ser ciente das restrições da topologia da rede de transporte de forma que a desagregação utilizada varie de acordo com os requisitos da conexão do usuário.

A proposta de desagregação das funções de rádio contribui para o desenvolvimento da RAN

baseada em software e virtualizada. Contudo, o conceito de virtualização, amplamente avançado em aplicações de tecnologia da informação (TI), traz diversas transformações à RAN em diferentes tópicos. Dentre elas, a orquestração do posicionamento das funções de rádio é alvo de atenção especial, tanto na indústria, quanto na academia. Dado que a virtualização oportuniza a implementação da RAN em ambientes na nuvem e na borda da rede. A partir do momento que as funções de rede forem virtualizadas é preciso encontrar maneiras de posicioná-las de acordo com as constantes mudanças de requisitos dos usuários e sua localização. Por exemplo, a operação da rede precisa ser extremamente flexível, criando e removendo funções de rede de diversos locais, de acordo com a demanda e os requisitos dos usuários. Portanto, a orquestração das funções de rede virtualizadas é um desafio atual para os sistemas de telecomunicação. Apesar dos desafios contextualizados para a orquestração do posicionamento das funções de rádio, o mercado de software possui larga experiência e diferentes ferramentas para auxiliar no gerenciamento de aplicações e podem ser aplicadas no contexto das redes móveis. Uma das principais ferramentas de gestão de recursos e desenvolvimento de aplicações virtualizadas é o Kubernetes (K8S). O K8S é uma ferramenta de código aberto, largamente utilizada e possui uma ampla e ativa comunidade. Além disso, K8S é altamente customizável, permitindo ser utilizada no contexto das redes móveis, e.g., na orquestração das funções da RAN.

O desenvolvimento de uma arquitetura de orquestração de funções virtualizadas, incluindo o cenário da RAN, tem sido investigado tanto na indústria quanto na academia. Na indústria há projetos de comunidades relevantes, como o Open Network Automation Platform (ONAP) pela Linux Foundation e o Open Source MANO (OSM) pela European Telecommunications Standards Institute (ETSI) (MAMUSHIANE et al., 2019). Ambos em cenário de gestão de infraestrutura e orquestração e gerenciamento de funções virtualizadas de forma genérica (não específico para a RAN), desenvolvidos sob a ferramenta K8S. Entretanto, tópicos relevantes direcionados a RAN em ambientes virtualizados, tais como o posicionamento e ciência da rede de transporte, tem baixa exploração e pouca consistência nos trabalhos desenvolvidos pela indústria (ERIKSSON, 2017). Na academia a orquestração do posicionamento é amplamente analisada com modelagem matemática (GARCIA-SAAVEDRA et al., 2018b, 2018a; DALLA-COSTA et al., 2020), mas possui grandes oportunidades de desenvolvimento de arquitetura customizadas. Dessa forma, K8S oferece um padrão de customização chamado Padrão de Operadores, que permite a introdução de componentes que atuam da mesma forma que componentes nativos. No contexto da orquestração da RAN, é possível explorar esse ponto de customização para adição de componentes que definam modelos arquiteturais e adaptem a ferramenta para este cenário específico.

O presente trabalho tem como objetivo desenvolver a arquitetura NG-RAN com foco no posicionamento das funções de rádio virtualizadas. Assim como proposto pelas organizações padronizadoras e com direcionamento a um ambiente de orquestração de funções virtualizadas em contêineres gerenciado pelo K8S. A principal contribuição é uma arquitetura customizada para funções de rede virtualizada (NFV - *Network Function Virtualization*), especificada pela organização padronizadora ETSI, porém, com foco na adaptação do K8S para suporte dos re-

quisitos da RAN. Através do padrão de operadores, foram adicionados dois componentes que atuam de forma integrada aos componentes nativos do K8S, o *RANDeployer* e *RANPlacer*. O *RANDeployer* lida com a execução das funções de rede virtualizadas, enquanto o *RANPlacer* é responsável pelo posicionamento dessas funções. Para avaliação da arquitetura, foi desenvolvido um algoritmo de posicionamento baseado na técnica de busca em largura (BFS - *Breadth-First Search*), a virtualização com base no conceito de contêineres e divisão dos nós CU, DU e RU da ferramenta OpenAirInterface (OAI), além de um algoritmo de automatização da alocação da OAI. Para a avaliação da arquitetura customizada, foi utilizada a topologia do consórcio de desenvolvimento do 5G na Europa denominado de PASSION (PASSION Project, 2020), conectando a gestão do K8S a ciência da topologia da rede de transporte. Os resultados obtidos mostram que o K8S, através de customizações, suporta os requisitos necessários, por exemplo, o posicionamento utilizando os operadores *RANPlacer* e *RANDeployer* apresentam um número de saltos 230% menor que o posicionamento nativo do K8S.

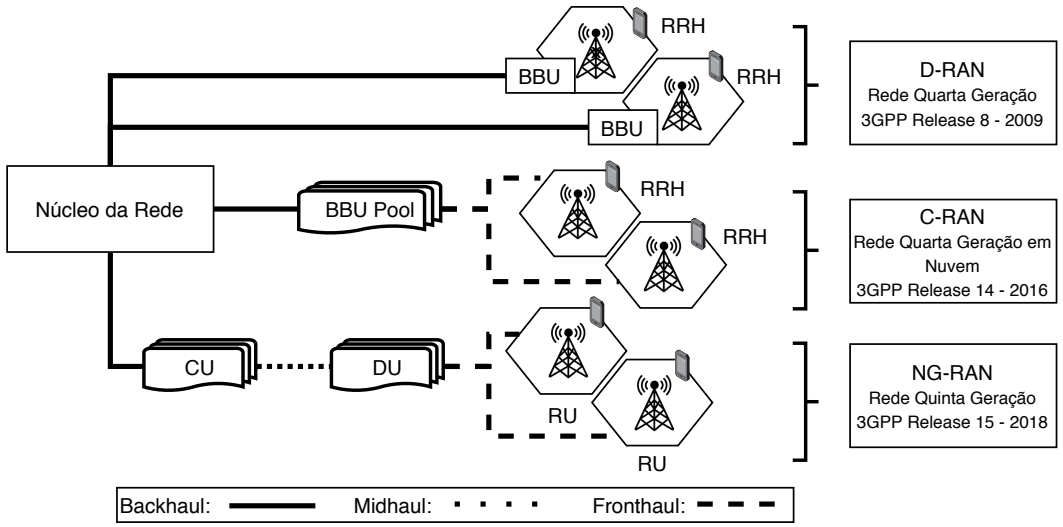
O artigo está organizado nas seguintes seções. Inicialmente, a Seção 2 apresenta os conceitos necessários para compreensão das seções subsequentes. A Seção 3 analisa os trabalhos já existentes que tem relação ao que foi desenvolvido. A Seção 4 apresenta a arquitetura elaborada, o protótipo desenvolvido e o ambiente de avaliação. A Seção 5 apresenta os resultados obtidos e faz uma análise do que pode ser percebido através deles, trazendo os impactos positivos e negativos do que foi desenvolvido. Por fim, a Seção 6 sumariza e conclui.

2 REDES DE RÁDIO DE ACESSO DE PRÓXIMA GERAÇÃO

A tecnologia em operação nas redes móveis LTE de Quarta Geração (4G) teve seu desenvolvimento iniciado pelo 3GPP a partir do *Release* 8. A estação rádio base (BS - *Base Station*) apresentou elevado aprimoramento, e foi denominado de *NodeB* evoluída (eNB - *evolved NodeB*). A eNB é composta por dois elementos de rede, a Unidade de Banda Base (BBU - *Baseband Unit*) e a Cabeça de Rádio Remota (RRH - *Remote Radio Head*) que consiste em unidades de rádio e antenas (AGRAWAL et al., 2017). A BBU tem responsabilidade sobre as funções de processamento de dados de banda base e, normalmente, é distribuído no mesmo espaço físico que a RRH. Neste caso, a base da arquitetura é o modelo D-RAN, que por sua vez possui padrões de comunicação e capacidade de processamento particulares para um número fixo de cargas de entrada da própria BS (CHEN et al., 2015). Entretanto, a arquitetura D-RAN carece de suporte as novas demandas de serviço, principalmente, baixa latência, altas taxa de dados, adaptabilidade em tráfego não uniforme, cobertura ultra-densa e o baixo consumo de energia (AGRAWAL et al., 2017). A Figura 1 detalha a RAN em cada estágio de evolução.

O conceito da RAN centralizada em ambientes de computação em nuvem (C-RAN - *Cloud-RAN*), introduzido a partir do *Release* 14, foi a primeira arquitetura desenvolvida para suprir as deficiências apresentadas pela D-RAN (3GPP, 2017). Sendo assim, na arquitetura C-RAN, diferentes BBUs passaram a estar em um local centralizado e as RRHs foram mantidas no formato distribuído. Tal alteração criou a rede de transporte *fronthaul* para a comunicação

Figura 1: Arquitetura D-RAN, C-RAN e NG-RAN.



Fonte: Elaborada pelo Autor

entre as BBUs e as RRHs. Com a arquitetura C-RAN, destacam-se as vantagens relacionadas à redução de custo de locais físicos (infraestrutura de prédios), operação e manutenção. Além disso, possibilita o compartilhamento e elasticidade (alocação de mais ou menos recursos de acordo com a demanda). Desta forma, a rede passou a operar com padrões não uniformes dos usuários. Por exemplo, permitindo adaptação para movimentos da sociedade urbana no dia-a-dia, i.e., de deslocamento de áreas residenciais para as áreas comerciais durante o período do dia, e retorno à noite (AGRAWAL et al., 2017).

A centralização das BBUs e consequente elevação nas taxas de dados proposta pela redes 5G, causaram impacto direto nos requisitos de largura de banda e latência das redes *fronthaul*. Tornando a arquitetura inviável em grande escala de aplicações. Na academia, pesquisas foram realizadas de forma a buscar alternativas de reduzir a quantidade de bits por segundo na conexão do *fronthaul*, porém, essa solução de contorno inclui a necessidade de inclusão de mais funções locais na BS e elevado processamento antes da transmissão. Com base nas arquiteturas anteriores, foi desenvolvido o *Release 15*, especificando a arquitetura NG-RAN. Sendo essa composta por gNBs (*Generation NodeBs*), sucessor das eNBs. De um ponto de vista funcional, a gNB é composta pelos nós CU, DU e RU. Tal transformação, tem como objetivo a distribuição de nós RUs e DUs na rede, consequente eficiência no manuseio dos recursos computacionais e garantia dos requisitos baseados nos serviços, e.g., requisito de latência (AL-DULAIMI et al., 2018).

Atualmente, a rede de transporte está sendo reprojetada e desenvolvida considerando segmentos de *Fronthaul* (FH), *Midhaul* (MH) e *Backhaul* (BH). O *Fronthaul* é responsável pela comunicação entre RU e DU (gNB-DU). O *Midhaul*, é o segmento onde ocorre a comunicação entre DU e CU (gNB-CU). Por fim, o *Backhaul* é responsável pela comunicação entre CU e núcleo da rede 5G. A integração entre esses segmentos de transporte é chamada de *Crosshaul* (CARDOSO et al., 2020). Assim como, a divisão proposta na arquitetura oportuniza a flexibi-

lização da composição das funções que compõem os nós de rádio e seus respectivos protocolos (MARSCH et al., 2018), conforme apresentado na próxima subseção.

2.1 Desagregação da RAN

A arquitetura RAN da 4G pode ser considerada como um bloco monolítico, onde o eNB contém todas as funções e protocolos a nível de acesso a rede. Isso resulta em uma arquitetura simples onde poucas interações entre nós lógicos precisam ser definidos. Segundo (BERTENYI et al., 2018), desde as primeiras fases de estudos relacionados a NG-RAN, foi identificado que a separação de sua pilha de protocolos poderia trazer alguns benefícios, dentre eles podem se elencar: (i) a implementação flexível de hardware, que possibilita soluções escaláveis com melhorr custo-benefício; (ii) uma arquitetura que permite a coordenação de desempenho dos recursos, balanceamento de carga e otimização de desempenho em tempo real, possibilitando a utilização de funções virtualizadas de software; e (iii) a adaptação para diferentes casos de uso de acordo com a Qualidade de Serviço (QoS) necessária, por exemplo, requisitos de latência.

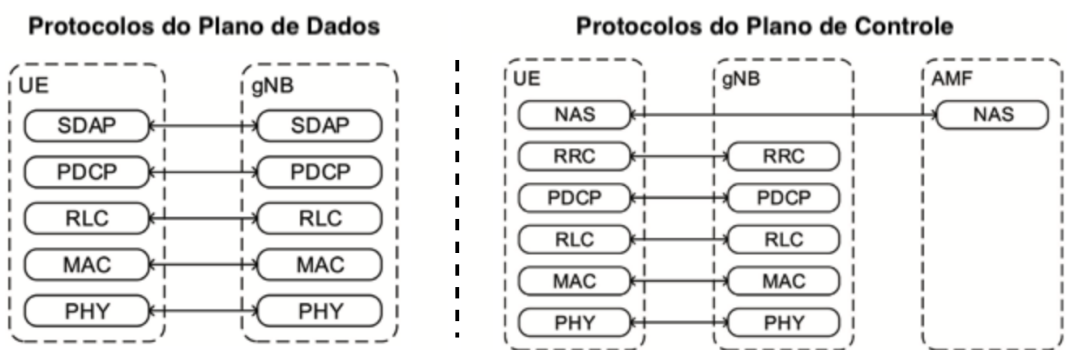


Figura 2: Pilha de protocolos do plano de dados e de controle

A escolha de como fazer a separação das funções da pilha de protocolos de nova geração (*NR - New Radio*), depende do cenário, limitações e serviço desejado, *e.g.*, baixa latência, alta banda, densidade de usuários específicas dentro de uma área geográfica, etc. (BERTENYI et al., 2018). Os protocolos da pilha da NG-RAN estão divididos entre os que compõem o plano de dados (também chamado de plano de usuário) e os que compõem o plano de controle. A Figura 2 demonstra os protocolos que são utilizados pelo plano de dados e plano de controle (3GPP, 2017; CARDOSO et al., 2020; BERTENYI et al., 2018; ETSI, 2017).

A divisão da pilha de protocolos da gNB em funções de rede tem o importante papel de reduzir o tráfego sobre o *fronthaul* e facilitar requisitos de sincronização e latência. Quanto mais o processamento é feito na RU, menores os requisitos sobre a rede de transporte (AL-DULAIMI; WANG; I, 2018). A 3GPP propôs oito opções de separação das funções entre as unidades centralizadas e distribuídas, considerando requisitos de transporte, em especial, vazão e latência (CARDOSO et al., 2020). Trazendo para a separação dos componentes CU, DU e RU, essas definições indicam o que será executado no DU e o que será executado no CU, podendo

variar de acordo com a necessidade de QoS.

As oito opções de separação das funções da RAN, ilustradas na Figura 3, são baseadas nas pilhas de protocolos descritas anteriormente. Cada protocolo, com exceção do NAS e do SDAP, é desagregado em uma função, com a especificidade dos protocolos RLC, MAC e PHY, que são divididos em camada inferior e superior (3GPP, 2017). Dentre os pontos de separação, no máximo duas desagregações são admitidas, onde o conjunto superior da separação resulta na CU, o conjunto intermediário resulta na DU, e o conjunto inferior combinado com a conexão física com o equipamento de Radio frequência (RF) resulta na RU.

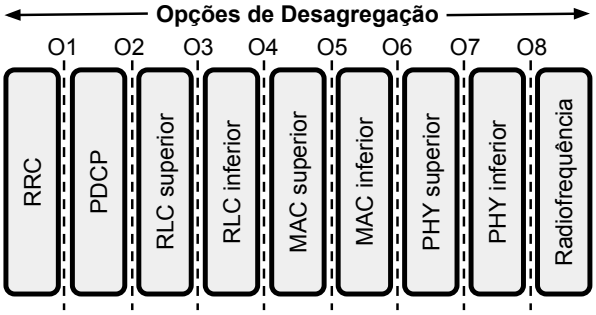


Figura 3: Desagregação da pilha de protocolos

A desagregação da NG-RAN facilita a flexibilização, aplicação dos requisitos do *crosshaul* e distribuição da pilha de protocolos das funções de rádio. A desagregação combinada com a virtualização permite a execução da CU, DU e RU sobre hardware de propósito genérico, compartilhados e otimizados. Essa combinação promove um controle flexível, escalável, com redução de custos e com uso eficiente de recursos computacionais (MARSCH et al., 2018). A próxima subseção aborda a virtualização no contexto das funções de rede, como pode ser desenvolvida, e as vantagens que esse movimento introduz no contexto da arquitetura NG-RAN.

2.2 Virtualização de Funções de Rede

O Instituto Padronizador Europeu de Telecomunicações (ETSI - *European Telecomunication Standards Institute*), propôs uma estrutura de arquitetura para a virtualização de funções de rede que pode ser vista na Figura 4. A arquitetura proposta consiste em três partes: Infraestrutura das Funções Virtualizadas de Rede (NFVI - *Nework Functions Virtualization Infrastructure*), Função Virtualizada de Rede (VNF), e Gerenciamento e Orquestração da NFV (NFV MANO - NFV *Management and Orchestration*). NFVI fornece todos recursos de infraestrutura necessários para a execução das funções virtualizadas e NFV MANO é responsável pelo gerenciamento dos recursos de hardware e software da rede (HABIBI et al., 2019).

A funcionalidade de gerenciamento e orquestração (NFV MANO) é composta pelo conjunto de três componentes: Orquestrador NFV (NFVO - *NFV Orchestrator*), Gerenciador de VNF (VNFM - VNF *Manager*) e Gerenciador de Infraestrutura Virtualizada (VIM - *Virtualized Infrastructure Manager*). NFVO é responsável por gerenciar o ciclo de vida dos Serviços de

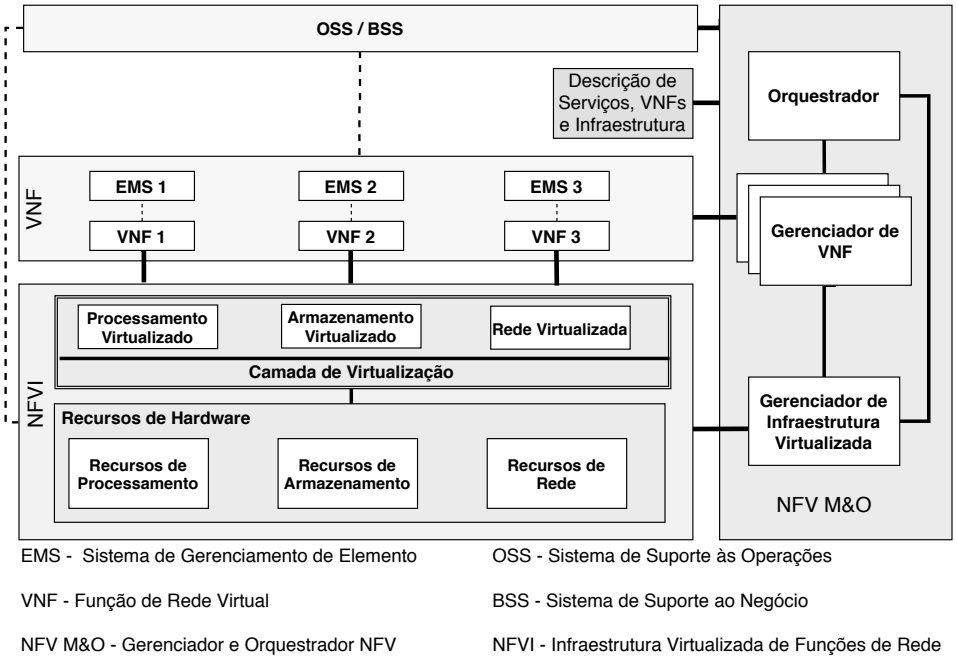


Figura 4: Proposta da ETSI de uma arquitetura NFV

Rede (NS - *Network Services*) e coordenar o gerenciamento do ciclo de vida dos NS, VNFs e dos recursos da NFVI para garantir uma alocação otimizada dos recursos e melhor conectividade. VNFM é o bloco que gerencia o ciclo de vida das VNFs. Por fim, VIM é o bloco responsável por controlar e gerenciar os recursos computacionais, de armazenamento, e rede, dentro do contexto de um domínio de infraestrutura (BERNARDOS CJ.; LYNCH, 2019).

ETSI se limita a definir a arquitetura básica para suportar a virtualização, mas não entra nos detalhes de implementação, abrindo espaço para diferentes projetos e pesquisas que buscam projetar e desenvolver uma proposta. Alguns dos projetos e ferramentas que vem sendo estudadas são: Open Source Mano (OSM), Kubernetes (K8S), Open Platform for NFV (OPNFV), entre outros. Atualmente, as duas formas de virtualização mais utilizadas são Máquinas Virtuais (VM - Virtual Machines) e Contêineres.

2.3 Orquestração de Funções de Rede

Quando se utiliza uma aplicação conteinerizada, esta pode ser composta por inúmeros contêineres, sendo que esse número é geralmente dinâmico, variando de acordo com a utilização da aplicação, sempre buscando manter seu funcionamento e desempenho. Dessa forma, a utilização de contêineres em ambientes produtivos envolve muito mais do que simplesmente a sua execução e o mesmo se aplica no contexto da orquestração de funções de rede conteinerizadas. Questões como definir o melhor local para sua execução, fornecer os requisitos de rede e armazenamento, lidar com descobrimento de serviços, balanceamento de carga, melhor utilização de recursos computacionais, entre outros aspectos. Os orquestradores de contêineres surgem para

ajudar a operar e automatizar essas necessidades (Red Hat, 2020).

O gerenciador de infraestrutura virtual e também orquestrador de contêineres mais popular e mais utilizado em ambientes produtivos e de larga escala é o Kubernetes (K8S). Além de ser o mais utilizado, o K8S possui uma ampla comunidade de usuários ativos que está constantemente contribuindo à sua base de código aberto. Adicionalmente, o K8S tem alto nível de maturidade e está alinhado com os critérios do principal órgão padronizador, o *Cloud Native Computing Foundation* (CNCF) (Cloud Native Computing Foundation, 2020). Embora, existam opções como Docker Swarm e Apache Mesos disponíveis no mercado e com objetivos similares o K8S é a ferramenta que mais se destaca.

Docker Swarm é uma ferramenta de *clustering* nativa para Docker que transforma vários nós executando Docker em um *cluster*, abstraindo a complexidade de modo que cada nó seja gerenciado de forma centralizada como um único ambiente Docker. O Apache Mesos é conhecido como o kernel de sistemas distribuídos. Ele foi desenvolvido baseando-se nos mesmos princípios do kernel Linux, só que em outro nível de abstração. O Mesos é utilizado em todos os nós de um *cluster* e provê suporte a *frameworks* como Apache Spark, Hadoop, Cassandra, Kafka, Elastic Search, etc. Por fim, disponibiliza uma API para alocar recursos e executar suas tarefas através do *cluster* ou *cloud* (The Apache Software Foundation, 2020).

2.4 Kubernetes

K8S é uma ferramenta de código aberto iniciada pela Google e que atualmente é mantida pela *Cloud Native Computing Foundation*. Se uma aplicação é capaz de ser executada em um contêiner o K8S poderá gerenciá-la. Além disso, K8S possui vários pontos de customização, tornando-o uma ferramenta altamente adaptável e permitindo que melhorias sejam feitas para a utilização nos mais diversos contextos, como na orquestração das funções virtualizadas da RAN das redes 5G (OGBUACHI et al., 2019; SHAH, 2019). K8S possui diversas funcionalidades, dentre as quais se destacam:

- Descobrimento de Serviços e Balanceamento de carga: K8S pode expor contêineres utilizando nomes ou o próprio endereço IP. Se o tráfego é alto, é possível distribuir as requisições entre diferentes contêineres para manter a aplicação estável.

- Gerenciamento de Armazenamento: permite que se utilize sistemas de armazenamento de escolha do usuário, como armazenamento local, NFS (*Network File System*), armazenamento fornecido por provedores de nuvem, entre outros.

- Atualizações e reversões automáticas: é possível descrever o estado desejado dos contêineres em execução, através dessa descrição, o K8S consegue alterar o estado atual para o estado desejado de forma controlada, mantendo a aplicação em funcionamento durante o fornecimento de novos contêineres com uma versão atualizada.

- Distribuição de contêineres: K8S automaticamente lida com a alocação dos contêineres

nos nós disponíveis, o usuário pode informar a quantidade de memória RAM e CPU necessárias para cada contêineres e o K8S alocará os contêineres da melhor maneira.

- Recuperação automática: se o K8S detecta que contêineres falharam ele os reinicia, os substitui por novos e finaliza contêineres que não respondem as validações definidas pelo usuário.

- Gerenciamento de configurações e informações sensíveis: é possível armazenar e gerenciar informações como senhas, tokens OAuth e chaves SSH. É possível atualizar e adicionar essas informações sem precisar construir novas imagens de contêineres.

No K8S, *Pods* são a menor unidade de implementação computacional que podem ser gerenciadas. Um *Pod* é composto por um ou mais contêineres que compartilham armazenamento e rede. Os contêineres que compartilham o mesmo *Pod* possuem o mesmo endereço IP e espaço de portas, podendo se comunicar pelo *localhost*. Os recursos e contêineres especificados por um *Pod* se encontram no mesmo nó e são orquestrados como uma unidade. Esse modelo de implementação de múltiplos contêineres em um *Pod* favorece aplicações que possuem acoplamento, em um mundo pré-contêineres, pode-se pensar em uma aplicação executada na mesma máquina física ou virtual (Cloud Native Computing Foundation, 2020).

A arquitetura do K8S, ilustrada na Figura 5, isola as aplicações de gerenciamento do *cluster* das aplicações de usuário, fazendo com que as aplicações necessárias para o funcionamento do K8S sejam executadas em nós separados das aplicações dos usuários, os quais são chamados respectivamente de nós mestres e nós escravos. O conjunto de aplicações que faz o gerenciamento do *cluster* é chamado de Plano de Controle. Esse plano é composto pelos seguintes componentes: *kube-apiserver* que é o componente responsável por toda comunicação no plano de controle; *etcd* que tem o papel de banco de dados utilizado para armazenar os dados relacionados ao plano de controle; *kube-scheduler* é o agendador do *cluster*, observa os contêineres que ainda não foram alocados para um nó e determina em que nó esses contêineres devem ser executados; *kube-controller-manager*; executa os processos de controle (um *loop* que observa um recurso específico do *cluster*, validando se o estado atual é o mesmo definido pelo usuário); e o *cloud-controller-manager* é o responsável pela integração da infraestrutura fornecida em ambientes de nuvem com o K8S, com ele é possível determinar, por exemplo, se um nó foi deletado após não obter resposta, configurar rotas e gerenciar balanceadores de carga.

Além dos componentes do plano de controle, existem componentes que todos os nós devem possuir para que o ambiente de execução do K8S seja fornecido, são eles: *kubelet* que garante que todos os contêineres definidos na especificação de um *Pod* estão executando da forma esperada; *kube-proxy* que gerencia as regras de rede nos nós; e o *container runtime* que é o software que é responsável por executar os contêineres. O K8S suporta qualquer *container runtime* que implemente sua interface de execução, exemplos de *container runtime* suportadas são Docker, Containerd e CRI-O, entre outras. Além dos componentes responsáveis por fornecer o ambiente no qual o K8S executa, o *cluster* depende de objetos que são persistidos e

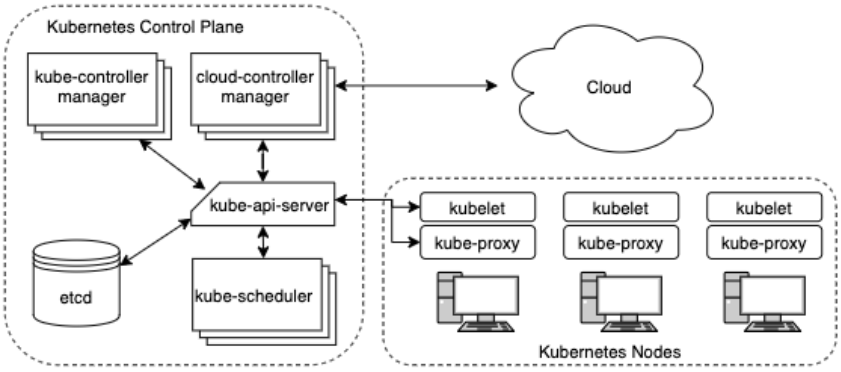


Figura 5: Arquitetura do Kubernetes

Fonte: Documentação do Kubernetes

responsáveis por descrever o estado desejado do *cluster*. Assim que um objeto é criado, o K8S trabalha constantemente para garantir que a descrição seja representada no ambiente. Todas as operações relacionadas a manipulação dos objetos ocorrem através do componente *kube-apiserver*. Existem inúmeros objetos disponíveis em um *cluster* Kubernetes, e mais podem ser adicionados através de customizações, alguns dos mais relevantes são: *Pods*, declara um ou mais contêineres, juntamente com informações relevantes para que o K8S consiga monitorá-lo e detectar um estado indesejado, além de outras informações; *Namespace*, abstração fornecida pelo K8S para suportar múltiplos *clusters* virtuais sobre a mesma infraestrutura; *Node*, abstração que representa as instâncias de máquinas virtuais ou físicas presentes no *cluster*; e *Service*: recurso responsável pelo descobrimento de serviços, para ter um ponto de acesso único para um conjunto determinado de *pods*.

Praticamente, todo objeto do K8S possui na sua definição interna dois sub-objetos: *Spec* e *Status*. A *Spec* é necessária durante a criação do objeto, possuindo o estado desejado. O *Status* descreve o estado atual, atualizado pelo K8S (FOUNDATION, 2020). Em outras palavras, se a aplicação está com algum problema, o *Status* irá refletir esse estado de erro, enquanto a *Spec* continuará contendo o estado para o qual o K8S deve alterá-la. Durante a criação de um objeto do K8S são campos mandatórios: *apiVersion*, que define a versão da API do K8S que está sendo utilizada; *kind*, onde é descrito o tipo de objeto que está sendo criado; *metadata*, que contém informações que identificam unicamente um objeto como nome e *namespace*, além de outros metadados que podem ser adicionados; *Spec*, descreve o estado desejado do objeto a ser criado.

Pontos de Customização

Um dos principais atrativos do K8S para a utilização com NFV é sua alta customização para orquestrar contêineres. São sete pontos de extensão ao todo, sendo eles (FOUNDATION, 2020): *Kubectl*, é um cliente para acesso ao *cluster* via linha de comando; extensão de acesso, por meio desta funcionalidade as validações definidas pelo usuário podem ser adicionadas as requisições recebidas pelo *kube-apiserver*; recursos customizados, com eles é possível adicionar

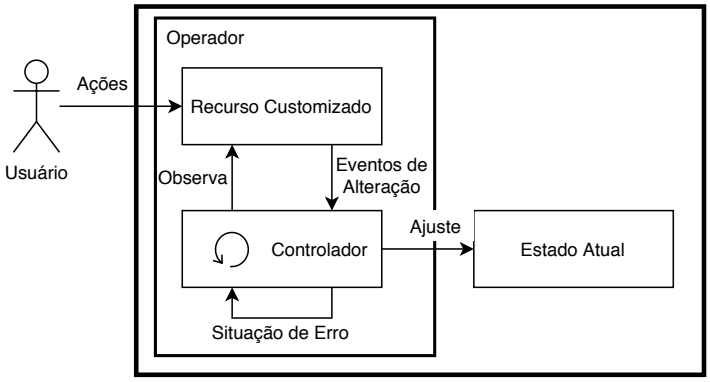
recursos definidos pelo usuário e disponibilizá-los através da API do *cluster*; agendador, pode ser alterado, completamente substituído ou até coexistir com outros; controladores, é possível adicionar controladores definidos pelo usuário; *plugins* de rede, com eles pode-se customizar o gerenciador de rede do K8S; e *plugins* de armazenamento, através deles é possível adicionar novos tipos de armazenamento. Dentro desses pontos de customização, a combinação da utilização de recursos customizados e controladores, combinação que forma o padrão de operadores, se destaca como um ponto para utilização na otimização do posicionamento de funções de rede no contexto da RAN na 5G.

Padrão de Operadores

Um operador é uma forma de empacotar, executar e manter uma aplicação do K8S, ou seja, adicionar aplicações as já fornecidas nativamente pelo orquestrador. Um operador é construído sobre o K8S para automatizar todo o ciclo de vida do software que ele gerencia, fornecendo automação específica para uma aplicação. Operadores tornam mais fácil a implantação e execução de serviços utilizados por aplicações e fornecem uma forma consistente de distribuir software em um *cluster* K8S, reduzindo o esforço com suporte através da identificação e correção automática de problemas (DOBIES; WOOD, 2020).

Operadores funcionam estendendo o plano de controle e a API do K8S. Na sua forma mais simples, um operador adiciona um recurso na API do Kubernetes, chamado *Custom Resource* (CR), e um componente no plano de controle que monitora e mantém os recursos do novo tipo. Baseados nas informações descritas no recurso adicionado, CR e componente do plano de controle podem tomar ações. A combinação da extensão da API e da adição de um componente no plano de controle, compõem o que é chamado de Operador. Na prática, o K8S compara um conjunto de recursos com a realidade, ou seja, o estado de execução do *cluster*. A partir dessa comparação, ele toma ações para fazer com que a realidade reflita o estado desejado descrito (DOBIES; WOOD, 2020). A Figura 6 ilustra o funcionamento do padrão de operadores.

O padrão de operadores se torna interessante para utilização na implantação das NG-RANs, pois pode ser utilizado para lidar com a inicialização e configuração das VNFs, a partir de uma definição declarada pelo operador da rede, dentre diversas outras questões do gerenciamento do ciclo de vida das funções. Normalmente, são implantados de forma que monitorem continuamente as aplicações, lidem com *backup* de dados, recuperem as aplicações de estados de erro, gerenciem as atualizações de versões, dentre inúmeras outras tarefas que possam ser automatizadas e que sejam necessárias (DOBIES; WOOD, 2020). Todas essas ações de gerenciamento de ciclo de vida podem assim ser facilmente reaproveitadas no contexto de redes móveis. Utilizar o padrão de operadores é muito promissor nesse contexto, pois auxilia na diminuição dos custos com manutenção das VNFs. Como os operadores são extremamente especializados no contexto de sua aplicação, eles executam operações para recuperá-las de um possível estado de erro, diminuindo a necessidade de intervenção manual de operadores de rede. Entretanto, não eximindo que em certos casos essas intervenções sejam necessárias.



Elaborada pelo autor

Figura 6: Padrão de Operadores

3 TRABALHOS RELACIONADOS

Atualmente, o desenvolvimento da tecnologia 5G é alvo de diferentes frentes de trabalho, sejam essas pelas organizações padronizadoras, indústria e academia. Neste sentido, a construção de arquiteturas de referência NFV são alvo de diversos grupos de trabalho. Dentre os principais projetos de código aberto para o ETSI MANO, se destacam CORD/XOS (*Central Office Re-architected as a Datacenter/Everything-as-a-Services*), ONAP, OSM, Cloudify, Open Baton, Gohan e Tacker. Lideradas pelas respectivas organizações ON.lab, Linux Foundation, ETSI, Giga Space, Fraunhofer, NTT Data e OpenStack Foundation (MAMUSHIANE et al., 2019). Destacam-se os projetos ONAP e OSM, uma vez que conforme a Tabela 1, possuem maior amplitude nos requisitos de recursos de orquestração e da arquitetura NFV MANO.

Tabela 1: Padrão de Orquestradores (MAMUSHIANE et al., 2019).

Solução de Orquestração	Líder	Orquestrador de recursos			NFV Mano Framework			Multi-Site
		Cloud	NFV	SDN	VNFM	NFVO	OSS/BSS	
CORD/XOS	ON.lab	X	X	X	X	X		X
ONAP	Linux Foundation	X	X	X	X	X	X	X
OSM	ETSI	X	X	X	X	X		
Cloudify	GigaSpace	X	X		X	X		
OpenBaton	Fraunhofer	X	X		X	X		X
X-MANO	H2020 Vital		X			X		X
Gohan	NTT Data	X	X	X	X	X	X	
Tacker	OpenStack Foundation	X	X		X	X		
TeNor	FP7 T-NOVA	X	X	X		X		

Para a evolução da virtualização da RAN em específico, ambos, OSM e ONAP, estão na mesma direção das principais iniciativas de virtualização da RAN, OpenRAN e O-RAN Alliance, e possuem suporte a virtualização por contêineres (GAVRILOVSKA et al., 2020). Outro ponto em comum entre OSM e ONAP é a utilização da ferramenta de gestão de infraestrutura e orquestração de contêineres Kubernetes. Visto que essa é a ferramenta mais popular e utilizada em ambientes produtivos e de larga escala de TI.

As soluções de orquestração OSM e ONAP não são desenvolvidas de forma customizada para a RAN. Neste sentido, Mosaic5G é a iniciativa sem fins lucrativos e promotora para contribuição entre a indústria ou acadêmia. Mosaic5G projetou a plataforma de código aberto Kube5G com as funcionalidades da arquitetura NFV MANO para orquestração de contêineres das redes móveis, incluindo a RAN, a partir do K8S (NIKAEIN; CHANG; ALEXANDRIS, 2018). Kube5G explora a funcionalidade de operadores com o desenvolvimento do Kube5G-Operator para automatização da implantação de recursos e serviços e disponibiliza dinamicidade para desagregação da RAN a partir do uso do software aberto de simulação OpenAirInterface (OAI).

Na academia, muitos trabalhos tem o propósito de virtualizar a RAN via o OAI em ambientes K8S ou em seus concorrentes. Neste sentido, o artigo (SEUNG-QUE et al., 2018) faz a virtualização da RAN com apenas uma desagregação (opção 2) utilizando a ferramenta de orquestração FLEXELL. Enquanto que (LUO et al., 2018) e (DZOGOVIC et al., 2019) desenvolvem a virtualização da RAN diretamente na ferramenta Docker, sem a especificação de uso de um orquestrador. Por fim, o trabalho (NOVAES et al., 2020) aplica o K8S para a implementação do OAI conteinerizada. Entretanto, nenhum dos trabalhos explora orquestração, posicionamento e escalonamento da RAN virtualizada e desagregada.

Ainda na literatura, embora não utilize uma ferramenta para o desenvolvimento de orquestração (por exemplo, K8S ou Docker Swarm), o trabalho (DALLA-COSTA et al., 2020) propõe um orquestrador para balanceamento de carga da C-RAN. O foco da pesquisa é na variação da opção de desagregação 7, e consequente posicionamento das funções de rádio virtualizadas, em uma infraestrutura com três níveis: nuvem, regional e borda. O tópico posicionamento é amplamente pesquisado pela academia, e consequentemente associado a orquestração (GARCIA-SAAVEDRA et al., 2018b, 2018a; MOLNER et al., 2019; FONSECA et al., 2019). Entretanto, o tema tem como base o desenvolvimento de algoritmos matemáticos, com baixa investigação e conexão com a arquitetura NFV MANO. Além de terem baixa ciência da topologia das redes.

Apesar de todo o desenvolvimento em andamento sobre arquiteturas virtualizadas para as rede móveis 5G, a funcionalidade de posicionamento das funções virtualizadas da RAN em arquiteturas NFV MANO baseadas em contêiner possuem uma lacuna aberta para investigação. Tal recurso, é apenas explorado na arquitetura do OSM, em parceria com o ARCTOS LABs, ainda com validações genéricas e com perspectivas futuras de conclusão (ERIKSSON, 2017). Neste sentido, o presente trabalho propõe explorar um arquitetura NFV MANO, alinhada com as principais iniciativas, utilizando K8S como orquestrador de funções de rádio virtualizadas. Além disso, contribuindo com o desenvolvimento do posicionamento das funções virtualizadas da RAN, a partir da construção de um operador para aplicação de algoritmos de posicionamento e um operador de implantação da nova arquitetura NG-RAN ciente da topologia.

4 ARQUITETURA DE ORQUESTRAÇÃO DE FUNÇÕES DESAGREGADAS

Nesta seção, é apresentada a arquitetura de orquestração das VNFs da NG-RAN desenvolvida utilizando K8S. A arquitetura tem como base a referência ETSI NFV. São abordadas as customizações necessárias para que o K8S consiga garantir os requerimentos das funções de rede da NG-RAN e proporcionar uma fácil implantação da rede através da descrição dos requerimentos por parte do Operador da Rede (OR). A Subseção 4.1 aborda a arquitetura proposta e sua adaptação na topologia da NG-RAN. A Subseção 4.2 detalha o protótipo desenvolvido utilizando o Kubernetes, enquanto a Subseção 4.3 detalha o ambiente no qual o protótipo foi validado.

4.1 Blocos Arquiteturais

A arquitetura contempla os componentes da arquitetura NG-RAN propostos para as redes móveis 5G. Além disso, suporta virtualização baseada na arquitetura NFV tendo como objetivo estruturar os componentes responsáveis pela orquestração e posicionamento das funções virtualizadas da RAN, desenvolvidos dentro do bloco funcional NFVO. Os componentes que compõem a arquitetura são: (i) RANPlacer: responsável pela otimização do posicionamento das funções de rede; (ii) Algoritmo de posicionamento: responsável pela otimização das funções de rede utilizadas pelo RANPlacer; (iii) RANDeployer: responsável pela execução das funções de rede, sua configuração e inicialização; e (iv) topologia: descrição da topologia de rede, onde as funções estão sendo posicionadas.

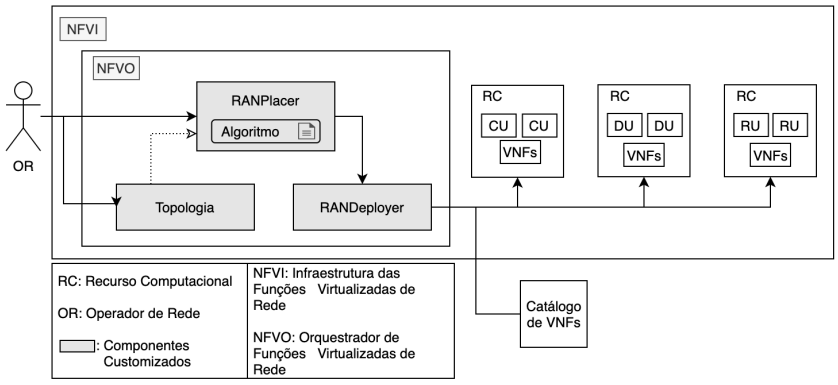


Figura 7: Blocos Arquiteturais

Para a implementação do posicionamento e execução das VNFs NG-RAN, dividiu-se as etapas em duas grandes ações. A primeira é denominada de otimização, executada pelo componente RANPlacer. A mesma leva em consideração o conjunto de VNFs existentes no *cluster*, ou que precisam ser posicionadas, e busca o melhor posicionamento. A segunda é a execução, realizada pelo RANDeployer. Esse é responsável por iniciar as VNFs nos Recursos Computacionais (RCs) selecionados pela tarefa de otimização. É importante destacar que o algoritmo

busca a otimização de um conjunto de funções de rede, de forma que o otimizador avalia o melhor posicionamento para este conjunto e não apenas para um indivíduo. A avaliação individual de uma cadeia de funções de rede, em um posicionamento sob demanda, é considerada uma aplicação diferente da proposta e não é desenvolvida no presente trabalho. As funções de rede ficam armazenadas no Catálogo de VNFs. Quando uma VNF é inicializada sua implantação é feita pelo RC, onde a função é executada. A visão geral da arquitetura pode ser observada na Figura 7.

Para realizar o posicionamento da rede, o algoritmo de posicionamento é projetado para ser inserido conforme o objetivo de cada operador de rede. De forma que, o algoritmo em si é genérico para o componente RANPlacer, sendo necessário apenas implementar a interface definida que é chamada pelo mesmo. O algoritmo recebe como entrada as seguintes informações: (i) topologia de rede; (ii) requisitos de cada Desagregação da RAN, conforme descrito em (TAYQ, 2017); (iii) recursos computacionais e sua capacidade disponível para utilização; (iv) requisitos de hardware de cada função de rede; e (v) posição das RUs de cada cadeia de VNFs que deve ser posicionada.

Com base nessas informações, o algoritmo deve determinar o posicionamento das cadeias de funções de rede. Uma cadeia de funções de rede é um conjunto composto de uma CU, uma DU e uma RU que se comunicam. A saída do algoritmo fornece a posição das CUs e DUs de cada cadeia. O posicionamento da RU é definido pelo OR, pois trata-se de uma configuração física envolvendo antenas de transmissão. Caso não exista nenhum posicionamento que supra os requisitos de Desagregação da RAN ou não existam recursos computacionais disponíveis, o algoritmo deve continuar e posicionar o máximo de cadeias de funções de rede possíveis.

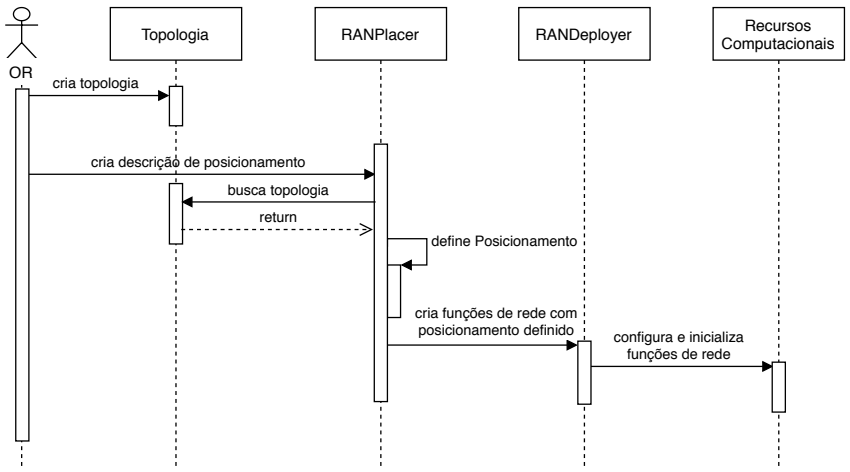


Figura 8: Diagrama de Sequência dos Componentes da Arquitetura

A Figura 8 apresenta a sequência de etapas que envolve os componentes da arquitetura. O processo é iniciado pelo operador de rede que fornece a topologia com as informações de rede. Após, o operador de rede, fornece para o *RANPlacer* uma descrição das funções de rede que devem ser posicionadas. Essa descrição inclui o posicionamento das RUs e o nome da topologia criada inicialmente, que será utilizada. Com essas informações o *RANPlacer* busca a topologia

fornecida e define o posicionamento das CUs e DUs, de acordo com o algoritmo utilizado. Com o posicionamento definido, *RANPlacer* envia a requisição de criação das funções de rede para o *RANDeployer*, que configura e inicializa as funções de rede nos recursos computacionais definidos, finalizando o processo de posicionamento e criação das funções de rede.

4.2 Protótipo

Para validar a arquitetura proposta, um protótipo foi construído utilizando o K8S. Essa ferramenta foi concebida para atuar como VIM, sendo responsável por gerenciar os RCs disponíveis e inicializar as funções de rede nos nós desejados. RCs possuem os recursos de hardware necessários para a execução dos contêineres, assim o K8S trata esses recursos através do objeto *Node*. Cada *Node* possui todos os componentes necessários para a inicialização e gerenciamento de *Pods*, que nesse contexto são as funções de rede (CU, DU e RU). As funções de rede são armazenadas no Catálogo de VNFs, que no protótipo foi representado por um repositório de imagens de contêineres. Uma imagem de contêiner é um binário que inclui todo o conteúdo necessário para execução de um contêiner, como bibliotecas, além de metadados que descrevem capacidades e necessidades do contêiner. Todos os nós possuem acesso a esse repositório e fazem a implantação da imagem da função de rede no momento em que essa for executada. No protótipo desenvolvido, o *Docker Registry* foi utilizado como Catálogo de VNFs. O *Docker Registry* é um servidor que armazena imagens de contêineres Docker e fica localizado fora do *cluster* K8S. Cada nó do *cluster* deve ter uma comunicação individual com o *Docker Registry*. Dessa forma, a partir do momento em que uma imagem de contêiner de uma VNF é criada, e passa pelo ciclo de desenvolvimento estabelecido, é possível enviá-la para o *Docker Registry* que deve armazená-la. O K8S registra *Docker Registries* e os utiliza para criar os contêineres.

Outro aspecto chave da arquitetura é a customização criada sobre o K8S para permitir atuar de maneira apropriada na orquestração de funções de rede da RAN. Para adaptar o K8S para o cenário de posicionamento de funções de rede da RAN, dois operadores foram desenvolvidos: RANPlacer e RANDeployer. Os operadores possuem as responsabilidades descritas na Seção 4.1, sendo o RANPlacer responsável por otimizar o posicionamento das funções de rede da RAN e o RANDeployer responsável por toda a configuração necessária para que uma cadeia de funções de rede seja executada. Dessa forma, RANDeployer cria os *Pods* e configurações (*ConfigMaps*) no K8S. Além dos operadores foi definido um padrão para descrição da topologia da rede, que é recebida pelo RANPlacer. Com essas informações RANPlacer e RANDeployer lidam com todo o processo de implantação das cadeias de funções de rede.

A topologia de rede, que foi determinada como uma entrada, foi definida como um *ConfigMap*, que serve para armazenar configurações não confidenciais. Essas configurações são armazenadas em uma estrutura chave e valor, diferentemente da maioria dos objetos do K8S, *ConfigMap* não possui o campo *spec*. No caso da topologia, essa informação é representada por um arquivo JSON. Duas informações principais precisam ser fornecidas no arquivo: Nós e Enlaces. Os nós representam os recursos computacionais, onde as funções de rede podem ser

posicionadas e devem ter os mesmos nomes dos recursos *Nodes* do K8S. Os Enlaces, representam as conexões entre os nós e contêm as informações de capacidade e latência.

Algoritmo de posicionamento

O algoritmo é uma parte importante para a execução da validação da estrutura proposta no ambiente K8S. Para implementar o algoritmo, uma interface foi definida e chamada de *Placement*. A interface *Placement* possui uma função *Place* que executa o posicionamento das funções de rede. Para o protótipo, uma estrutura da linguagem Golang foi utilizada para implementar a interface. Durante a criação da estrutura, são fornecidas a topologia, requisitos de desagregação, informações dos nós do *cluster* e recursos necessários para execução de cada função de rede. As informações passadas em tempo de criação são mantidas como propriedades internas da classe que podem ser utilizadas na implementação de suas funções. O algoritmo, baseado nessas informações e no posicionamento das RUs fornecido para a função *Place*, preenche a posição da CU e DU, caso exista um caminho que supra os requisitos de latência e banda e que os nós selecionados possuam os recursos computacionais necessários. Essas validações precisam ser feitas pelo algoritmo. Caso o posicionamento seja possível a função retorna *true*, caso contrário *false*. Caso ocorra algum erro na execução, um erro é retornado. No Código Fonte (*Listing*) 1 são exibidos a assinatura da interface que deve ser implementada e a função que retorna uma estrutura que implementa a interface requisitada na linguagem Golang.

Listing 1 Interface de Posicionamento e Função de Criação de Implementação da Interface

```
1 type Placement interface {
2         Place([]*ChainPosition) (bool, error)
3 }
4 func NewPlacement(topology, disaggregations, k8sNodes,
5      requestedResources) Placement {}
```

A fim de avaliar a arquitetura proposta, foi desenvolvido um algoritmo de posicionamento simples. O algoritmo consiste de uma busca em profundidade (BFS - *Breath First Search*) que encontra os caminhos possíveis entre o núcleo da rede até a RU informada pelo usuário. Tendo os caminhos possíveis, ele avalia um a um e escolhe o primeiro que supre as demandas (algoritmo guloso) de banda, latência e recursos computacionais. Caso nenhum caminho possua os requisitos necessários, o algoritmo tenta posicionar a próxima cadeia de funções da RAN de acordo com a posição da próxima RU da lista recebida. Para a validação da arquitetura apenas o cenário que posiciona a CU, DU e RU em nós diferentes foi considerado, utilizando as desagregações O2 e O6 mencionadas na Figura 3. É importante destacar que outras desagregações são possíveis de acordo com a implementação do algoritmo.

Assim como para a Topologia, foi definido um padrão para o fornecimento dos requisitos de rede que devem ser supridos para o *RANPlacer*. Como essas informações são estáticas, não precisam ser fornecidas pelo operador de rede quando um novo posicionamento for requisitado

e são adicionadas ao *cluster* K8S, juntamente com a instalação do operador *RANPlacer*. Da mesma forma que a topologia, essa informação é fornecida através de um *ConfigMap* e segue o formato JSON. No *ConfigMap*, devem ser informados os requisitos de latência e banda que cada desagregação deve possuir, ou seja, para cada desagregação devem ser informados os requisitos do *Backhaul*, *Midhaul*, *Fronthaul* e *Crosshaul*. Quando a validação de rede para o posicionamento for feita, esses requisitos precisam ser supridos. Os requisitos utilizados para validação podem ser vistos na Tabela 2.

Tabela 2: Requisitos de rede da desagregação

	Backhaul	Midhaul	Fronthaul	Crosshaul
Latência (Milisegundos)	-	30	2	30
Banda (Mbps)	151	151	152	-

OpenAirInterface e Free5GC

Para a avaliação e desenvolvimento do mapeamento proposto, o projeto OpenAirInterface (OAI) foi utilizado. OAI é uma aliança de software com código aberto e com a missão de fornecer software e ferramentas para a pesquisa das redes móveis de quinta geração (5G) (ALLIANCE, 2020). Entretanto, o mesmo teve de ser customizado para ter suporte aos três nós de rede (CU, DU e RU), assim como virtualizado de forma conteinerizada. O OAI já fornece uma base de código que pode ser utilizada para alguns cenários de redes móveis e está em constante desenvolvimento. As funções de rede foram desagregadas em CU, DU e RU seguindo as desagregações O2 e O6 mencionadas na Figura 3.

Para realizar a desagregação mencionada duas imagens de contêineres foram geradas. Uma imagem contém as informações para execução do *ltesoftmodem*, executável gerado através do *build* do projeto OAI e que implementa o LTE eNodeB, enquanto a outra imagem contém as informações para execução do *lte-uesoftmodem*, executável gerado através do *build* do projeto OAI e que possui a implementação do LTE UE. A imagem *ltesoftmodem* é utilizada na execução da CU e DU, enquanto a imagem *lte-uesoftmodem* é utilizada na execução da RU (OpenAirInterface Software Alliance, 2020). O papel que cada contêiner irá desempenhar é definido através de um arquivo de configuração que é fornecido ao *ltesoftmodem* e *lte-uesoftmodem* em tempo de execução. A Figura 9 exibe os *Pods* (funções de rede) com execução do *core*, *cu*, *du* e *ru*. Além disso, demostra a conexão entre o dispositivo que simula o equipamento do usuário e a Internet, passando por toda a cadeia de rede da RAN, núcleo, e chegando à Internet. Além disso, o Free5GC foi utilizado para o Núcleo 5G, definido na *Release 15* da 3GPP.

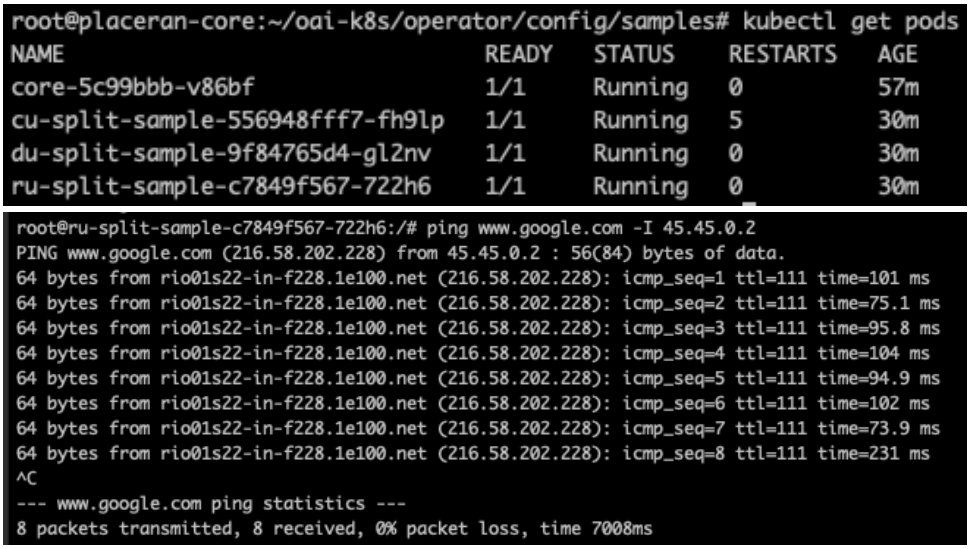


Figura 9: Configuração Núcleo e RAN

RANDeployer e RANPlacer

K8S oferece o padrão de operadores que permite estender seu comportamento. A fim de encontrar o melhor posicionamento para VNFs, o padrão de operadores se mostra uma boa opção, pois, através da linguagem de programação Golang, se tem a liberdade de implementar o comportamento desejado junto ao K8S. Para este cenário, através da descrição das VNFs que devem ser posicionadas no *cluster*, o operador pode executar um algoritmo de otimização definido e determinar o posicionamento das funções de rede. Um operador pode interagir diretamente com o componente *API-Server*, e tendo o posicionamento das funções de rede, criar os recursos necessários no *cluster*. Após a tarefa de otimização ser executada, e as funções de rede criadas no K8S, essas precisam ser executadas nos nós selecionados. Dessa forma, a tarefa de execução ainda precisa ser feita. O agendador do K8S é responsável por definir o melhor nó para os *Pods* criados no K8S e executar as operações necessárias para que eles sejam executados. Nesse caso, como o posicionamento já foi definido pelo Otimizador (Operador), a etapa de definição do posicionamento não é necessária, e o agendador apenas valida se o nó selecionado possui os requisitos mínimos para receber a função de rede e, caso a resposta seja positiva, solicita a inicialização do *Pod* no nó.

Para atender o objetivo descrito, foram desenvolvidos dois operadores: RANPlacer e RANDeployer. Ambos operadores adicionam um recurso customizado na API do K8S com o mesmo nome. O RANDeployer é responsável por iniciar as funções de rede da RAN, que nesse caso são: CU, DU e RU. Portanto, esse operador aceita uma descrição de onde cada uma dessas funções devem ser executada. Caso não seja fornecida, o agendador irá selecionar um nó para sua execução, seguindo o comportamento padrão do K8S. Além disso, o RANDeployer gera os arquivos de configuração necessários e faz o encadeamento entre as funções de rede fornecendo o que é necessário para que a comunicação ocorra. O RANDeployer reflete uma cadeia de CU,

DU e RU, ou seja, para cada cadeia que deve ser posicionada um recurso RANDeployer deve ser criado no *API-Server*. O RANDeployer não possui inteligência para o posicionamento, ele apenas cria as funções de rede a nível de K8S. O RANPlacer é responsável por determinar o posicionamento das funções de rede e criar os recursos de RANDeployer, já com o posicionamento desejado. Entretanto, para executar sua tarefa e posicionar as funções de rede da melhor maneira, é necessário fornecer informações relacionadas a topologia da rede e gasto de banda da alocação de um posicionamento específico. Dessa forma, o RANPlacer passa a ter conhecimento da topologia da rede e da capacidade de banda e latência entre as conexões, podendo aplicar um posicionamento mais inteligente.

O padrão de operadores combina dois pontos de extensão: a extensão do *API-Server*, adicionando novos recursos a API do K8S, e a adição de um controlador que monitora e toma ações sobre os recursos definidos. Por exemplo, a *spec* descreve o estado desejado que o K8S deve refletir no *cluster*. Nesse caso, as informações que descrevem o estado desejado são o posicionamento dos nós, nos quais as funções de rede devem ser criadas e o IP do núcleo da rede, com o qual a cadeia de funções vai estabelecer a comunicação. Para isso, um recurso do tipo *ConfigMap*, nativo do K8S, é criado para cada função e fornecido ao respectivo *Pod*. Esse *ConfigMap* contém os IPs que devem ser utilizados para a comunicação entre *Core*, CU, DU e RU. O usuário final não precisa estar ciente da existência do *ConfigMap*, e tão pouco da execução e configuração dos *Pods*. Esse é um trabalho de responsabilidade do operador. As informações de posicionamento da CU (*cuNode*), DU (*duNode*) e RU (*ruNode*) não são obrigatórias no objeto RANDeployer. Caso não sejam fornecidas, o agendador do K8S utiliza o comportamento padrão para o posicionamento. Um pseudo-código de exemplo de objeto RANPlacer pode ser observado no Código Fonte (*Listing*) 2.

Listing 2 Exemplo de objeto RANDeployer

```
1  apiVersion: oai.unisinos/v1beta1
2  kind: RANDeployer
3  metadata:
4    name: ngran-1
5    namespace: oai
6  spec:
7    coreIP: 10.233.105.56
8    cuNode: node3
9    duNode: node4
10   ruNode: node2
```

O RANPlacer recebe informações similares as do recurso RANDeployer. O IP do *Core* da rede precisa ser fornecido, pois é utilizado para criar os RANDeployers. O número de RANDeployers criados varia de acordo com o que é fornecido no campo *rus*. O campo *rus*

é um vetor que recebe o nome de um RANDeployer e o posicionamento da RU. Além disso, o RANDeployer espera receber as informações da topologia da rede, que deve ser criada no *cluster* como um *ConfigMap* e seu nome fornecido no campo *topologyConfig*. Um exemplo da descrição do *RANPlacer* pode ser observado no Código Fonte (*Listing*) 3.

Listing 3 Exemplo de objeto RANPlacer

```
apiVersion: oai.unisinos/v1beta1
kind: RANPlacer
metadata:
  name: placer-sample
  namespace: oai
spec:
  coreIP: 10.233.105.76
  topologyConfig: "topology"
  rus:
  - ranDeployerName: "1"
    ruNode: "node6"
  - ranDeployerName: "2"
    ruNode: "node7"
  - ranDeployerName: "3"
    ruNode: "node8"
```

É importante destacar, que todas as implementações desenvolvidas para este trabalho estão disponíveis no GitHub[3].

4.3 Ambiente de Avaliação

O protótipo descrito na Seção 4.2 foi avaliado em um ambiente experimental. A Subseção 4.3.1 apresenta a configuração do ambiente, descrevendo os recursos computacionais disponíveis e configuração do K8S. Além disso, a Subseção 4.3.2 descreve a topologia utilizada para validação e as variações de banda e latência.

4.3.1 Recursos Computacionais

Para validar o protótipo desenvolvido foi criado um *cluster* K8S na versão 1.19, com dezesseis nós hospedados em VMs. Dentre os dezesseis nós, dois são utilizados como nós mestre e são reservados para execução de funções do sistema, um nó é utilizado para a execução do núcleo da rede e os treze restantes são disponibilizados para o posicionamento de funções de

[3]https://github.com/juliorenner/oai-k8s

rede. Os nós utilizados são VMs homogêneas com dois núcleos de processamento, 4GB de memória RAM e uma interface de rede. O *cluster* kubernetes configurado utiliza Calico como ferramenta para o gerenciamento de rede. O Calico é uma solução de virtualização e segurança de rede de código aberto para contêineres, VMs e cargas de trabalho baseadas em servidores físicos (Tigera Inc., 2020).

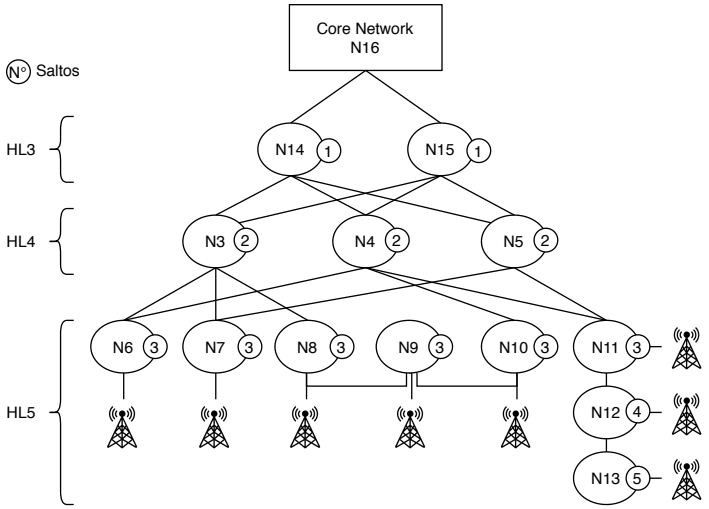
Além das máquinas utilizadas para o *cluster*, uma VM adicional foi provisionada. Nela foi configurado o Docker Registry, que armazena as imagens de contêineres com os arquivos executáveis e dependência para execução do OAI e Free5GC. Essa VM é acessível por todos os nós do *cluster* e possui uma configuração diferente das demais. Essa utiliza a mesma configuração de processamento, porém possui 8GB de memória. Exclusivamente para a execução dos testes de avaliação da utilização de recursos das funções da RAN, três novos nós foram adicionados ao *cluster* K8S. Os nós adicionados possuem 4GB de memória RAM e quatro núcleos de processamento. Esses nós não foram utilizados para outras validações executadas. As máquinas virtuais utilizadas para hospedar os nós do *cluster* K8S estão em um ambiente compartilhado composto por 16 servidores *blade* Dell PowerEdge M620, com dois processadores Inter(R) Xeon(R) CPU (Central Processing Unit) E5-2650 @2.00GHz com 8 núcleos (Intel Corporation, 2012), 288 GB de memória RAM DDR3 com frequência de 1333 MHz e dois adaptadores de rede Broadcom de 10 Gb. Como plataforma de virtualização foi utilizado o *hypervisor* VMware ESXI 6.7. Esse ambiente está fisicamente interligado, para prover redundância, em dois switches Dell PowerConnect M8024.

4.3.2 Topologia

Para a arquitetura desenvolvida a topologia de rede é fornecida como parâmetro de entrada para o algoritmo de posicionamento. A Figura 10 apresenta a topologia aplicada ao trabalho. Cada *N* representa um nó do *cluster* K8S. Além dos nós definidos na topologia, dois nós são exclusivos para a execução das funções de gerência do K8S, sendo os equivalentes ao *N1* e *N2*, que não estão incluídos na topologia.

A topologia utilizada é baseada no projeto PASSION (PASSION Project, 2020). A mesma é dividida em níveis hierárquicos (HL - *Hierarchical Level*), sendo esses compostos por cinco níveis: HL1, HL2, HL3, HL4 e HL5. Para o trabalho proposto, apenas os níveis HL3, HL4 e HL5 são aplicados, pois o nível HL3, conecta-se ao núcleo de rede (PASSION Project, 2020). No trabalho, os níveis hierárquicos são representados por número de saltos entre o núcleo e cada nó, para assim diferenciar o custo de cada alocação da RAN. Nativamente, o K8S não possui tal ciência de topologia, de forma que, os níveis hierárquicos e o número de saltos não são levados em consideração para o posicionamento. Embora a hierarquia apresente conexão direta entre os nós, a distância e o número de salto da rede de transporte possui variações de valores. Desta forma, a Tabela 3, fundamentada em (PASSION Project, 2020), apresenta a distância e o número de saltos da rede de transporte considerando os níveis da topologia. Divididos em máximo, mínimo, médio e desvio padrão. Tais valores possuem relação direta com a latência

Figura 10: Topologia de validação



Fonte: Elaborada pelo autor

dos links entre os nós.

Tabela 3: Dados da topologia do Projeto PASSION

Links entre HLs	Média (Km)	Média (saltos)	Máximo (Km)	Máximo (saltos)	Mínimo (Km)	Mínimo (saltos)	Desvio Padrão (Km)	Desvio Padrão (Saltos)
HL5 - HL4	24,04	8,81	64,70	24,00	2,33	2,00	11,56	5,06
HL5 - HL3	23,20	11,07	51,20	25,00	3,25	3,00	8,62	4,61
HL4 - HL3	24,73	7,17	36,80	12,00	11,25	4,00	7,10	3,92
HL5 - HL5	1,6	1,99	5,88	4	0	1	1,16	1,53
HL4 - HL4	8,69	2	22	2	0,5	2	5,98	0,00
HL3 - HL3	13.10	2,68	36,07	7	1,61	2	8,84	2,71

Para a avaliação consistente do trabalho desenvolvido, foram levados em consideração que a rede de transporte é constituída de dispositivos ópticos e de encaminhamento de pacotes (*switches* e roteadores). Tal decisão, é de extrema relevância para definição da capacidade (em Mbps) e latência (em milisegundos) de cada link. Para definição dos links, devido a limitação do ambiente virtualizado foram definidas as capacidades entre os links de 300Mbps, 450Mbps, 600Mbps, 1200Mbps e 1500Mbps. Para definição da latência, diferentes parâmetros são utilizados para a definição do valor, tais como (i) a latência de propagação nos links é 0,005 ms/Km, (ii) a latência dos dispositivos de encaminhamento é de 0,05 ms por dispositivo e (iii) a latência dos dispositivos ópticos é divida em duas etapas: (a) 0,000725 ms por salto por nó de transporte e (b) 0,03 ms por link. Dessa forma, com a variação dos parâmetros de capacidade e latência dos links (relação entre a Tabela 3, capacidade dos links e latência dos dispositivos de transporte) foram idealizadas três configurações diferentes aplicadas a mesma topologia para validação da arquitetura e algoritmo de posicionamento desenvolvido. A Tabela 4 apresenta os valores aplicados a cada cenário de validação.

Tabela 4: Parâmetros de capacidade e latência

Links entre HLs	Banda Máxima versus Latência Mínima	Banda Aleatória versus Latência Désvio Padrão	Banda Mínima versus Latência Aleatória
HL5 - HL4	600Mbps - 0,16 0,175ms	300 600Mbps - 0,167 0,22ms	300Mbps - 0,25 1ms
HL5 - HL5	600Mbps - 0,16 0,175ms	300 600Mbps - 0,167 0,22ms	300Mbps - 0,25 1ms
HL4 - HL3	1500Mbps - 0,22ms	1200 1500Mbps - 0,199ms	1200Mbps - 2 3ms
HL3 - CN	1500Mbps - 2ms	1200 1500Mbps - 2ms	1200Mbps - 2 3ms

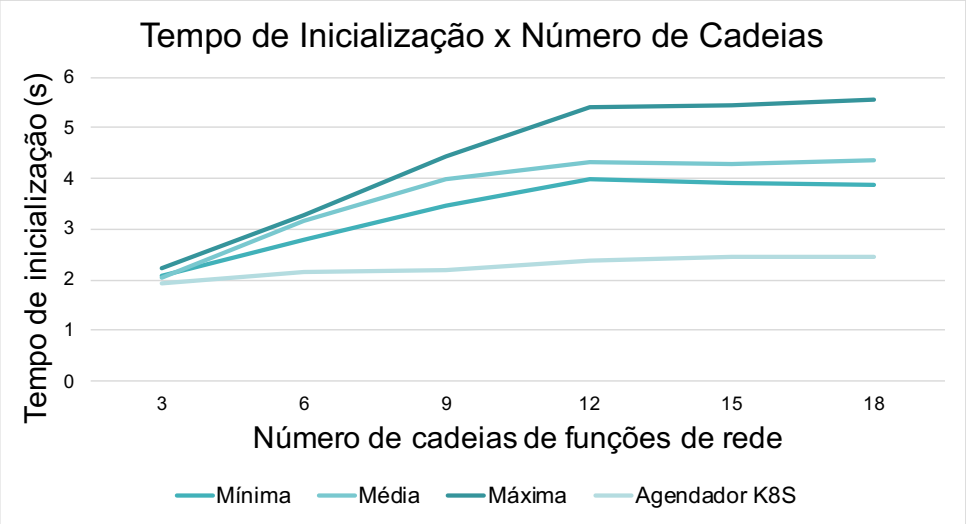
5 RESULTADOS

Os valores medidos para a avaliação foram: (i) Tempo de Inicialização, i.e., tempo percorrido entre a criação do recurso no K8S até o momento em que o *Pod* (Função de Rede) foi iniciada; (ii) Número de Saltos, i.e., número médio de saltos que cada cadeia de funções de rede possui após o posicionamento; (iii) Percentual de alocação, i.e., o percentual de funções de rede que foram posicionadas com sucesso; e (iv) Utilização de Recursos Computacionais, que refere-se a utilização de CPU e de Memória das funções de rede virtualizadas. Para avaliar as métricas descritas foram executados repetidos testes com um número crescente de cadeias de função de rede, que devem ser posicionadas e foram utilizadas três configurações de topologia baseadas na Tabela 4: (i) *Máxima* que possui *links* com maior banda e menor latência, (ii) *Média* que possui links com banda e latência distribuídas entre os maiores valores e os menores, e (iii) *Mínima* que possui *links* com menor banda e maior latência.

Para cada configuração da topologia de rede, foram posicionadas 3, 6, 9, 12, 15 e 18 cadeias de funções de rede. Além desses três cenários de validação utilizando o algoritmo de posicionamento, foram posicionadas o mesmo número de funções de rede com o agendador nativo do K8S com fins de comparação. Cada combinação de posicionamento (configuração da topologia e número de cadeias de funções de rede) foi executada 30 vezes e os valores médios das métricas coletadas foram considerados. Finalmente, foi definido o posicionamento das *RUs*, mantendo o mesmo posicionamento em todos os cenários. Na topologia exibida, a *RU* pode ser posicionada entre o intervalo de nós N6 e N13, por possuírem os equipamentos de rádio frequência. O posicionamento foi feito sequencialmente, posicionando uma RU por nó e reiniciando no N6 após todos receberem uma *RU*.

A diferença no tempo de inicialização do posicionamento utilizando o agendador do K8S foi menor para todos os posicionamentos das funções de rede, como pode ser observado na Figura 11. Quando o agendador do K8S posiciona os *Pods* criados existe um processo para posicionamento que envolve filtrar os nós com os requisitos necessários e, de acordo com os critérios nativos, avaliar o melhor nó disponível para o posicionamento. Entretanto, no posicionamento da *PlaceRAN* esse filtro é feito pelo algoritmo, além de outras avaliações da topologia e de requisitos de latência. O algoritmo precisa percorrer os nós da topologia, que são representados por um grafo, para encontrar os caminhos para posicionar as funções de rede. Neste caso, o

Figura 11: Tempo de Inicialização



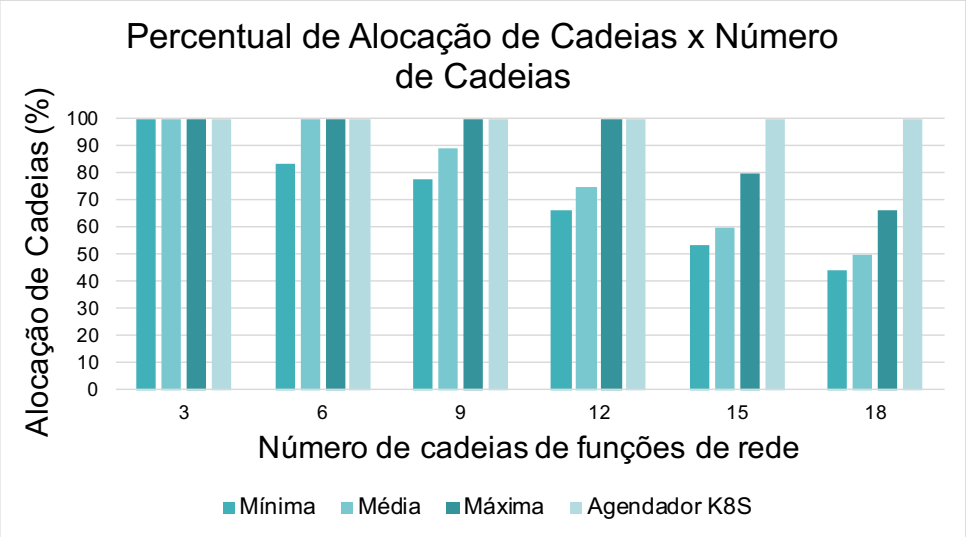
Fonte: Elaborada pelo autor

algoritmo BFS foi utilizado e consequentemente existe aumento do tempo de execução. Além disso, o *PlaceRAN* cria recursos *RANDeployer* que inicializam as funções nos nós selecionados. Mesmo com os nós já fornecidos, quando o *Pod* for criado esse passa pelo agendador nativo do K8S, para que o posicionamento seja executado, i.e., a diferença é que o nó já foi selecionado anteriormente. Ainda assim, o agendador do K8S valida que os recursos necessários estão realmente disponíveis para enviar a solicitação de criação do *Pod* para o nó. Portanto, além da adição de tempo proporcionada pela execução do algoritmo, boa parte do processo padrão ainda é executado no posicionamento do *PlaceRAN*.

Outro ponto a ser considerado na Figura 11 é o tempo de inicialização dos posicionamentos que utilizam topologia se mantém estáveis a partir do posicionamento de 12 cadeias de funções de rede. Esse comportamento pode ser melhor compreendido analisando a Figura 12, onde são exibidos os percentuais de sucesso na alocação das cadeias de funções de rede. Pode-se observar a partir de 12 cadeias posicionadas que o percentual de alocação diminui, mostrando que o número máximo de cadeias que podem ser posicionadas foi atingido. Dessa forma, o tempo de inicialização se torna contínuo, pois a quantidade de cadeias posicionadas permanece a mesma. É importante destacar que o percentual de posicionamento exercido pelo agendador do K8S permanece sempre em 100%, porém, o agendador do K8S não considera os requisitos de rede e por não haverem limitações as funções de rede sempre são posicionadas. Em outras palavras, existem recursos computacionais, mas as cadeias não são posicionadas de forma que cumpram os requisitos de rede estabelecidos, tornando o serviço de RAN desagregada não funcional.

A principal métrica para avaliar a efetividade do algoritmo de posicionamento utilizado é o número de saltos entre as funções de rede da mesma cadeia. A fim de comparar a efetividade do algoritmo, as cadeias de rede foram posicionadas utilizando o agendador nativo do K8S, que não respeita os requisitos de rede definidos, porém, esse detalhe foi desconsiderado para fins

Figura 12: Percentual de Alocação de Cadeias de Função de Rede



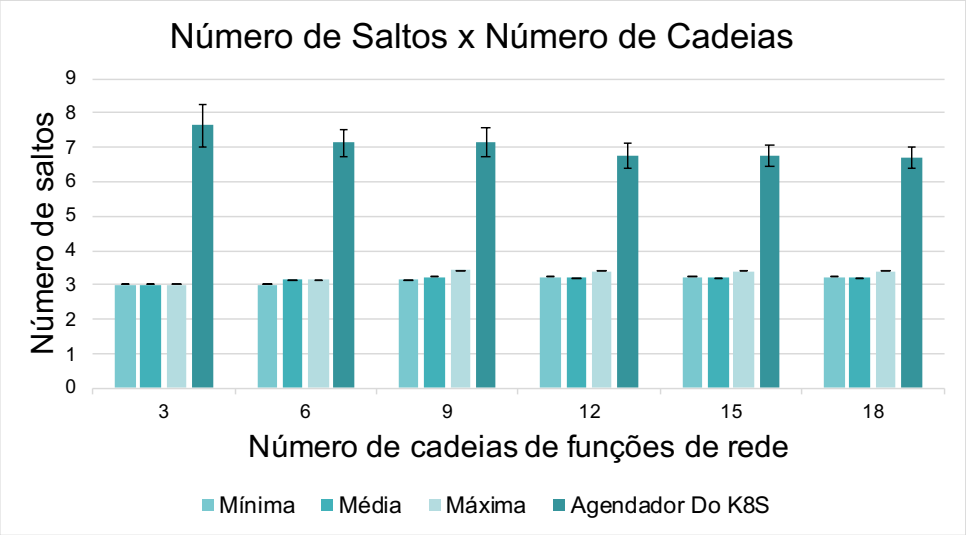
Fonte: Elaborada pelo autor

de comparação. Após o posicionamento das *RUs* definido pelo OR, o número de saltos foi obtido considerando a topologia em questão, possibilitando a comparação com o posicionamento com topologia. A Figura 13 compara o número de saltos médio obtido para uma cadeia no posicionamento de cada cenário avaliado. Pode-se observar que para as cadeias posicionadas pelo agendador do K8S, que não tem o conhecimento da topologia e demais informações de entrada, as cadeias de funções de rede posicionadas possuem aproximadamente 4 saltos a mais, um percentual de aproximadamente 230%, destacando que esse número tende a ser ainda maior para topologias com mais nós. Além disso, é possível visualizar a margem de erro utilizando o desvio padrão em cada uma das barras. Nos casos das barras de erro relacionadas as execuções que consideram a topologia em questão, os resultados obtidos para todas as execuções foi o mesmo, i.e., não há desvio padrão. Já as execuções de posicionamento do K8S possuem taxas de erro que não exibem grandes variações.

Para avaliar a utilização de recursos computacionais, três nós adicionais foram inseridos no *cluster* K8S, exclusivamente para essa avaliação. Com os três nós adicionais, o recurso *RANDeployer* foi utilizado e cada função de rede (*CU*, *DU* e *RU*) foi agrupada em um nó. Dessa forma, pode-se analisar como cada função de rede se comporta a medida que escala. As Figuras 14a e 14b exibem o consumo de CPU e memória, respectivamente, a medida que o número de funções de rede em cada nó aumenta. Como mencionado na Subseção 4.2, as funções de rede *CU* e *DU* utilizam a mesma imagem de contêiner, o que justifica que ambas tenham métricas de consumo mais próximas. Além disso, pode-se observar que ambas as funções possuem uma demanda maior por recursos de processamento se comparado ao consumo dos recursos de memória, que tende a ser mais estável.

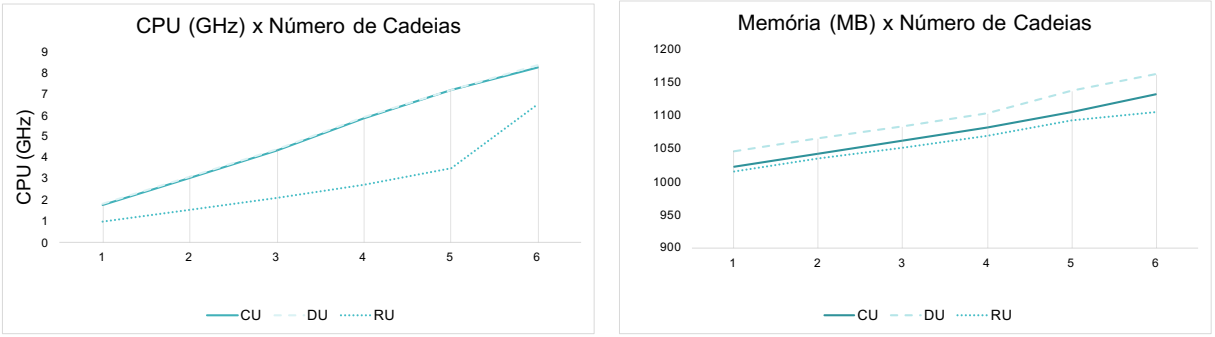
Sumarizando os resultados obtidos, no que se refere ao tempo de inicialização, por adicionar uma camada a mais de processamento para determinar o posicionamento, as customizações

Figura 13: Número Médio de Saltos



Fonte: Elaborada pelo autor

Figura 14: Utilização de Recursos



(a) Utilização de CPU

(b) Utilização de Memória

Fonte: Elaborada pelo autor

validadas aumentam o tempo para inicializar as cadeias de funções de rede. No número de saltos os valores do K8S ficam entre 6 e 7 saltos, enquanto com o algoritmo de posicionamento esse valor é reduzido para uma média de 3 saltos, o que para uma topologia pequena é uma diferença significativa. No percentual de alocação, a medida que o número de cadeias que devem ser posicionadas aumentam, o percentual de alocação diminui, resultado da capacidade e latência das conexões da topologia. Finalmente, o consumo de recursos mostra um aumento linear conforme o número de funções de rede são posicionadas em um nó, destacando que CPU tem uma utilização mais alta se comparado a memória. Portanto, é possível concluir que a arquitetura desenvolvida se mostrou eficaz na orquestração das funções virtualizadas da RAN.

6 CONCLUSÃO

O posicionamento e gerenciamento das funções de rede são etapas fundamentais para a virtualização dos componentes das redes 5G. Nesse contexto, a arquitetura desenvolvida mostrou bons resultados para a otimização de um conjunto de funções de rede através da utilização do padrão de operadores da ferramenta K8S. Como demonstrado nos resultados, o número de saltos diminuiu consideravelmente utilizando as adaptações que possuem conhecimento sobre a topologia e os requisitos da rede. Além do posicionamento, o padrão de operadores também foi utilizado para automatizar todo o ciclo de vida das funções de rede, o que é peça chave para reduzir os custos de operação da rede.

Os operadores desenvolvidos atualmente controlam apenas as etapas de criação, configuração e deleção das funções de rede. Entretanto, podem ser utilizados para automatizar outras tarefas necessárias para ambientes em nuvem e específicas dos cenários da 5G, as possibilidades são imensas. Além da extensão dos operadores outros cenários relacionados ao posicionamento de funções de rede podem ser explorados utilizando o K8S. Dentre eles tem-se o posicionamento de funções de rede sob-demanda, onde o posicionamento não é feito considerando um grupo de funções de rede, mas apenas uma, e a otimização de funções de rede já existentes, onde o melhor posicionamento para as funções de rede existentes é determinado e ocorre a migração das mesmas para diferentes nós, atendendo as mudanças de requisitos dos usuários. Além disso, a avaliação de novas métricas como latência, *jitter* e vazão também podem ser consideradas. O protótipo prova que a ferramenta K8S pode ser facilmente estendida para lidar com os cenário das redes móveis, o que abre um grande leque de possibilidades para sua utilização nesse contexto.

Referências

3GPP. Study on New Radio Access Technology; Radio Access Architecture and Interfaces (Release 14). n. 38.801, 2017.

AGRAWAL, R. et al. Cloud RAN challenges and solutions. **Annals of Telecommunications**, v. 72, n. 7-8, p. 387–400, 2017.

AL-DULAIMI et al. **5G Networks: Fundamental Requirements, Enabling Technologies, and Operations Management**. [S.l.]: John Wiley & Sons, 2018.

AL-DULAIMI, A.; WANG, X.; I, C.-L. **5G Networks: Fundamental Requirements, Enabling Technologies, and Operations Management**. [S.l.: s.n.], 2018.

ALLIANCE, O. S. **OpenAirInterface**. 2020. <https://www.openairinterface.org/>. [Último acesso: 26-abril-2020].

BERNARDOS CJ., R. A. Z. J. C. L. A. P.; LYNCH, P. **Network Virtualization Research Challenges**. 2019. <https://www.rfc-editor.org/info/rfc8568>. [Último acesso: 04-julho-2020].

BERTENYI, B. et al. NG radio access network (NG-RAN). **Journal of ICT Standardization**, River Publishers, v. 6, n. 1, p. 59–76, 2018.

CARDOSO, K. V. et al. **A softwarized perspective of the 5G networks**. 2020.

CHEN, T. et al. Software defined mobile networks: concept, survey, and research directions. **IEEE Communications Magazine**, v. 53, n. 11, p. 126–133, 2015.

Cloud Native Computing Foundation. **Kubernetes Documentation | Kubernetes**. 2020. Disponível em: <https://kubernetes.io/docs/home/>.

DALLA-COSTA, A. G. et al. Orchestra: A customizable split-aware nfv orchestrator for dynamic cloud radio access networks. **IEEE Journal on Selected Areas in Communications**, IEEE, v. 38, n. 6, p. 1014–1024, 2020.

DOBIES, J.; WOOD, J. **Kubernetes Operators - Automating the container orchestration platform**. [S.l.]: o'reilly, 2020.

DZOGOVIC, B. et al. Connecting Remote eNodeB with Containerized 5G C-RANs in OpenStack Cloud. In: IEEE. **6th IEEE International Conference on Cyber Security and Cloud Computing (CSCloud)**. [S.l.], 2019. p. 14–19.

ERIKSSON, M. Placement of services in distributed clouds. **Open Source MANO, PoC Overview**, 2017.

ETSI, G. Digital cellular telecommunications system (Phase 2+) (GSM); Universal Mobile Telecommunications System (UMTS); LTE; 5G; Release description;Release 15 (3GPP TR 21.915 version 15.0.0 Release 15). 2017.

FONSECA, F. F. et al. Optimizing allocation and positioning in a disaggregated radio access network. In: SBC. **Anais do XXXVII Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos**. [S.l.], 2019. p. 791–804.

FOUNDATION, T. L. **Kubernetes Documentation**. 2020. Disponível em: <https://kubernetes.io>.

GARCIA-SAAVEDRA et al. Wizhaul: On the centralization degree of cloud ran next generation fronthaul. **IEEE Transactions on Mobile Computing**, IEEE, v. 17, n. 10, p. 2452–2466, 2018.

GARCIA-SAAVEDRA, A. et al. Fluidran: Optimized vran/mec orchestration. In: **INFOCOM 2018-IEEE Conference on Computer Communications**. [S.l.: s.n.], 2018. p. 2366–2374.

GAVRILOVSKA, L. et al. From Cloud RAN to Open RAN. **Wireless Personal Communications**, Springer, p. 1–17, 2020.

HABIBI, M. et al. A Comprehensive Survey of RAN Architectures Toward 5G Mobile Communication System. **IEEE Access**, v. 7, p. 70371–70421, 2019.

Intel Corporation. **Processador Intel® Xeon® E5-2650 (cache de 20M, 2,00 GHz, Intel® QPI com 8,00 GT/s) Product Specifications**. 2012. Disponível em: <https://ark.intel.com/content/www/br/pt/ark/products/64590/intel-xeon-processor-e5-2650-20m-cache-2-00-ghz-8-00-gt-s-intel-qpi.html>.

LUO, Y.-C. et al. A computation workload characteristic study of C-RAN. In: **IEEE 38th International Conference on Distributed Computing Systems**. [S.l.: s.n.], 2018. p. 1599–1603.

MAMUSHIANE, L. et al. Overview of 9 Open-Source Resource Orchestrating ETSI MANO Compliant Implementations: A Brief Survey. In: IEEE. **2019 IEEE 2nd Wireless Africa Conference (WAC)**. [S.l.], 2019. p. 1–7.

MARSCH, P. et al. **5G system design: architectural and functional considerations and long term research**. [S.l.]: John Wiley & Sons, 2018.

MARSCH ÖMER BULAKçI, O. Q. M. B. P. **5G System Design: Architectural and Functional Considerations and Long Term Research**. [S.l.: s.n.], 2018.

MOLNER, N. et al. Optimization of an integrated fronthaul/backhaul network under path and delay constraints. **Ad Hoc Networks**, Elsevier, v. 83, p. 41–54, 2019.

NIKAEIN, N.; CHANG, C.-Y.; ALEXANDRIS, K. Mosaic5G: Agile and flexible service platforms for 5G research. **ACM SIGCOMM Computer Communication Review**, ACM New York, NY, USA, v. 48, n. 3, p. 29–34, 2018.

NOVAES, C. et al. Virtualized C-RAN Orchestration with Docker, Kubernetes and OpenAirInterface. **arXiv preprint arXiv:2001.08992**, 2020.

OGBUACHI, M. C. et al. Context-Aware K8S Scheduler for Real Time Distributed 5G Edge Computing Applications. 2019.

OpenAirInterface Software Alliance. **Soft Modem Build Script**. 2020. Disponível em: <https://gitlab.eurecom.fr/oai/openairinterface5g/-/blob/master/doc/BUILD.md>.

PASSION Project. **Photonic technologies for progrAmmable transmission and switching modular systems based on Scalable Spectrum/space aggregation for future agIle high capacity metrO Networks**. 2020. Disponível em: <http://www.passion-project.eu/project/>.

Red Hat. **What is container orchestration?** 2020. Disponível em: <https://www.redhat.com/en/topics/containers/what-is-container-orchestration>.

SEUNG-QUE, L. et al. Virtualization and orchestration of radio access network. In: IEEE. **IEEE International Conference on Communication Systems**. [S.l.], 2018. p. 321–325.

SHAH, D. D. J. Building Modern Clouds: Using Docker, Kubernetes Google Platform. 2019.

TAYQ, Z. **Fronthaul integration and monitoring in 5G networks**. Tese (Doutorado), 2017.

The Apache Software Foundation. **Apache Mesos - Documentation Home**. 2020. Disponível em: <https://mesos.apache.org/documentation/latest/index.html>.

Tigera Inc. **About Calico**. 2020. Disponível em: <https://docs.projectcalico.org/about/about-calico>.