Pós-Graduação em Ciência da Computação

Francisco Edigleison da Silva Barbosa

Edge-On: Uma plataforma multicamadas com suporte ao processamento e compartilhamento de conteúdo na borda da rede para a Internet de Veículos

Universidade Federal de Pernambuco
posgraduacao@cin.ufpe.br
http://cin.ufpe.br/~posgraduacao

Recife
2019

Francisco Edigleison da Silva Barbosa

Edge-On: Uma plataforma multicamadas com suporte ao processamento e compartilhamento de conteúdo na borda da rede para a Internet de Veículos

Trabalho apresentado ao Programa de Pós-graduação em Ciência da Computação do Centro de Informática da Universidade Federal de Pernambuco como requisito parcial para obtenção do grau de Mestre em Ciência da Computação.

Área de Concentração: Redes de Computadores e Sistemas Distribuídos
Orientador: Kelvin Lopes Dias

Recife
2019

Catalogação na fonte
Bibliotecária Mariana de Souza Alves CRB4-2105

B238e Barbosa, Francisco Edigleison da Silva
Edge-On: uma plataforma multicamadas com suporte ao processamento e compartilhamento de conteúdo na borda da rede para a Internet de Veículos/ Francisco Edigleison da Silva Barbosa – 2019.
105 f., fig.; tab.

Orientador: Kelvin Lopes Dias.
Dissertação (Mestrado) – Universidade Federal de Pernambuco. CIn, Ciência da Computação. Recife, 2019.
Inclui referências e apêndices.

1. Redes de Computadores e Sistemas Distribuídos. 2. Internet de Veículos. 3. Virtualização de Funções de Rede. 4. Redes Definidas por Software. I. Dias, Kelvin Lopes (orientador). II. Título.

004.6 CDD (22. ed.) UFPE-CCEN 2020-06

Francisco Edigleison da Silva Barbosa

"**Edge-On:** Uma plataforma multicamadas com suporte ao processamento e compartilhamento de conteúdo na borda da rede para a Internet de Veículos"

Dissertação de Mestrado apresentada ao Programa de Pós-Graduação em Ciência da Computação da Universidade Federal de Pernambuco, como requisito parcial para a obtenção do título de Mestre em Ciência da Computação.

Aprovado em: 06 de setembro de 2019.

BANCA EXAMINADORA

Prof. Dr. Nelson Souto Rosa
Centro de Informática / UFPE

Prof. Dr. Reinaldo Cézar de Morais Gomes
Departamento de Sistemas e Computação / UFCG

Prof. Dr. Kelvin Lopes Dias
Centro de Informática / UFPE
(Orientador)

À minha família, por sua capacidade de acreditar em mim e investir em mim. Mãe, seu cuidado e dedicação foi que deram, em alguns momentos, a esperança para seguir. Pai e irmãos, a presença de vocês significou segurança e certeza de que não estou e nunca estive sozinho nessa caminhada.

AGRADECIMENTOS

Se você está lendo esta página é porque eu consegui. E não foi fácil chegar até aqui. Do processo seletivo, passando pela aprovação até a conclusão do Mestrado, foi um longo caminho percorrido. Nada foi fácil, nem tampouco tranquilo. ("A sola do pé conhece toda a sujeira da estrada"). Contudo, dificuldades maiores foram vencidas bem antes de tudo isso.

Quero agradecer a todos aqueles que sempre confiaram em mim, desde sempre. À minha família e aos meus verdadeiros amigos.

"E aprendi que se depende sempre
De tanta, muita, diferente gente
Toda pessoa sempre é as marcas
das lições diárias de outras tantas pessoas.
É tão bonito quando a gente entende
Que a gente é tanta gente
Onde quer que a gente vá.
É tão bonito quando a gente sente
Que nunca está sozinho
Por mais que pense estar..." (Caminhos do coração – Gonzaguinha.)

Devo um agradecimento enorme a todos os membros da minha família pelo incentivo nessa minha caminhada, meu pai, meus irmãos, mas minha mãe tem um espaço especial em minha trajetória acadêmica. Mãe nunca irei esquecer tudo o que a senhora fez e faz por mim. Obrigado!

Agradeço ao meu professor e orientador Kelvin Dias, por acreditar e apostar na minha capacidade e no potencial deste trabalho, proporcionando meios para torná-lo um trabalho científico. Agradeço por toda dedicação, disponibilidade e principalmente a paciência no decorrer de todo o mestrado. Também não poderia deixar de agradecer pela ajuda, dicas e correções nesse percurso, ao qual foi de suma importância para minha evolução pessoal e profissional.

Agradeço especialmente aos meus amigos de laboratório e de universidade, que por muitas vezes debatemos sobre idéias relacionadas a essa dissertação e a temas diversos, levando sempre aquele local alegria e diversão.

Agradeço a todos os professores do CIn pelo conhecimento compartilhado e pelo exemplo de dedicação e comprometimento.

Muito obrigado pela ajuda de todos, pois sem esse apoio não seria capaz de chegar onde cheguei.

RESUMO

As redes veiculares (*Vehicular Ad Hoc Networks - VANETs*) evoluíram dos modos de comunicação tradicionais denominados veículo para veículo (*Vehicle-to-Vehicle - V2V*) e veículo para infraestrutura (*Vehicle-to-Infrastructure - V2I*) para a chamada Internet de veículos (*Internet of Vehicles - IoV*), permitindo comunicação veículo para tudo (*Vehicle-to-Everything - V2X*) através da sinergia com a computação em nuvem e a Internet das coisas (*Internet of Things - IoT*). Estima-se que haverá mais de 300 milhões de carros conectados até 2022, gerando mais de 400 *PetaBytes* de dados nos mais diversos tipos de redes móveis. Dessa forma, surgem vários desafios no projeto de arquiteturas e mecanismos que viabilizem o compartilhamento de conteúdo de maneira eficiente, bem como o suporte aos requisitos de baixa latência nas comunicações para tomadas de decisão em aplicações focadas em carros conectados e segurança dos passageiros e pedestres. Neste sentido, por um lado o paradigma de redes definidas por *software* (*Software Defined Networking - SDN*) tem sido empregado para permitir melhor gerenciamento, flexibilidade e programabilidade da rede veicular; por outro, computação na borda (*Edge Computing*) e na névoa (*Fog Computing*) podem auxiliar na redução da latência, permitindo armazenamento e processamento mais próximos do usuário. Contudo, soluções utilizando estes e outros blocos construtores das redes 5G ainda são incipientes diante dos requisitos para a efetiva operação e implantação de serviços para a IoV. Esta dissertação propõe uma plataforma hierárquica denominada Edge-On para processamento de dados de ponta a ponta, que permite a replicação na borda da rede dos conteúdos mais requisitados, distribuindo-os tanto na camada (*Multi-access Edge Computing - MEC*), quanto nos próprios veículos por meio de um modelo de (*On Board Unit - OBU*) e Fog baseados em um plano de dados *OpenFlow* e com suporte a *containers*. Edge-On considera camada de nuvem remota *OpenStack*, controlador SDN e *switches* como funções de rede virtuais (*Virtual Network Function - VFN*) , camada intermediária MEC na borda da rede e próxima às (*Road-Side Units - RSU*), provendo comunicação via APIs, bem como orquestração Docker *Swarm* e (*Network Function Virtualization - NFV*). O modelo em camadas e de orquestração propostos viabilizam a criação de templates e execução dinâmica de aplicações em microsserviços. A camada Fog nos veículos considera uma estratégia que permite compartilhamento de conteúdo entre os veículos via protocolo Torrent (*Peer-to-Peer - P2P*), sem a necessidade de infraestrutura, tolerante a falhas e com suporte à mobilidade. Avaliamos a proposta em cenários de transferência de arquivos e compartilhamento de conteúdo. Os resultados são promissores e indicam que a arquitetura pode lidar com os requisitos rigorosos em termos de latência para aplicações IoV.

Palavras-chaves: Internet de Veículos. Virtualização de Funções de Rede. Redes Definidas por Software. Edge Computing. Fog Computing. Container.

ABSTRACT

*Vehicular Ad Hoc Network*s (VANETs) have evolved from traditional *Vehicle-to-Vehicle* (V2V), and *Vehicle-to-Infrastructure* (V2I) communications to the so-called *Internet of Vehicles* (IoV) enabled by the integration with cloud computing and *Internet of Things* (IoT). Reports estimate that there will exist over 300 million connected cars by 2022, producing over 400 PetaBytes of data across a wide range of mobile networks. Such a scenario creates several challenges to design the architectures and mechanisms to enable efficient data dissemination such as content sharing. They also need to support low latency communications for decision making in applications focused on connected cars and the safety of passengers and pedestrians. On the one hand, the *Software Defined Networking* (SDN) has been employed to allow better management, flexibility, and programmability of both wired and vehicular network; on the other hand, Edge Computing and Fog Computing can help reduce latency, deploying storage and processing closer to end devices. However, solutions using these and other building blocks of 5G networks are still incipient given the requirements for effective operation and deployment of services for IoV. This dissertation proposes a hierarchical platform called Edge-On for end-to-end data processing, which allows the replication of the most requested content at the edge of the network, distributing it both in the *Multi-access Edge Computing* (MEC) layer and in the vehicles themselves through an *On Board Unit* (OBU) model and a Fog based on an OpenFlow data plan and with container support. Edge-On considers OpenStack remote cloud layer, SDN controller and switches to be *Virtual Network Function* (VNF), MEC mid-tier at the network edge and near *Road-Side Units* (RSU), providing communication via APIs, as well as Docker Swarm orchestration and *Network Function Virtualization* (NFV). The proposed layered model and orchestration enable the creation of templates for services and applications as well as the dynamic execution of microservices. The fog layer in vehicles allows infrastructure-independent, fault-tolerant content sharing between vehicles via *Peer-to-Peer* (P2P) protocol with mobility support. We evaluated the proposal in both file transfer and content sharing scenarios. The results are promising and indicate that the architecture can handle stringent latency requirements for IoV applications.

Keywords: Internet of Vehicles. Network Function Virtualization. Software Defined Networking. Edge Computing. Fog Computing. Container.

LISTA DE ILUSTRAÇÕES

Figura 1 – Cenário geral para Redes Veículares. 22
Figura 2 – Referência da arquitetura MEC em um ambiente NFV 29
Figura 3 – Arquitetura alto nível de uma fog computacional, edge e nuvem 32
Figura 4 – Arquitetura NFV alto nível proposta pela ETSI. 36
Figura 5 – Visão simplificada da arquitetura SDN 39
Figura 6 – Especificação do *Switch OpenFlow* 40
Figura 7 – Comparação entre *containers* e Máquinas Virtuais 41
Figura 8 – Arquitetura Docker Swarm Alto Nível. 43
Figura 9 – Modelo de Alto Nível Edge-On 56
Figura 10 – Implementação do padrão NFV na arquitetura de nuvem 57
Figura 11 – Implementação NFV e container na arquitetura de MEC 58
Figura 12 – Migração/Replicação de Aplicações e Serviços 59
Figura 13 – Instanciação dos Serviços de Rede e das Aplicações na Nuvem 61
Figura 14 – Instanciação dos Serviços de Rede e das Aplicações na MEC 63
Figura 15 – Proposta de implementação da OBU 65
Figura 16 – Diagrama de sequência Edge-On 66
Figura 17 – Cenário Físico 73
Figura 18 – Resultados Preliminares 76
Figura 19 – Cenário de testes entre Nuvem e MEC 77
Figura 20 – Gráfico do tempo de resposta entre Nuvem vs MEC 79
Figura 21 – Tempo de Resposta entre Nuvem vs MEC 80
Figura 22 – Gráfico do tempo de resposta entre Nuvem vs MEC 82
Figura 23 – Tempo de requisição de uma página Web: Nuvem vs MEC 83
Figura 24 – Cenário de teste de Tranferência de arquivo 84
Figura 25 – Gráfico de tranferência de arquivo 85
Figura 26 – Arquitetura Node Car 97
Figura 27 – Arquitetura Tacker Alto Nível. 100
Figura 28 – Modelo de Alto nível do Funcionamento do Traefik. 102

LISTA DE TABELAS

Tabela 2 – Comparação dos Trabalhos Relacionados 54
Tabela 3 – Configuração de *Hardware* e *Software* do *testbed* 72
Tabela 4 – Parâmetros de Avaliação 73
Tabela 5 – Projeto de experimentos 74
Tabela 6 – Estatística descritiva em (ms) para o tempo de resposta entre Nuvem vs MEC 79
Tabela 7 – Estatística descritiva em (ms) para o tempo de resposta HTTP: Nuvem vs MEC 81
Tabela 8 – Estatística descritiva em (s) para o tempo de resposta de compartilhamento do conteúdo 85

LISTA DE ABREVIATURAS E SIGLAS

AP	*Access Point*
API	*Application Programming Interface*
BS	*Base Station*
CA	*Certification Authorities*
CDN	*Content Delivery Network*
DNS	*Domain Name Server*
eNB	*Evolved Node B*
ETSI	*European Telecommunications Standards Institute*
GPS	*Global Positioning System*
GRE	*Generic Routing Encapsulation*
IaaS	*Infrastructure as a Service*
IoT	*Internet of Things*
IoV	*Internet of Vehicles*
IP	*Internet Protocol*
ITS	*Intelligent Transportation System*
LIDAR	Light Detection And Ranging
LTS	Long Term Support
MANET	*Mobile Ad Hoc Network*
MEAO	*MEC Application Orchestrator*
MEC	*Multi-access Edge Computing*
MEO	*MEC Orchestrator*
ML	*Machine Learning*
MNO	*Mobile Network Operator*
NAT	*Network Address Translation*
NFV	*Network Function Virtualization*
NFVI	*NFV-Infrastructure*
NFVO	*NFV-Orchestrator*
NIST	*National Institute of Standards and Technology*

NS	*Network Service*
OBU	*On Board Unit*
ONF	*Open Networking Foundation*
P2P	*Peer-to-Peer*
PaaS	Platform as a Service
QoS	*Quality of Service*
RNIS	*Radio Network Information Service*
RSU	*Road-Side Units*
SaaS	*Software as a Service*
SDN	*Software Defined Networking*
SFC	*Service Function Chaining*
SLA	*Service Level Agreement*
SUMO	*Simulation of Urban Mobility*
TOF	*Traffic Offloading Function*
V2C	*Vehicle-to-Cloud*
V2D	*Vehicle-to-Device*
V2G	*Vehicle-to-Grid*
V2I	*Vehicle-to-Infrastructure*
V2P	*Vehicle-to-Pedestrian*
V2V	*Vehicle-to-Vehicle*
V2X	*Vehicle-to-Everything*
VANET	*Vehicular Ad Hoc Network*
VANETs	*Vehicular Ad Hoc Networks*
VFC	*Vehicular Fog Computing*
VIM	*Virtualized Infrastructure Manager*
VM	*Virtual Machine*
VN	*Veicular Networking*
VNF	*Virtual Network Function*
VNFM	*Network Functions Virtualization Manager*
VNFs	*Virtual Network Functions*
VSDN	*Vehicle Software Defined Networking*

WAVE *Wireless Access in Vehicular Environments*

LISTA DE SÍMBOLOS

γ	Letra grega Gama
$\in$	Pertence
δ	Delta
θ	Teta
σ	Sigma
μ	Mi

SUMÁRIO

1 INTRODUÇÃO 16
1.1 MOTIVAÇÃO 16
1.2 OBJETIVO GERAL 18
1.3 OBJETIVOS ESPECÍFICOS 19
1.4 ESTRUTURA DA DISSERTAÇÃO 19

2 FUNDAMENTAÇÃO TEÓRICA 20
2.1 VANETS 20
2.1.1 Aplicações VANETs 23
2.2 CLOUD COMPUTING 24
2.2.1 Características Essenciais 24
2.2.2 Modelos de Serviços 25
2.2.3 Modelos de Implantação 26
2.2.4 MEC - Multi-access Edge Computing 27
2.2.5 Fog Computing 31
2.3 NETWORK FUNCTIONS VIRTUALIZATION 34
2.3.1 Arquitetura de Referência NFV 35
2.4 SOFTWARE DEFINED NETWORKING 37
2.4.1 Protocolo OpenFlow 38
2.5 TECNOLOGIA DE *CONTAINERS* 41
2.5.1 Docker 42
2.6 REDES PEER-TO-PEER (P2P) 45
2.7 CONSIDERAÇÕES FINAIS 47

3 TRABALHOS RELACIONADOS 48
3.1 SOLUÇÕES BASEADAS EM MEC 48
3.2 SOLUÇÕES BASEADAS EM FOG 50
3.3 SOLUÇÕES INTEGRADAS MEC E FOG 52
3.4 CONSIDERAÇÕES FINAIS 54

4 PLATAFORMA PROPOSTA 55
4.1 DESCRIÇÃO DA PLATAFORMA 55
4.2 INSTANCIAÇÃO DOS SERVIÇOS DE REDE E DAS APLICAÇÕES NA NUVEM 60
4.2.1 Fluxos de instanciação dos serviços de rede e das aplicações na nuvem 60

4.3 FLUXOS DE INSTANCIAÇÃO DOS SERVIÇOS DE REDE E DAS APLICAÇÕES NA MEC . . . 62
4.4 COMUNICAÇÕES INTERVEICULAR E PROPOSTA DE UMA OBU PARA COMUNICAÇÃO E PROCESSAMENTO DE DADOS . . . 64
4.5 PSEUDOCÓDIGO DE OPERAÇÃO DA PLATAFORMA . . . 67
4.6 CONSIDERAÇÕES FINAIS . . . 68

5 AVALIAÇÃO EXPERIMENTAL . . . 70
5.1 SOFTWARES UTILIZADOS . . . 70
5.1.1 Especificação dos Equipamentos . . . 71
5.1.2 Ambiente de Testes . . . 72
5.2 MÉTRICAS E PARÂMETROS . . . 74
5.3 RESULTADOS E DISCURSSÕES . . . 75
5.3.1 Experimentos sem a proposta . . . 75
5.3.1.1 Tempo de Resposta . . . 75
5.3.2 Plataforma Edge-On . . . 76
5.3.2.1 Sinergia entre nuvem e MEC . . . 76
5.3.2.1.1 Tempo de Resposta: Nuvem vs MEC . . . 78
5.3.2.1.2 Requisição a uma aplicação web: Nuvem vs MEC . . . 80
5.3.2.2 Transferência de Arquivo . . . 82
5.4 CONSIDERAÇÕES FINAIS . . . 86

6 CONCLUSÃO . . . 87
6.1 CONSIDERAÇÕES FINAIS . . . 87
6.2 CONTRIBUIÇÕES . . . 88
6.3 TRABALHOS FUTUROS . . . 88

REFERÊNCIAS . . . 90

APÊNDICE A – FERRAMENTAS E AMBIENTES DE AVALIAÇÃO 96

APÊNDICE B – CÓDIGOS DAS APLICAÇÕES QUE FORAM EXECUTADAS EM CONTAINERS DOCKER . . . 103

1 INTRODUÇÃO

Este Capítulo descreve a motivação, introduzindo conceitos na área de IoV, tecnologias relacionadas e desafios que concernem à experiência da utilização de aplicações nesses ambientes, bem como a solução proposta. Em seguida, são apresentados o objetivo geral, os objetivos específicos e, por fim, a estrutura de capítulos desenvolvidos para esta dissertação.

1.1 MOTIVAÇÃO

Na era da IoT, os sensores estão espalhados por todas as cidades inteligentes, na indústria 4.0 e até em casas inteligentes, onde todos os aparelhos estão conectados para proporcionar mais conforto e comodidade aos moradores. Os sensores também estão dentro dos veículos. Eles monitoram o estado do motor (temperatura, nível de óleo e refrigeração), pressão dos pneus, o estado dos sistemas mecânicos, elétricos e hidráulicos (freios, aceleradores); e até mesmo o conforto e segurança dos passageiros (cinto de segurança, nível de atenção, estilo de pilotagem, reflexo). A evolução da automação levou à adição de sensores para auxiliar o motorista, como radares, câmeras, detecção de luz (Light Detection And Ranging - *LIDAR*), que atualmente suportam funções essenciais para estacionar e evitar colisões. Eles ainda podem ser usados para aplicações de direção semi-autônomas nas quais um veículo segue outro, e somente o primeiro veículo na fila deve ter uma aplicação de motorista totalmente autônoma na qual o motorista é dispensável. Um relatório recente da (MARKETSNDMARKETS, 2019) prevê que o tamanho do mercado global de carros conectados atinja US $212,7 bilhões até 2027.

Redes VANETs (AL-SULTAN et al., 2014), um tipo especial de Rede Ad Hoc Móvel (*Mobile Ad Hoc Network - MANET*) com características de mobilidade rápida e dinâmica, proporcionam troca de informações entre veículos através de um modelo de comunicação conhecido como V2V; com infraestruturas como (*Evolved Node B - eNB*) de (Long Term Support - *LTS*) e 802.11p RSU, no modelo V2I e com quaisquer outros elementos no caminho, como objetos IoT, pedestres, *smartphones*, em busca de um Sistema de Transporte Inteligente (*Intelligent Transportation System - ITS*). Redes heterogêneas sem fio baseadas em 802.11p, *Bluetooth*, 3G / UMTS, 4G / LTE-A e 5G, juntamente com protocolos de roteamento otimizados para mobilidade, têm sido componentes importantes para alcançar essa visão de ITS. Dessa forma, é possível utilizar dados de sensores internos e externos de veículos, além de sensores ambientais, como temperatura, tráfego, condições de pista, semáforos e algoritmos inteligentes de alimentação para tomar decisões de rotas, controle de tráfego global e prevenção de acidentes.

Grande parte das aplicações IoT e de redes VANETs, assim como os aplicativos dos

usuários, são baseados no modelo web de solicitação-processamento-resposta que depende muito da Internet e do acesso as infraestruturas em nuvem (LE et al., 2017). Tradicionalmente, os dados são enviados para servidores na Nuvem, processados e seus resultados acionam a atuação para construir um ITS. Assim, uma vez que expandimos as capacidades de comunicação dos veículos para chegar a Internet, também podemos estender os conceitos de VANETs, onde os veículos trocam dados entre si, a IoV, onde a nuvem apresenta poder de processamento quase ilimitado, suporta ITS, novos serviços e modelos de negócios com algoritmos aprimorados, telemática e até Aprendizado de Máquina (*Machine Learning - ML*)(YANG et al., 2017).

Neste contexto, a computação Fog surge como um novo conceito que complementa a computação em nuvem. De acordo com algumas definições (CONSORTIUM et al., 2017) (YI; LI; LI, 2015), a Fog distribui as funções de computação, armazenamento, controle e rede mais próximo dos usuários ao longo de uma cadeia fim-a-fim de processamento e armazenamento da nuvem aos objetos *Cloud-to-Thing Continuum*. A Fog aproveita esses recursos de, teoricamente, qualquer dispositivo capaz de se comunicar e operar de forma autônoma ou controlada por interfaces pré-definidas e possivelmente abertas, como *switches*, roteadores, dispositivos móveis, *smartphones*, *tablets* e veículos. A literatura aponta que a Fog atende melhor à aplicações que exigem latência próxima de zero, especialmente aquelas relacionadas a carros autônomos(AL-SULTAN et al., 2014).

Assim, a indústria automotiva pode se beneficiar tanto da enorme quantidade de dados coletados de sensores quanto dos avanços na computação em nuvem e das redes veiculares. No entanto, o processamento de dados em uma nuvem remota e centralizada para fins de infoentretenimento ou tomada de decisão é prejudicial aos requisitos de latência ultrabaixa de vários cenários e aplicações, como direção autônoma, segurança, realidade aumentada e jogos. Além disso, o modelo de processamento centralizado não é escalável e pode impactar na sobrecarga do servidor e no congestionamento da rede, devido às solicitações terem que cruzar toda a rede em direção à plataforma distante da nuvem. Se o setor automotivo adotar o modelo em que todas as aplicações devem passar pela infraestrutura de nuvem, o tráfego resultante poderá gerar sobrecarga ainda maior na rede e nos servidores.

Para atender aos requisitos de latência e da prevenção da sobrecarga, o IoV exige novas perspectivas para o desenvolvimento de algoritmos e técnicas que controlam o tráfego de dados de veículos e usuários com base na sinergia entre tecnologias de nuvem, rede e virtualização, como SDN, NFV, computação na borda (*Edge Computing*) e *containers*, além dos avanços em relação às redes sem fio. O paradigma SDN, bastante investigado na academia, e já adotado por grandes empresas, propõe a separação entre plano de dados e plano de controle e mantém uma visão global da rede. A arquitetura NFV, padronizada e mantida pelo Instituto Europeu de Normas de Telecomunicações (*European Telecommunications Standards Institute - ETSI*), permite a criação de redes dinâmicas em um *pool* de recursos virtuais, facilitando qualquer configuração necessária para a consolidação de

serviços. A computação de borda move os recursos de processamento e armazenamento da nuvem para a proximidade dos usuários finais. Por fim, os containers permitem a virtualização em nível de aplicação, facilitando a execução de serviços semelhantes em diferentes plataformas facilitando a migração e a replicação de aplicações.

A academia e a indústria vêm desenvolvendo algoritmos complexos que podem permitir que os usuários acessem aplicações mesmo sob alta mobilidade (CHAHAL et al., 2017). A maioria desses mecanismos procura reduzir a latência no acesso a serviços em nuvem, seja no modelo V2V ou no V2I. Estudos recentes têm apontado que a complexidade computacional para tais algoritmos está aumentando (OUYANG; ZHOU; CHEN, 2018). Tal análise pode dificultar o progresso nesta área e agora aponta para a necessidade de adotar novas abordagens cada vez mais inovadoras. Entre essas soluções, aquelas que possuem componentes distribuídos, seja para serviços ou para a própria configuração da rede, destacam-se que podem alavancar a descentralização natural das redes veiculares. Os algoritmos P2P auto-organizados e descentralizados surgem como uma tendência para complementar e otimizar o comportamento da rede (SATOH, 2013).

Neste trabalho, propomos o Edge-On, uma plataforma que, além de trazer serviços para a borda da rede, utiliza-se de veículos, que podem ter um alto poder de processamento (SOUA; TOHME, 2018) na execução de serviços *web* baseados em P2P por meio da integração entre SDN, NFV e Containers. A plataforma proposta oferece recursos tolerantes a falhas, pois demonstramos a possibilidade de que o serviço irá operar no modo P2P, independente da intervenção direta na nuvem. A plataforma proposta também elimina o problema de um ponto único de falha, porque os pares podem se autoconfigurar, instanciar serviços e trocar dados diretamente. Uma vez executados em containers e microsserviços, eles são leves e podem operar em dispositivos com recursos limitados.

Para o melhor de nosso conhecimento, este é o primeiro trabalho que avalia o equilíbrio entre a visão global da rede, e o controle intrínseco à SDN, com a distribuição de um algoritmo P2P sendo executado em containers em uma infraestrutura baseada em NFV no contexto de IoV. Identificamos o papel que cada elemento, centralizado e distribuído, pode desempenhar em uma arquitetura IoV e apresentamos alguns caminhos que outros pesquisadores podem seguir para obter serviços com latência próxima de zero.

1.2 OBJETIVO GERAL

O objetivo geral desta proposta de dissertação de mestrado é projetar, implementar e avaliar uma plataforma utilizando-se de ferramentas de *softwares* para orquestração e gerência de recursos em um sistema de computação em nuvem/borda/névoa (*Cloud/Edge/Fog computing*) multicamada, distribuído, dinâmico e heterogêneo, a fim de trazer computação para a borda da rede.

1.3 OBJETIVOS ESPECÍFICOS

- Criar uma plataforma hierárquica *Cloud/Edge/Fog Computing* para a execução de aplicações em redes *Vehicular Ad Hoc Networks* (VANETs).
- Elaborar um arcabouço MEC-NFV a partir da plataforma *OpenStack* e do serviço NFV *tacker*.
- Conceber uma adição a arquitetura de (*On Board Unit - OBU*) que dará suporte à SDN e a execução de aplicações em containers.
- Elaborar uma estratégia de orquestração das funções virtualizadas de rede (*Virtual Network Functions - VNFs*) e das aplicações em execução em containers.
- Criar uma estratégia de replicação e balanceamento de uma aplicação entre nuvem e MEC.
- Criar uma estratégia de compartilhamento P2P de conteúdo entre os próprios veículos.
- Elaborar uma alternativa para a aplicação utilizar o mínimo de recursos computacionais possível de cada veículo.

1.4 ESTRUTURA DA DISSERTAÇÃO

Esta dissertação contém 6 Capítulos organizados da seguinte forma:

O Capítulo 2 apresenta os conceitos e definições básicos sobre Redes Veiculares, Redes Definidas por Software, Computação em Nuvem, Computação na Borda da Rede, Funções de Redes Virtuais e Tecnologia de Container.

O Capítulo 3 apresenta e discute os trabalhos relacionados que inspiraram o tema proposto por esta dissertação.

O Capítulo 4 descreve a proposta da plataforma Edge-On, os elementos fundamentais que fazem parte da plataforma e detalhes de cada tecnologia para o bom funcionamento da proposta.

O Capítulo 5 descreve as avaliações realizadas, detalhando o ambiente de testes e cenários definidos em cada experimento e suas respectivas métricas. Além dos resultados obtidos com a utilização da plataforma proposta.

Posteriormente, o Capítulo 6 conclui o trabalho apresentando um resumo dos resultados alcançados, contribuições e mostrando direções para trabalhos futuros.

2 FUNDAMENTAÇÃO TEÓRICA

Neste capítulo, os conceitos das tecnologias que deram base ao desenvolvimento da proposta desta dissertação são apresentados. Este está dividido em duas subseções. A Seção 2.1 apresenta conceitos referentes as tecnologias VANET, SDN, NFV, *container*, Computação em Nuvem e Computação na Borda. A Seção 2.2 apresentada os elementos que formam o ambiente bem como o seu funcionamento.

2.1 VANETS

Os avanços recentes em tecnologia têm possibilitado um aumento no número de veículos equipados com sensores, (*Global Positioning System - GPS*) e dispositivos *Wi-fi*. Eles capacitam os veículos, comumente chamados de nós, há realizarem troca de informações entre si, também conhecida como V2V, permitindo assim o surgimento das redes veiculares *ad hoc* VANETs (ZHANG, 2011). Elas representam um sistema de comunicação veicular que tem potencial para melhorar a segurança no trânsito por meio do compartilhamento de informações para a prevenção de acidentes, engarrafamentos, além de auxiliar também na investigação pós-acidente (AL-SULTAN et al., 2014). Um dos objetivos principais por trás da distribuição e compartilhamento destas informações é fornecer mensagens de segurança para alertar os motoristas sobre possíveis perigos nas vias, buscando assim diminuir o número de acidentes, salvar vidas proporcionando resgates mais rápidos por parte dos transportes de saúde e proporcionar aos passageiros viagens mais agradáveis tanto por segurança quanto por entretenimento a bordo do veículo.

As redes veiculares são uma parte fundamental da estrutura de ITS, pois viabilizam a troca de informações entre veículos automotores com infraestrutura fixa localizada às margens de ruas ou de rodovias. As redes veiculares tem suas particularidades em relação a outras redes sem fio devido serem distribuídas, auto-organizadas e potencialmente de alta mobilidade, composta de veículos que se comunicam por um meio sem fio. O deslocamento de cada nó representado por um veículo, tais como caminhões e ônibus com interfaces de comunicação sem fio, é restringido pela direção da estrada, abrangendo o tráfego e os regulamentos de trânsito da via. Essas restrições introduzem características específicas às VANETs como a alta dinâmica na topologia da rede e a conectividade intermitente (BASAGNI et al., 2013).

O ITS é uma arquitetura que traz consigo uma variedade de serviços ao qual requerem conexões flexíveis, escaláveis e eficientes entre os veículos e as infraestruturas nas margens das ruas. Neste sistema, protocolos baseados em infraestrutura, como protocolos celular e protocolos de VANETs estão sendo integrados para facilitar o processo de coleta de informações. Além disso, determinados serviços ITS podem ter requisitos para aplicações

que necessitam de qualidade de serviço (BOUKERCHE; ROBSON, 2018).

A Figura 1 apresenta uma arquitetura geral para as VANETs e ilustra possíveis cenários de comunicação. Nesse cenário o veículo pode tanto receber quanto transmitir informações. As estratégias de comunicação V2V e V2I estão apresentadas na figura, demonstrando o uso do (*Wireless Access in Vehicular Environments - WAVE*). As estratégias comuns de comunicação nas redes veiculares estão listadas da seguinte forma:

- **Veículo-para-Veículo.** O V2V consiste em comunicações entre veículos no modo *ad hoc* (KAUSHIK, 2013). A comunicação é possível através de transmissões *multi-hop* de veículos que estão no alcance, e suporta a transmissão de mensagens importantes relacionadas ao tráfego, como o estado da estrada, condições de condução e acidentes de trânsito.

- **Veículo-para-Infraestrutura.** O V2I representa as comunicações entre veículos e infraestrutura de rede fixa para a troca de informações (KAUSHIK, 2013). Essas comunicações, na maioria dos casos, envolvem o acesso de redes externas, como a *Internet* ou qualquer serviço tradicional de nuvem, por meio de RSUs e *gateways*. Os *links* de comunicação fornecidos na V2I apresentam melhor segurança, mas a largura de banda que eles exigem é maior do que para os *links* V2V.

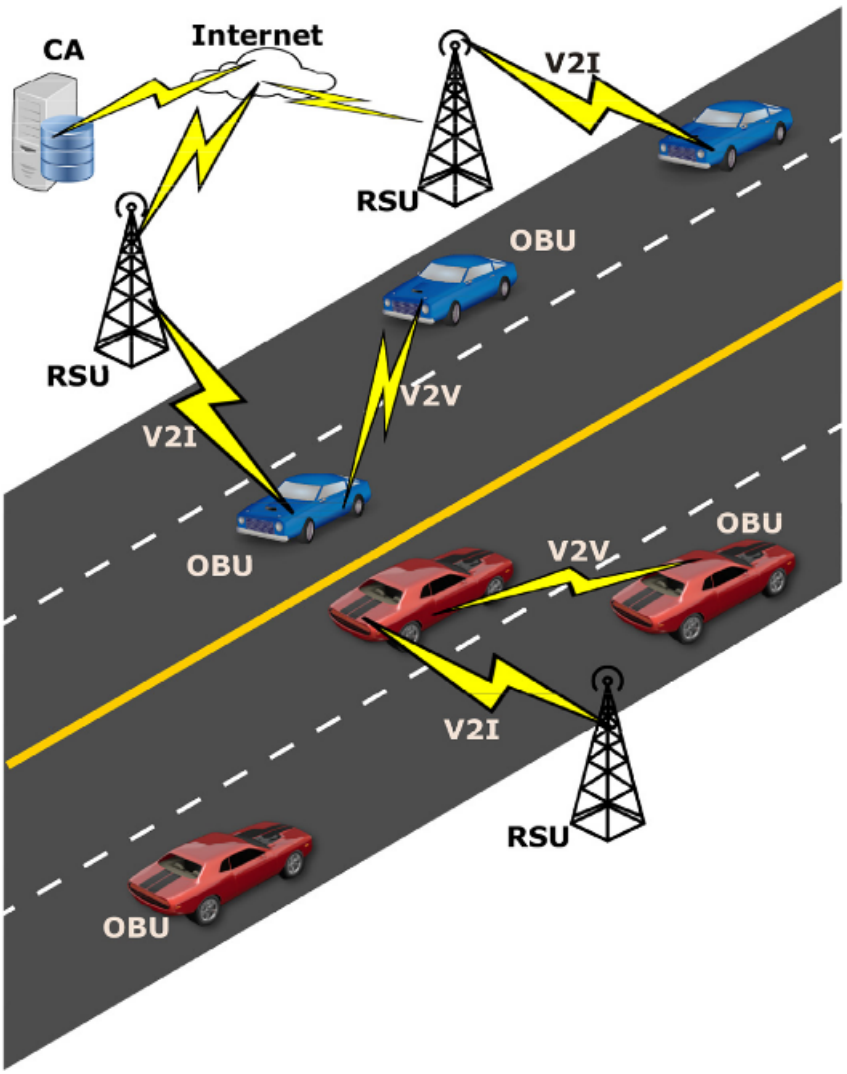
- **Veículo-para-Tudo.** O V2X compreende a transferência de dados entre um veículo e qualquer outro elemento com recursos de rede, incluindo qualquer elemento envolvido nesta comunicação (MÜLLER; ROERMUND; STEIGEMANN, 2016). Esse método de comunicação veicular é geral e inclui outras abordagens específicas, como V2I, V2V, (*Vehicle-to-Pedestrian - V2P*), (*Vehicle-to-Device - V2D*) e (*Vehicle-to-Grid - V2G*).

Para (BASAGNI et al., 2013), as formas de comunicação V2V e V2I possuem diferentes funcionalidades e vantagens. As comunicações V2V oferecem vantagens em dar suporte a aplicações de segurança em tempo crítico, como evitar acidentes de colisões entre veículos, informações sobre perigo ou congestionamento nas vias, evitando assim custos de infraestrutura e taxas para os usuários. A vantagem do V2I em relação ao V2V é o acesso a aplicativos na Internet (entretenimento), coordenação global dos dados e previsão, segurança, garantia de (*Quality of Service - QoS*) e cobertura mais ampla das vias.

Um dos meios de comunicação sem fio que possibilita que os veículos e as RSUs se comuniquem é chamado de WAVE. Os principais componentes que constituem o ambiente de Redes Veiculares, considerados partes essenciais de sua arquitetura são: OBU, RSU e (*Certification Authorities - CA*). Cada componente tem sua propriedade própria em relação aos demais componentes.

A OBU está presente em todos os veículos, é responsável pela comunicação V2V, em que o veículo utiliza sua OBU para se comunicar com outro veículo, e pela comunicação

Figura 1 – Cenário geral para Redes Veículares.



Fonte: (BOUKERCHE; ROBSON, 2018)

V2I em que um veículo utiliza sua OBU para se comunicar com uma RSU. Ela é responsável por coletar os dados fornecidos para depois poderem ser usados por aplicativos e serviços (AL-SULTAN et al., 2014). Ela é integrada a um conjunto de sensores, onde pode agregar informações sobre a posição, velocidade e aceleração/desaceleração de um veículo e é responsável por transferir esses dados a outros dispositivos da rede, dispositivos esses que podem ser veículos ou RSUs.

RSU é um componente de comunicação essencial que permite a comunicação com a infraestrutura de rede. É um dispositivo físico de localização fixa posicionado ao lado de estradas e rodovias, ou em localizações específicas como postos de gasolina, estacionamentos e restaurantes e é responsável por coletar e disseminar dados que podem estar relacionados ao *status* do tráfego, como acidentes futuros, pontos de estacionamento mais próximos, postos de gasolina e comprimento de uma fila de tráfego. Para ser capaz de executar suas funções, uma RSU apresenta pelo menos uma interface de rede que a conecta

à *Internet* e uma interface de comunicação sem fio de curto alcance (BABY et al., 2013).

A CA é responsável por evitar problemas de segurança em redes veiculares (MEJRI; BEN-OTHMAN; HAMDI, 2014). Esse componente está ciente de seus vizinhos, assim podendo fornecer validação e avaliação da entidade em fontes de informação ao receber mensagens. Uma CA é uma parte totalmente confiável no ambiente e é crucial para redes veiculares.

Atualmente existem inúmeras tecnologias de acesso sem fio disponíveis, tais como: 3G/4G, WLAN/Wifi, WiMAX, DSRC/WAVE ou IEEE 802.11p que podem ser usadas para fornecer a interface de comunicação exigida pelos veículos, a fim de se comunicar uns com os outros, comunicação V2V ou para se comunicar com as RSUs, comunicação V2I. Essas tecnologias de comunicação visam melhorar a segurança no trânsito, a eficiência do tráfego, entretenimento devido às longas horas no trânsito proporcionando conforto ao motorista e ao passageiro, permitindo assim um conjunto de aplicações de segurança e não relacionadas à segurança. Algumas dessas tecnologias dependem de uma infraestrutura centralizada para coordenar as comunicações entre os nós. Em contrapartida, outras tecnologias operam no modo *ad hoc* (comunicação distribuída) (AL-SULTAN et al., 2014).

2.1.1 Aplicações VANETs

De acordo com (AL-SULTAN et al., 2014), as comunicações V2V e V2I possibilitam o desenvolvimento de um grande número de aplicações que podem prover uma variedade de informações aos motoristas e viajantes. O número elevado de dispositivos integrados a bordo do veículo com a interface de rede, diferentes tipos de sensores e receptores (*Global Positioning System - GPS*), concede aos veículos a habilidade de coletar, processar e disseminar informações sobre ele mesmo e sobre outros veículos em sua proximidade. O grande interesse coletivo e esforços para explorar as redes veiculares em ambientes de tráfego a fim de aumentar a segurança nas estradas e prover conforto aos passageiros geraram inúmeras aplicações para um vasto conjunto de propósitos. Tais aplicativos se encaixam em três categorias principais: segurança rodoviária, eficiência de tráfego e conforto e entretenimento do passageiro. (BOUKERCHE; ROBSON, 2018).

- **Aplicações para segurança no trânsito:** O objetivo dessas aplicações é utilizar a comunicação sem fio entre os veículos ou entre os veículos e a infraestrutura, onde essas aplicações se concentram em fazer uso dos recursos dessa comunicação que a Rede Veicular introduz como um facilitador para fornecer mensagens críticas. Essas mensagens podem oferecer suporte a notificações de risco, prevenção de colisões e alertas, que podem fornecer informações sobre condições climáticas adversas em prol da segurança nas estradas e da redução de acidentes de trânsito.

- **Aplicações para eficiência no tráfego:** Essas aplicações visam auxiliar os motoristas e o sistema de gerenciamento de transporte. Elas fornecem meios para melhor

trabalhar com as vias, auxiliando na tomada de decisões durante a condução e complementando os processos de decisão com informações adicionais. Por exemplo, os aplicativos podem ajudar a alertar sobre a possibilidade de engarrafamentos, a detecção de veículos em filas ou a ultrapassagem de outros veículos.

- **Aplicações para o conforto e entretenimento dos passageiros:** Esta categoria envolve aplicações que não são relacionadas à segurança. Essas aplicações priorizam o conforto do motorista e dos passageiros. Elas são projetados para tornar as viagens em veículos mais agradáveis ao fornecer serviços a bordo, como acesso à *Internet*, mensagens e entretenimento. Os passageiros podem jogar jogos *online*, baixar vídeos enquanto o veículo está conectado à uma rede de infraestrutura.

2.2 CLOUD COMPUTING

Uma nuvem refere-se a um ambiente de TI externo ou interno que é concebido com a finalidade do cliente remotamente poder fazer o provisionamento de recursos computacionais escaláveis. Segundo a definição do (*National Institute of Standards and Technology - NIST*) (MELL; GRANCE et al., 2011) computação em nuvem é definida como um modelo para a habilitação, sob demanda de acesso à rede ubíqua, conveniente para um *pool* compartilhado de recursos de computação configuráveis (por exemplo, redes, servidores, armazenamento, aplicações e serviços) que possam ser rapidamente provisionados e liberados com o mínimo de esforço de gerenciamento ou interação com o provedor de serviços. Com características de auto serviço por demanda, acesso à rede ampla, *pooling* de recursos, elasticidade e escalabilidade.

Este modelo de nuvem é composto por cinco características essenciais: autoatendimento sob demanda (*On-demand self-service*), acesso amplo à rede (*Broad network access*), agrupamento de recurso (*resource pooling*), Elasticidade rápida (*Rapid elasticity*), Serviço mensurado (*Measured service*). Ela também é composta por três modelos de serviços: Software como Serviço (*Software as a Service - SaaS*), Plataforma como Serviço (Platform as a Service - *PaaS*) e Infraestrutura como Serviço (*Infrastructure as a Service - IaaS*) e é composto de quatro modelos de implantação: Nuvem Privada (*Private Cloud*), Nuvem Comunitária (*Community Cloud*), Nuvem Pública (*Public Cloud*) e Nuvem Híbrida (*Hybrid Cloud*).

2.2.1 Características Essenciais

As características podem ser entendidas como o diferencial promovido pela nuvem computacional. De acordo com (MELL; GRANCE et al., 2011), um sistema pode ser considerado uma nuvem computacional se apresentar as seguintes características:

- **Autoatendimento sob demanda (*On-demand self-service*):** O consumidor pode provisionar por conta própria recursos de computação, como tempo de execução de um servidor, armazenamento e rede, automaticamente e conforme necessário, sem necessitar intervenção humana dos provedores de serviços.
- **Acesso amplo à rede (*Broad network access*):** Os recursos estão disponíveis através da rede e são acessados através de mecanismos padronizados que promovem o uso por dispositivos clientes leves de diversas plataformas, tais como: (*smartphones*, *tablets*, *laptops* ou *desktops*).
- **Agrupamento de recurso (*Resource pooling*):** Os recursos de computação do provedor são agrupados para atender a múltiplos consumidores em modalidade multi-inquilinos, com recursos físicos e virtuais diferentes dinamicamente atribuídos conforme a demanda dos consumidores. Há uma certa independência de localização geográfica, uma vez que o consumidor em geral não controla ou conhece a localização exata dos recursos fornecidos (como armazenamento, processamento, memória e rede), mas pode ser capaz de especificar a localização em um nível de abstração mais alto (como país, estado ou *datacenter*).
- **Elasticidade rápida (*Rapid elasticity*):** Os recursos podem ser provisionados e liberados elasticamente, em alguns casos automaticamente, para rapidamente aumentar ou diminuir de acordo com a demanda. Para o consumidor, os recursos disponíveis para provisionamento muitas vezes parecem ser ilimitados e podem ser alocados em qualquer quantidade e a qualquer momento.
- **Serviço mensurado (*Measured service*):** Os sistemas na nuvem automaticamente controlam e otimizam o uso dos recursos através de medições em um nível de abstração apropriado para o tipo de serviço (como armazenamento, processamento, comunicação de rede e contas de usuário ativas). A utilização de recursos pode ser monitorada, controlada e informada, gerando transparência tanto para o fornecedor como para o consumidor do serviço utilizado.

2.2.2 Modelos de Serviços

Conforme a definição de (MELL; GRANCE et al., 2011), há três opções de modelos de entrega de serviços de nuvem:

- **Software como Serviço (SaaS):** É um modelo onde a capacidade fornecida ao consumidor e a utilização de serviços do provedor são executados em uma infraestrutura da nuvem. As aplicações são acessíveis a partir de vários dispositivos clientes, tais como um navegador Web. Todo o controle e gerenciamento da rede, sistemas operacionais, armazenamento e servidores, é feito pelo provedor do serviço. Exemplos de SaaS: Gmail, Google Docs, Google Drive, Google Calendar e etc.

- **Plataforma como Serviço (PaaS):** É fornecida ao consumidor a capacidade para implantar sobre a infraestrutura de nuvem aplicações criadas ou adquiridas usando linguagens de programação, bibliotecas, serviços e ferramentas suportadas pelo provedor. O consumidor tem controle sobre os aplicativos implementados e possivelmente definições de configuração para o ambiente de hospedagem dos aplicativos. Se destacam como exemplos de PaaS: App Engine do Google e o Windows Azure da Microsoft.

- **Infraestrutura como Serviço (Iaas):** O consumidor recebe a capacidade de provisionar processamento, armazenamento, redes e outros recursos de computação fundamentais, onde ele será capaz de implantar e executar diversos *softwares*, que podem incluir sistemas operacionais e aplicativos. Neste cenário, o usuário não tem o controle da infraestrutura física, mas, com os métodos providos pela virtualização, possui controle sobre os recursos virtualizados de armazenamento, rede e processamento. Exemplo de IaaS é a AWS, Azure e o serviço oferecido pelo OpenStack.

2.2.3 Modelos de Implantação

De acordo com (MELL; GRANCE et al., 2011), um ambiente de computação em nuvem pode ser implantado nas seguintes formas:

- **Nuvem Privada (*Private cloud*):** No modelo de implantação privada, a infraestrutura de nuvem é provisionada para uso exclusivo de uma única organização que agrupa vários consumidores (por exemplo, unidades de negócio). Pode gerida e operada pela organização, um terceiro, ou alguma combinação deles, e ela pode existir localmente ou remotamente.

- **Nuvem Comunitária (*Community cloud*):** No modelo de implantação comunitária, a infraestrutura é provisionada para o uso exclusivo de uma determinada comunidade de consumidores onde pode ocorrer o compartilhamento de uma infraestrutura de nuvem por diversas empresas, sendo esta suportada por uma comunidade específica que partilhou seus interesses, tais como a missão, os requisitos de segurança, política e considerações sobre flexibilidade. Este tipo de modelo de implantação pode existir localmente ou remotamente e pode ser administrado por alguma empresa da comunidade ou por terceiros.

- **Nuvem Pública (*Public cloud*):** No modelo de implantação público, a infraestrutura de nuvem é disponibilizada para o público em geral, sendo acessado por qualquer usuário que conheça a localização do serviço. Pode ser detida, gerida e operada pelo o meio acadêmico ou organização governamental, ou alguma combinação deles. As restrições de acesso, bem como políticas de autenticação e autorização neste modelo são menos rigorosas. Um dos benefícios da nuvem pública é que elas

tendem a ser muito maiores que as nuvens privadas e, portanto, mantém uma maior quantidade de recursos. Isso permite uma maior escalabilidade de recursos, e por serem mais utilizadas à resolução de possíveis *bugs* são resolvidas mais rapidamente.

- **Nuvem Híbrida (*Hybrid cloud*):** No modelo de implantação híbrida, a infraestrutura de nuvem é uma composição de duas ou mais infraestruturas de nuvem (privada, comunitária ou pública) que permanecem entidades únicas, mas são unidas por tecnologia padronizada ou proprietária que permite a portabilidade de dados e aplicações.

2.2.4 MEC - Multi-access Edge Computing

A *Edge Computing* refere-se a um amplo conjunto de técnicas projetadas para mover a computação armazenamento e rede para fora da infraestrutura de nuvem remota, seja ela pública ou privada, para mais perto da fonte de dados. Para a classe emergente de "Aplicações 5G" isso é uma questão de necessidade frequente. Localizar esses aplicativos em uma nuvem tradicional não permite atender a certos requisitos rigorosos, como a latência de requisição e resposta. Em outros casos, como IoT e V2X, espera-se que a quantidade de dados aumente substancialmente (SABELLA et al., 2019). A computação de borda pode atenuar isso, coletando e processando dados mais próximos do usuário. O ambiente da MEC é caracterizado por baixa latência, proximidade dos dispositivos finais, alta largura de banda e percepção em tempo real das informações da rede de rádio e reconhecimento de local (HU et al., 2015).

O MEC se concentra em habilitar a computação na borda da rede de acesso (móvel ou não), trazendo o poder de processamento para o mais próximo possível do cliente final sem estar localizado no dispositivo do usuário. MEC permite a implantação bem sucedida de novos casos de uso e vários serviços que podem ser personalizados de acordo com as exigências e demandas do cliente. Uma abordagem arquitetural baseada em microsserviços é particularmente adequada para esse novo padrão devido sua portabilidade. Algumas das principais aplicações e casos de uso são (SABELLA et al., 2019):

- Otimização de entrega de conteúdo de vídeo;
- Análise de fluxo de vídeo e vigilância por vídeo;
- Realidade Aumentada e Realidade Virtual;
- Ativação de aplicativos corporativos;
- Aplicações com necessidades críticas de comunicação, como segurança, controle de tráfego e de saúde;
- Carros conectados;

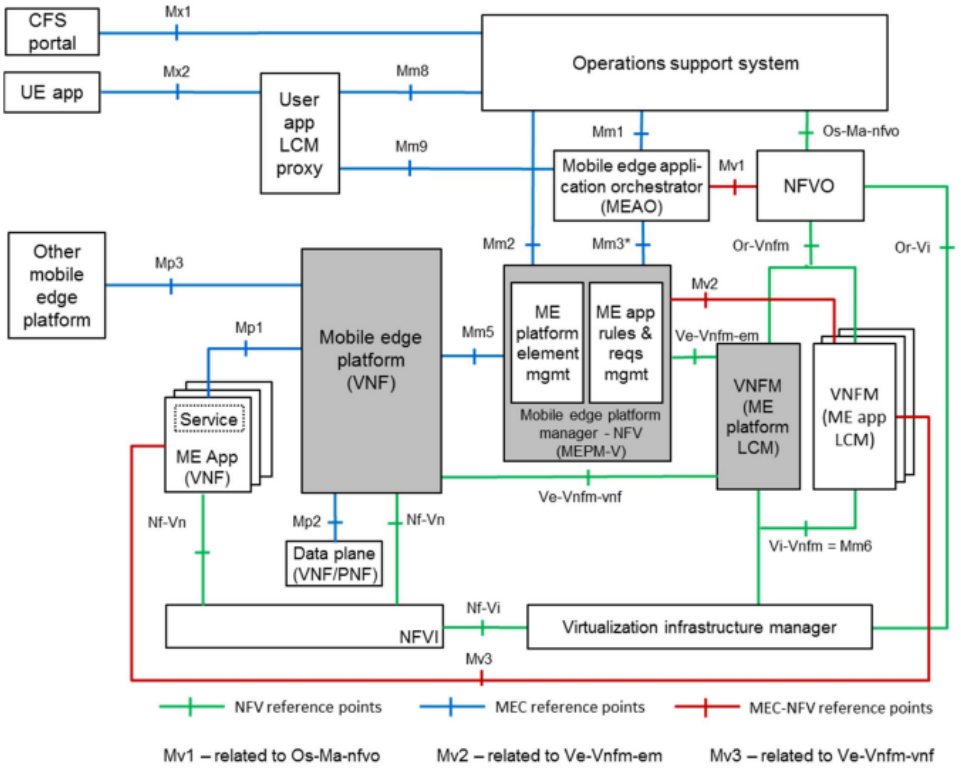
- Aplicativos IoT e *Gateway*;
- Serviços cientes de localização e contexto; e
- Aplicações de *Smart City*

O MEC é baseado em uma plataforma virtualizada, com uma abordagem complementar ao NFV. Onde o NFV está focado em funções de rede, e o *framework* MEC em permitir que os aplicativos sejam executados na borda da rede. A infraestrutura que hospeda MEC e NFV são bastante semelhantes; assim, para permitir que os operadores se beneficiem ao máximo possível de seu investimento, existe o benefício de reutilizar a infraestrutura e o gerenciamento de infraestrutura do NFV na maior medida possível, hospedando os dois serviços em conjunto: VNFs e os aplicativos MEC na mesma infraestrutura(HU et al., 2015). Assim o (*Mobile Network Operator - MNO*) procurará soluções eficientes, abertas e orientadas por uma tecnologia padronizada , alavancando a automação, reutilizando ativos e recursos já existentes o máximo possível, como por exemplo, um *host* que hospeda NFV distribuído provavelmente será usado também para hospedar aplicações de borda. Acredita-se que a orquestração de borda será um quesito obrigatório para permitir as funcionalidades de mobilidade em tempo real exigidas pelos casos de uso vertical automotivo e V2X (ETSI-GR-MEC-017, 2018). Para tal, uma solução de orquestração de três camadas precisa ser totalmente adaptada à implantação do MEC (GIUST et al., 2018):

- **Infraestrutura:** O gerenciamento dos nós de determinadas infraestruturas poderá diferir em capacidades de tamanho, processamento, armazenamento e latência; a gestão da camada de virtualização dessas infraestruturas poderá fazer o uso de orquestradores de nuvem privada como Openstack, geograficamente distribuídos através de múltiplos *clusters* para atender regiões específicas.
- **Plataforma MEC:** É a estrutura no topo de cada nó de borda responsável por habilitar as aplicações MEC. Ela deve ser segura, gerenciada automaticamente e integrada a qualquer ambiente NFV existente. Esta plataforma pode fornecer serviços adicionais que as aplicações da MEC poderão consumir.
- **Aplicações MEC:** Elas exigem uma orquestração aninhada porque precisam consumir serviços, serem encadeados uns aos outros, oferecer suporte à mobilidade do usuário e serem integradas a infraestruturas de nuvens externas. Uma estrutura flexível de alocação de recursos dentro de um *pool* limitado ou não de recursos em um local na borda deve ser implementada, levando em conta o problema de localização de aplicações MEC sobre vários nós de borda, com base nos requisitos de latência e *QoS*.

Pode-se visualizar na Figura 2 a utilização e a sinergia entre os dois padrões do ETSI, onde a arquitetura MEC utiliza-se de um ambiente NFV alavancando a automação dos

Figura 2 – Referência da arquitetura MEC em um ambiente NFV



Fonte: (ETSI-GR-MEC-017, 2018)

serviços, e o orquestrador NFV fica responsável por gerenciar o ciclo de vida das instâncias. A plataforma proposta Edge-On buscou seguir as melhores práticas e padrões estabelecidos pelo ETSI buscando a sinergia e utilizando o melhor do que cada tecnologia tem a oferecer.

O orquestrador MEC (*MEC Orchestrator - MEO*) foi substituído por um orquestrador de aplicações MEC (*MEC Application Orchestrator - MEAO*) que se baseia no orquestrador NFV (*NFV-Orchestrator - NFVO*) para orquestração de recursos e para orquestração do conjunto de VNFs, e de aplicações MEC com um ou mais serviços de rede NFV. A principal funcionalidade do orquestrador de borda será o gerenciamento à nível de aplicação da arquitetura MEC. Esse orquestrador de borda será responsável pelas seguintes funções (ETSI-GS-MEC-003, 2019):

- Manter uma visão geral do sistema com base na implantação de *hosts* MEC, recursos disponíveis, disponibilidade de serviços bem como sua topologia;
- Integração de dependências de aplicações, incluindo a verificação da integridade e autenticidade de pacotes, validar as regras e os requisitos da aplicação e, se necessário, ajustá-los para cumprir as políticas do operador, manter um registro de pacotes integrados e preparar os gerenciadores de infraestrutura de virtualização para lidar com as aplicações;

- Selecionar *host(s)* MEC apropriado(s) para instanciação de aplicação com base em restrições, como latência, disponibilidade de recursos e serviços;
- Ficar responsável pelo ciclo de vida de instâncias e aplicações.
- Desencadear a realocação de aplicações conforme necessário, quando suportado.

Um orquestrador da infraestrutura MEC foi adequadamente definido pelas especificações do ETSI-MEC para abordar totalmente as duas últimas camadas (intermediária e superior), ou seja, ciclo de vida das instâncias e das aplicações, mas seu papel e sua responsabilidade precisam ser mais investigadas quando implantados em uma arquitetura NFV ao qual contém seu próprio orquestrador. Especificamente, a implantação do MEC em um ambiente NFV, descrito no (ETSI-GR-MEC-017, 2018), que analisa esse cenário com o objetivo de reutilizar o máximo possível as capacidades do sistema NFV-MANO. Dessa forma, os administradores de rede podem gerenciar conjuntamente funções de redes e suas aplicações, otimizando assim a criação e a orquestração de serviços de rede. Além disso, isso permite reutilizar o investimento feito para a implementação de um ambiente NFV executando aplicações no topo da MEC. O MEC por padrão tem seu próprio orquestrador, mas quando executado em uma arquitetura NFV, o administrador poderá utilizar o orquestrador da NFV para gerência do ambiente. Pois o papel e responsabilidade do orquestrador MEC precisa ser mais investigados quando implantados em uma arquitetura NFV (GIUST et al., 2018).

Alguns recursos e configurações são totalmente definidos por padrão e normas, enquanto outros são deixados para a implementação específica do provedor MEC. Para que todos os padrões sejam utilizados corretamente são fornecidas algumas considerações preliminares de alto nível para o desenvolvimento e a implantação de uma aplicação na infraestrutura MEC (SABELLA et al., 2019):

- **Solução baseada em DNS:** A aplicação deve ser projetada para oferecer suporte à uma solução baseada em DNS para redirecionamento de tráfego. A aplicação deve enviar uma consulta DNS para um nome de domínio registrado.
- **Nome do domínio:** Registrar um nome para o serviço que deverá ser conhecido pela aplicação cliente para obter acesso.
- **Nuvem como *backend*** Se houver um requisito para o serviço estar disponível, independentemente de existir ou não um sistema local do MEC, será necessário um serviço de *backend* como uma solução reserva hospedada na nuvem ou em outras infraestruturas.
- **Dados de usuário sensíveis ao contexto:** Os dados do contexto do usuário podem ser confidenciais do ponto de vista legal e não podem ser transferidos de uma

jurisdição para outra. Essa transferência de contexto de uso é algo que o desenvolvedor deve se preocupar quando for manipular sua aplicação, caso a mobilidade for suportada pela aplicação.

- **Empacotamento da aplicação:** Um aplicação MEC é executada como uma aplicação virtualizada, em uma (*Virtual Machine - VM*) ou uma aplicação em *container*, além da infraestrutura de virtualização fornecida pelo *host* MEC. O empacotamento apropriado da aplicação está diretamente ligado ao ambiente de tempo de execução do sistema, e portanto, o desenvolvedor precisa adaptar-se adequadamente, fornecendo um empacotamento, tais como: VMs, *containers* do Docker ou Kubernetes.

- **Forneça metadados com requisitos da aplicação:** A infraestrutura deve fornecer para a aplicação seus requisitos mínimos, tais como tolerância de latência, recursos de rede, recursos de armazenamento e CPU.

2.2.5 Fog Computing

Com uma maior visibilidade e um crescente estudo sobre redes veiculares, um relatório da Cisco estima que haverá mais de 300 milhões de carros conectados até 2022, gerando mais de 400 *PetaBytes* de dados sobre os mais diversos tipos de redes móveis(MAI; SCHLESINGER, 2011). Tendo como base esse número elevado de dados que serão trafegados, estudos atuais na área de IoT, discutem que servidores em nuvem que por muitas vezes são os responsáveis por fornecerem serviços computacionais a essas aplicações IoT, podem prejudicá-las, uma vez que muitas delas são sensíveis a atrasos de comunicação (DASTJERDI et al., 2016)(BONOMI et al., 2012). Esses mesmos problemas poderão afetar as redes VANETs.

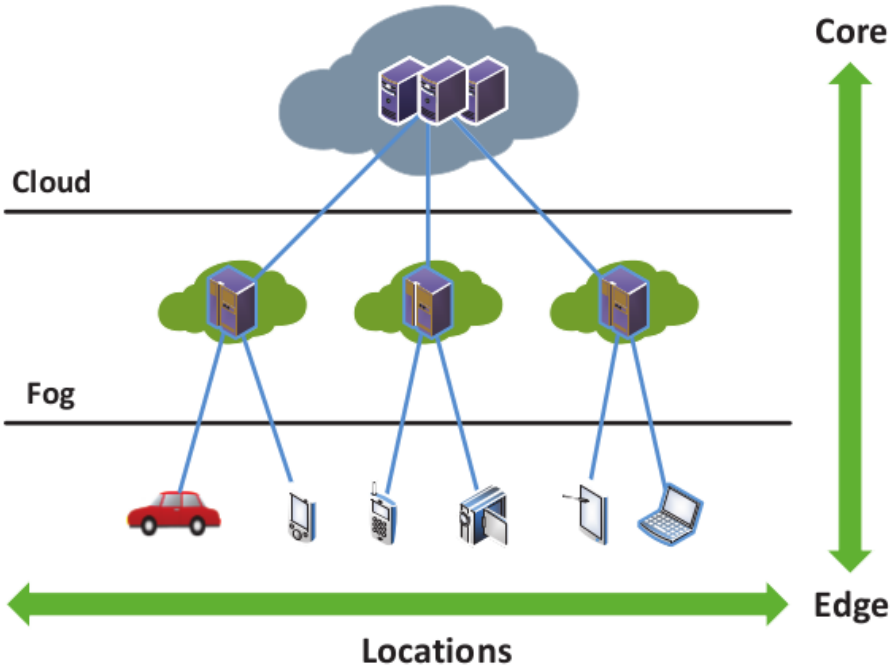
Tentando contornar esse problema, o conceito de fog computacional foi proposto em 2012 pela CISCO como uma extensão do paradigma de computação em nuvem, como uma forma de aproximar elementos de computação na borda da rede, utilizando a capacidade computacional dos diversos dispositivos que se encontram espalhados diminuindo a latência e os atrasos de comunicação em relação a serviços em nuvem distantes e muitas vezes sobrecarregados (MAHMUD; KOTAGIRI; BUYYA, 2018). Fog computacional não é um paradigma que busca substituir a computação em nuvem, e sim servir como suporte e extensão. Assim, a fog computacional fornece uma plataforma virtualizada que disponibiliza recursos computacionais, como capacidade de processamento e armazenamento entre os dispositivos espalhados na borda da rede e os servidores na nuvem (BONOMI et al., 2012). Essa estratégia possibilita aumentar a *QoS*, e diminuir a latência das aplicações que estão sendo executadas nos dispositivos finais uma vez que serviços distantes e centralizados são evitados (DASTJERDI et al., 2016)(STOJMENOVIC; WEN, 2014).

Para (YI; LI; LI, 2015), fog computacional surge como uma infraestrutura de computação na qual aplicações e serviços podem ser executados tanto em uma infraestrutura

de nuvem, como em dispositivos da própria rede provendo recursos computacionais, comunicação e integração de dados de forma a suportar aplicações sensíveis à latência e que estão distribuídas geograficamente. (VAQUERO; RODERO-MERINO, 2014) definem fog como um ambiente em que uma grande quantidade de dispositivos heterogêneos e descentralizados se comunicam e cooperam para executar tarefas sem a intervenção de entidades superiores participantes da arquitetura computacional. Na visão de (DASTJERDI; BUYYA, 2016), a fog computacional é um paradigma distribuído que fornece serviços similares aos fornecidos pela nuvem, contudo, o mesmo está em execução na borda da rede de forma distribuída.

Na Figura 3 é possível ver uma arquitetura de referência (STOJMENOVIC; WEN, 2014) que representa a organização de uma fog computacional. A primeira camada, no topo da figura, é composta pelos serviços de nuvem computacional. A segunda camada é composta pelos equipamentos de rede, responsáveis por conectar os usuários ao serviço de nuvem. Por fim, a terceira camada é composta pelos dispositivos finais, que são usuários como celulares, veículos inteligentes, sensores e atuadores, os quais geram muitos dados que podem ser processados ainda na borda da rede ou que utilizam os elementos da segunda camada para se conectarem uns aos outros e aos serviços de nuvem.

Figura 3 – Arquitetura alto nível de uma fog computacional, edge e nuvem



Fonte: (STOJMENOVIC; WEN, 2014)

Os nós da fog podem ser implantados em qualquer um dos elementos da segunda e terceira camada da arquitetura da Figura 3. Em algumas propostas de uso do paradigma de fog computacional pode-se encontrar uma tendência ao uso de dispositivos de rede

como nós de fog. É o caso das propostas apresentadas em (BITTENCOURT et al., 2017) e (VARGHESE et al., 2017), que utilizam cloudlets, roteadores, *gateways*, e pontos de acesso como nós de fog.

Nesse mesmo sentido, outros trabalhos como (LE et al., 2017), (NING; HUANG; WANG, 2019), (HE et al., 2017) e (NIKOLOUDAKIS et al., 2016), propõem a utilização dos próprios dispositivos finais como nós de fog. Pois em fog computacional, grande parte do processamento pode ocorrer nos próprios dispositivos que geram os dados, tendo em vista que nos últimos anos o seu poder de processamento e armazenamento teve significativa evolução. A arquitetura fog cria uma plataforma virtual que fornece serviços de processamento e de armazenamento entre o ambiente de nuvem e os dispositivos finais.

Dessa forma, fog computacional possibilita que esta integração com a nuvem implique em algumas vantagens (DASTJERDI et al., 2016):

- **Redução no tráfego de rede:** Conforme o número de dispositivos geradores de dados aumenta na rede, consequentemente a quantidade de tráfego gerado também aumenta, e fog computacional possibilita reduzi-los ao evitar que os processos busquem recursos em uma infraestrutura de nuvem que estará distante;
- **Mobilidade:** Fog computacional deve suportar a mobilidade dos elementos que estão na borda da rede;
- **Escalabilidade:** Por estender os serviços que estão no núcleo da rede, fog computacional possui um elevado grau de virtualização e escalabilidade, se adaptando ao grande número de dispositivos e à massiva quantidade de dados gerados por eles;
- **Baixa latência:** A proximidade dos recursos computacionais na borda da rede implica em uma baixa latência nas proximidades;
- **QoS:** Fog computacional pode possibilitar que os níveis de *QoS* das aplicações sejam melhorados, uma vez que a latência pode ser diminuída e os recursos expandidos.

A Fog computacional é uma plataforma altamente virtualizada onde normalmente fornece serviços de computação, armazenamento e rede entre dispositivos finais e *Data Centers* de computação em ambientes de nuvem tradicionais. Mas, o processamento pode não ocorrer exclusivamente nos dispositivos finais da rede, pois esse processamento pode acontecer em camadas superiores, como nos dispositivos de rede por exemplo. Processamento, armazenamento e os recursos de rede são componentes fundamentais para a construção de sistemas de computação em nuvem e fog. Entretanto, diversas características fazem com que o desenvolvimento de uma arquitetura de fog computacional a partir de uma plataforma de computação em nuvem não seja uma tarefa trivial (BONOMI et al., 2012). Alguns dos desafios existentes estão listados abaixo:

- Conhecimento prévio de localização dos dispositivos e baixa latência de comunicação entre os mesmos.
- Dispositivos localizados em ambientes altamente distribuídos.
- Grande volume de dispositivos.
- Suporte à mobilidade, como por exemplo, carros conectados a rede, *smartphones*, *smartwatches* e etc.
- Processamento em tempo real.
- Acesso predominantemente *Wireless*.
- Heterogeneidade. Dispositivos de fog computacional tendem a ser os mais variados, com diferentes tipos de dados gerados, protocolos de comunicação, *hardware*, sistema operacional, entre outros.
- Interoperabilidade e virtualização. Em algumas situações, como por exemplo transmissão de vídeo em tempo real, requerem a cooperação de diferentes provedores. Nesse sentido, os serviços precisam ser virtualizados entre os diferentes domínios pois fog computacional precisa ser capaz de interoperar entre eles.

2.3 NETWORK FUNCTIONS VIRTUALIZATION

As redes de computadores são constituídas de diversos serviços, muitos dos quais são implementados em *software e hardware* verticalmente integrados e proprietários (*e.g., firewall, Network Address Translation,* sistemas de detecção de intrusão), onde já não satisfazem mais a necessidade de escalabilidade da rede para atender os provedores de serviços de redes e de telecomunicações. Isso acontece frequentemente devido às limitações dos fabricantes de equipamentos de redes, onde os mesmos geralmente não possuem interoperabilidade com equipamentos de diferentes fornecedores, forçando a aquisição de quase toda a infraestrutura de rede de um mesmo fabricante (HERRERA; BOTERO, 2016).

Estes serviços representam uma importante fração das despesas de capital (*CAPital EXpenditures - CAPEX*) e despesas operacionais (*OPerational EXpenditures- OPEX*) de uma rede, já que cada um desses componentes requer hardware especializado (HERRERA; BOTERO, 2016). Além disso, muitos destes serviços de rede produzidos em *hardware* são complexos para projetar, integrar, gerenciar e solucionar problemas, o que torna uma raridade encontrar profissionais com habilidades necessária para desempenhar tal tarefa. Além disso, esses *hardwares* atingem rapidamente o fim de sua vida útil, ficando sem novas atualizações (CHIOSI et al., 2012). Por fim, pode existir um desperdício dos recursos disponíveis no sistema, uma vez que esses equipamentos físicos não fornecem meios para ajustar ou compartilhar seus recursos de maneira dinâmica, de acordo com a demanda. Por

exemplo, se um determinado equipamento está utilizando metade dos recursos disponíveis, estes poderiam ser alocados para outra tarefa de um outro serviço.

Devido à todas estas dificuldades encontradas, várias empresas da área de telecomunicações se uniram e definiram um novo conceito para o provimento de novas soluções e serviços de redes de computadores. Então, a partir disso, surge o paradigma NFV. Este propõe a virtualização das funções de rede em hardware de uso geral.

O conceito de NFV surge para solucionar estes problemas visando transformar a maneira como as operadoras de redes arquitetam suas infraestruturas, simplificando o projeto, desenvolvimento e gerência dos diversos serviços. Através de técnicas de virtualização, serviços de rede são disponibilizados por meio de dispositivos virtuais, denominados de VNFs, que são executados em hardware genérico *e.g.*, arquitetura x86. Dessa forma, a inclusão de novos serviços, bem como a alteração e atualização de serviços já existentes, não necessitam da instalação de equipamentos de *hardware* especializados, diminuindo custos e flexibilizando o provisionamento de serviços, que podem estar localizados em data centers ou no usuário final (CHIOSI et al., 2012).

As funções de rede podem ser implementadas e executadas em uma variedade de servidores padrões da indústria utilizando uma camada de virtualização. Essas funções são iniciadas sob demanda e removidas quando não são mais necessárias, podem ser movidas ou instanciadas em vários locais da rede, conforme for necessário, sem a necessidade de instalação de novos equipamentos (CHIOSI et al., 2012) .

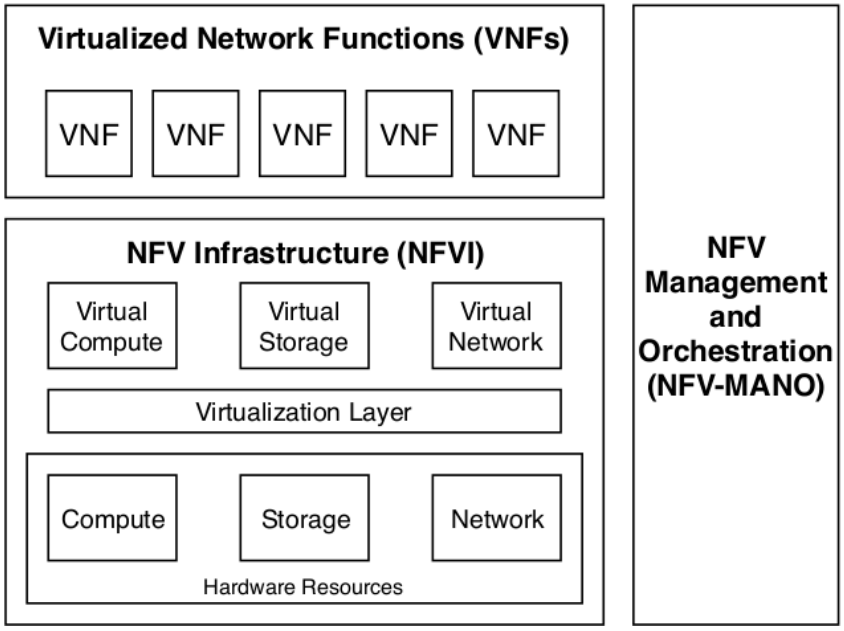
A abordagem baseada em NFV apresenta diversas vantagens em comparação com as redes tradicionais, tais como uma melhor eficiência de capital em comparação com implementações de *hardware* dedicados e uma maior flexibilidade na criação de VNFs em *hardware* de propósito geral. Isto ajuda tanto na escalabilidade como em dissociar a funcionalidade da localização, o que permite que o *software* esteja localizado nos locais mais apropriados. Também possibilita uma rápida inovação de serviços por meio da implantação desses serviços baseados em *software*, melhor eficiência operacional resultante de procedimentos comuns de automação e operação, e.g, economia de energia e espaço físico. Além disso, a flexibilidade do gerenciamento da rede também melhora, pois há uma padronização e abertura de interfaces entre funções de rede virtualizadas e a infraestrutura com as entidades de gerenciamento, de modo que esses elementos possam ser oferecidos por diferentes fornecedores de forma desacoplada, fazendo com que as VNFs possam ter seus recursos ajustados de maneira automática, aumentando ou diminuindo de acordo com a demanda necessária, fazendo uso eficiente dos recursos do sistema (ETSI-GS-NFV, 2013).

2.3.1 Arquitetura de Referência NFV

De acordo com o ETSI, a arquitetura NFV é dividida em alguns blocos funcionais. Esses blocos permitem a instanciação dinâmica de funções de rede virtuais, o relacionamento,

provisionamento de recursos e a conectividade entre as instâncias de VNFs. Essa arquitetura é composta por três principais domínios de trabalho: Funções de Rede Virtualizadas, Infraestrutura NFV (NFVI) e Orquestração e Gerenciamento NFV (NFV-MANO), conforme ilustrado na Figura 4 (ETSI-GS-NFV, 2013). Essa figura representa a arquitetura de alto nível, mostrando os principais elementos e suas interações:

Figura 4 – Arquitetura NFV alto nível proposta pela ETSI.



Fonte: (ETSI-GS-NFV, 2013)

- **Funções de Rede Virtualizadas:** São implementações de *software* de funções de rede executadas sobre a infraestrutura NFV. Implementações essas que podem incluir funções de rede convencionais, tais como, *switching*, *routing*, servidor DHCP, detectores de intrusão, *firewall*, entre outros. Um fator importante é que o comportamento funcional e o estado de uma função de rede na grande maioria das vezes devem ser independentes do fato dela ser virtualizada ou não. Espera-se então que o comportamento funcional e as interfaces operacionais externas devem ser os mesmos, tanto em uma implementação física quanto em uma virtualizada. Uma VNF pode ser composta por vários componentes internos. Por exemplo, uma VNF pode ser implementada em várias VMs, onde cada VM hospeda um função de rede específica. No entanto, em outros casos, todas as funções de rede podem ser implementadas em uma única VM.

- **Infraestrutura NFV:** É a combinação de todos os componentes de *hardware e software* que criam o ambiente no qual torna-se possível implantar, gerenciar e executar as VNFs. Além do mais, conforme pode-se observar na Figura 4, os recursos físicos presentes nesse bloco incluem elementos de rede, elementos de armazenamento e elementos de computação. Os componentes de *software* são abstrações dos

elementos de *hardware*. Essas abstrações são obtidas por meio de uma camada de virtualização, como um *hypervisor*, que desacopla os recursos virtuais dos recursos físicos subjacentes.

Já a infraestrutura para o provimento do (*NFV-Infrastructure - NFVI*) pode ser distribuída por diferentes localidades. A rede que fornece conectividade entre esses locais é considerada parte da infraestrutura do NFV. No entanto, a camada de virtualização e os recursos de *hardware* se apresentam com uma única entidade para as VNFs, fornecendo-lhes os recursos virtualizados desejados.

- **Orquestração e Gerenciamento NFV:** De acordo com o (ETSI-GS-NFV, 2013), o NFV-MANO é responsável por prover funcionalidades para o gerenciamento dos serviços, orquestrando e gerindo os recursos de infraestrutura e recursos de *software* para as VNFs. Este orquestrador inclui também, a orquestração e o gerenciamento do ciclo de vida das VNFs como instanciação, monitoramento, atualização, consulta e finalização. Vários gerentes VNF podem ser implantados; Gerente de VNF podem ser implantados para cada VNF, ou um gerente de VNF podem servir a múltiplas VNFs. Em suma, NFV-MANO é responsável por todas as tarefas específicas que envolvam o gerenciamento dos recursos de virtualização dentro da arquitetura NFV.

2.4 SOFTWARE DEFINED NETWORKING

Nos últimos anos a demanda por um ambiente flexível e dinâmico que acompanhe a velocidade dos acontecimentos e inovações tecnológicas através de uma maior facilidade e agilidade para monitorar e aplicar novas configurações nos elementos da rede tem sido uma tendência. Desta necessidade, começaram a surgir propostas para que os planos de dados e de controle em uma infraestrutura de rede fiquem fisicamente separados. Esse é o princípio básico das redes SDNs.

O SDN separa o plano de dados, que compreende os dados que atravessam as conexões com fio e sem fio; o plano de controle é um sistema logicamente centralizado comumente chamado de controlador que regula o encaminhamento de dados e também tem a visão global da rede (KREUTZ et al., 2014). Esses recursos contornam as limitações de uma decisão de roteamento totalmente distribuída, que pode ser onerosa em relação a mecanismos de mensagens e configuração, redundância de dados e tende a ser abaixo do ideal. Assim, a SDN cria as rotas apropriadas para os dados, seja na solicitação ou na resposta, tanto na conexão com fio, ou seja, aquela que conecta a infraestrutura da nuvem com a (*Base Station - BS*) e RSU, e no *link* sem fio controlando os parâmetros das interfaces.

As redes SDN se caracterizam por conceber uma nova arquitetura para redes de computadores que permite a implementação de funcionalidades sem a necessidade de intervenção do fabricante do equipamento e com alto grau de flexibilidade, já que estas funcionalidades podem ser implementadas no *software* do controlador. Elas compõem um conjunto

de abordagens para redes de computadores visando permitir que administradores de rede gerenciem os serviços de forma mais simples, mediante uma abstração que desacopla o plano de dados do plano de controle. Esta arquitetura foi projetada para permitir que redes sejam controladas de forma mais fácil, eficiente e que os recursos de rede sejam gerenciados, utilizando algoritmos de controle sob medida para as demandas dos usuários sobre a rede (SHIN; NAM; KIM, 2012).

As SDNs (ONF, 2019a) tem ganhado muita atenção da comunidade acadêmica e da indústria na área da computação e redes em geral. Grandes empresas como Google, Intel, Dell, Samsung, entre várias outras, são parceiros da (*Open Networking Foundation - ONF*) (ONF, 2019c), com o principal objetivo de incentivar a adoção da SDN como padrão aberto.

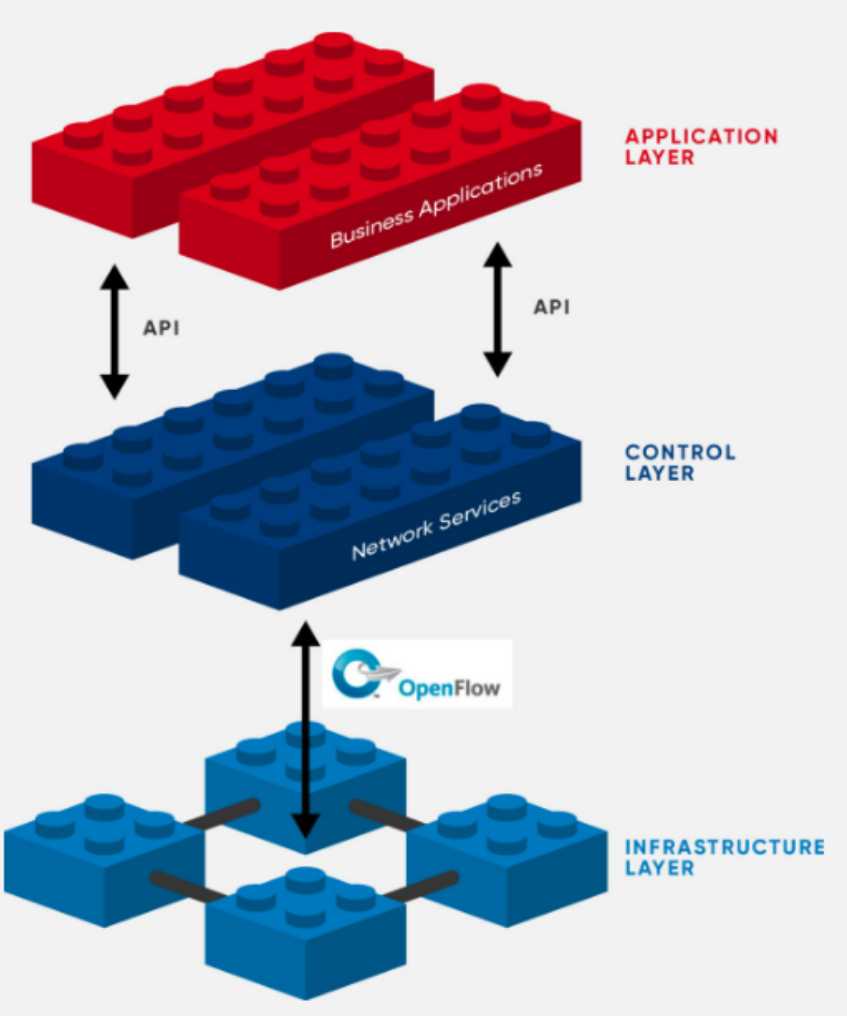
Em resumo a SDN é uma arquitetura emergente, dinâmica, gerenciável e adaptável, o que a torna ideal para a natureza dinâmica de alta largura de banda necessária para atender as aplicações de hoje. Essa arquitetura separa as funções de controle e encaminhamento de rede, permitindo que o controle de rede se torne diretamente programável e a infraestrutura subjacente seja abstraída para aplicativos e serviços de rede. SDN facilita a implantação de novos protocolos e aplicações, simplifica e flexibiliza o gerenciamento da rede e possibilita múltiplas opções de controle quanto aos seus fluxos de tráfego, dentre outros, sendo amplamente favorável à inovação e evolução da rede, de modo geral. O protocolo OpenFlow é um elemento fundamental para a construção de soluções SDN (ONF, 2019c). A Figura 5 mostra uma visão simplificada da arquitetura SDN.

Como pode ser visualizado na figura, a programação entre o plano de dados e o de controle para realizar a separação entre eles, é feita usando uma (*Application Programming Interface - API*) de comunicação definida entre os *switches* e o controlador SDN fazendo uso do protocolo *Openflow*. O controlador exerce um controle direto sobre os elementos do plano de dados por meio da API (BIZANIS; KUIPERS, 2016), dessa forma o controlador fica responsável por fazer as decisões relevantes para que o plano de dados execute apenas encaminhamento de pacotes para o destino mais adequado, fazendo com que seja possível ter um melhor desempenho e configurações eficientes, tornando as redes flexíveis para acompanhar e suprir as necessidades das tecnologias emergentes.

2.4.1 Protocolo OpenFlow

Um dos protocolos amplamente utilizados na implantação das redes SDN é o *OpenFlow*, que é aberto e padronizado para a comunicação entre controladores e equipamentos de comutação (MCKEOWN et al., 2008). Trata-se de um protocolo de baixo nível que permite implementar o controle dos nós da rede por meio da definição de uma interface externa para a troca de mensagens entre os dispositivos de rede e os controladores. O protocolo *OpenFlow* habilita o conceito SDN, permitindo que os administradores possam definir fluxos de dados e determinar quais caminhos esses fluxos devem percorrer independente do *hardware* que está sendo utilizado (LARA; KOLASANI; RAMAMURTHY, 2013).

Figura 5 – Visão simplificada da arquitetura SDN



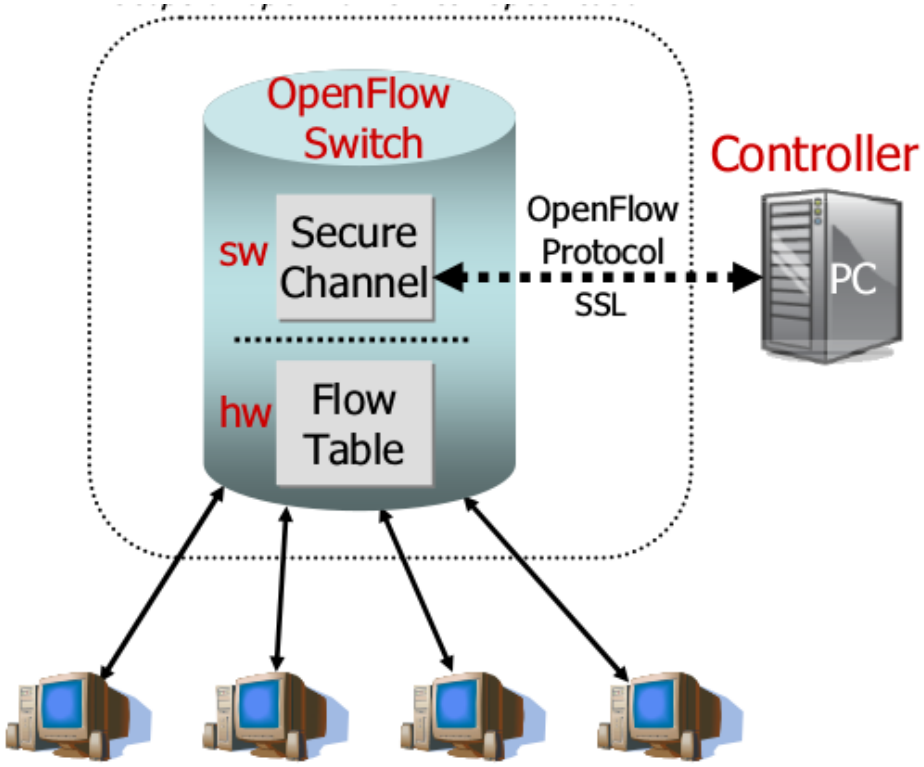
Fonte: (ONF, 2019c)

O protocolo *OpenFlow*, por ser baseado em um padrão de código aberto busca a interoperabilidade entre diferentes fabricantes e permite controlar, gerenciar e monitorar equipamentos que suportem esse protocolo, com decisões de encaminhamento dos pacotes sendo feitas de maneira centralizada, possibilitando a construção de uma rede programada, independentemente dos comutadores de rede. A programabilidade é possível pela atualização da tabela de fluxos dos comutadores de rede (GUEDES et al., 2012). Alterando a tabela de fluxos, um controlador *OpenFlow* pode mudar o comportamento dos fluxos de pacotes que chegam ao comutador. Este equipamento possui três características principais (MCKEOWN et al., 2008), e podem ser melhores visualizados na Figura 6:

1. Uma **tabela de fluxo** ou mais ficará a cargo do administrador da rede, onde uma ação pode ser associada a cada entrada de fluxo, determinando como processar cada fluxo;

2. Um **canal seguro** que liga o equipamento ao controlador remoto, permitindo que os comandos e pacotes de rede sejam comunicados de forma segura;

Figura 6 – Especificação do *Switch OpenFlow*



Fonte: (MCKEOWN et al., 2008)

3. O **protocolo *OpenFlow*** que fornece uma forma aberta e padronizada para as comunicações entre controlador e comutador *OpenFlow*.

Um *switch OpenFlow* possui uma sequência de tabelas de encaminhamento, que definem como os quadros são tratados (ONF, 2019b). Cabe ao controlador definir como estas tabelas são preenchidas e assim definir como os fluxos se comportarão na rede. O controlador cria entradas na tabela, que são aplicadas caso os cabeçalhos do pacote sigam o padrão especificado. No *switch* há um processo de verificação que analisa cada tabela buscando uma regra para o quadro recebido. Três situações podem ocorrer a partir deste processo:

1. Encaminhar o fluxo de entrada para uma determinada interface de saída, caso haja uma entrada equivalente para o fluxo em questão em sua tabela de fluxos. A segunda possibilidade é que os *switches* enviem mensagens *Packet-In* do tipo assíncrona para o controlador quando não existe nenhuma regra para um determinado pacote.

2. Caso não haja entrada equivalente na tabela de fluxos ao receber o pacote. O controlador irá decidir o que fazer com esse pacote. Inclusive, decidirá se uma entrada na tabela de fluxos do *switch* será adicionada ou não. Assumindo que ele irá fazer um encaminhamento, uma alteração é realizada para o encaminhamento do destino resultando em uma mensagem *Flow-Mode*. O estado do *switch openflow* é modificado

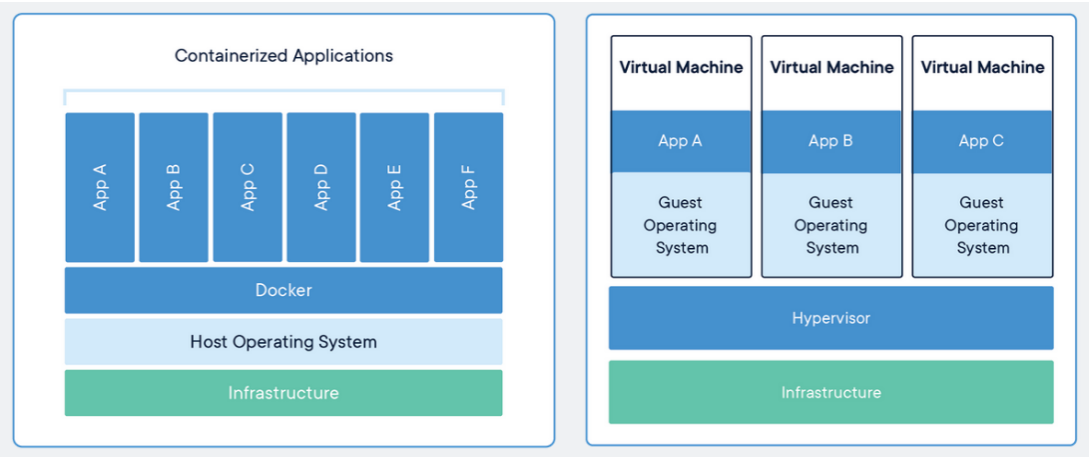
por ela. Em seguida, o controlador envia uma mensagem *Packet-Out* informando que os pacotes estão prontos para serem encaminhados.

3. Por último, o *switch OpenFlow* também deve ser capaz de descartar pacotes. Como o espaço para armazenar as regras é limitado, quando existe alguma regra de fluxo ociosa, elas precisam ser removidas. Portanto, um *Flow-Removed* é aplicado para remover uma entrada de fluxo de uma tabela.

2.5 TECNOLOGIA DE *CONTAINERS*

Os *containers* (DOCKER, 2019b) são uma estratégia de virtualização em nível de sistema operacional. Essa técnica é baseada no isolamento de recursos específicos do *kernel* para que aplicações tenham a impressão de estarem executando em um ambiente individual. A arquitetura básica para o funcionamento de *containers* é ilustrado na Figura 7. Um aspecto importante mostrado nessa figura é que o *kernel* do sistema operacional é compartilhado entre aplicações em máquinas virtuais e aplicações em *containers*. Entretanto, a forma de acesso ao *kernel* (do ponto de vista das aplicações) é diferente, pois o conjunto de aplicações (*Bins*) e bibliotecas (*Libs*) do sistema não é necessariamente o mesmo oferecido pelo sistema operacional nativo, pois cada *container* é responsável por suas próprias configurações independente do *host* hospedeiro.

Figura 7 – Comparação entre *containers* e Máquinas Virtuais



Fonte:(DOCKER, 2019d)

De acordo com o projeto Docker,*container* e máquinas virtuais possuem isolamento de recursos e benefícios de alocação similares. Contudo, uma diferente abordagem arquitetural permite que *containers* sejam mais portáveis e eficientes(DOCKER, 2019a).

Essas diferenças podem ser melhor percebidas na Figura 7, que ilustra as camadas presentes entre os recursos físicos de uma infraestrutura computacional e as aplicações que ali estão sendo executadas. Os *containers* são uma abstração na camada de aplicativo que agrupa código e dependências. Vários *containers* podem ser executados na mesma

máquina e compartilhar o *kernel* do sistema operacional com outros *containers*, cada um sendo executado como processos isolados no espaço do usuário. Os *containers* ocupam menos espaço que as VMs (as imagens de *container* geralmente são medidas em MBs). Já as máquinas virtuais são uma abstração do *hardware* físico, transformando um servidor em vários servidores. O *hypervisor* permite que várias VMs sejam executadas em uma única máquina. Cada VM inclui uma cópia completa de um sistema operacional, aplicação, binários e bibliotecas. Além disso, as VMs também podem ser lentas para inicializar quando comparadas com *containers*.

Assim sendo, *containers* podem ser mais eficientes pois são executados diretamente no sistema operacional hospedeiro, eliminando a necessidade de um sistema operacional visitante e toda a memória que seria consumida por ele. Além disso, em ambientes nos quais o uso de virtualização se faz necessário, *containers* podem ser executados em máquinas virtuais. Neste cenário, uma estratégia possível é a alocação de menos máquinas virtuais do que tradicionalmente se utiliza, sendo que cada máquina virtual pode passar a gerenciar uma quantidade maior de recursos físicos com a tecnologia de *container*, garantindo o isolamento entre processos.

2.5.1 Docker

O Docker foi a plataforma escolhida para fazer a execução de *containers* neste trabalho. Existem outras implementações capazes de executar *containers*, tais como o Rkt [1], o LXC [2] e o OpenVZ [3], contudo, nenhuma delas entrega todos os recursos oferecidos pelo Docker.

O Docker é a tecnologia padrão que de fato melhor explora a virtualização baseada em *container*. Ele fornece a capacidade de empacotar qualquer aplicação com todas as suas dependências (por exemplo, bibliotecas, binários, arquivos de dados, etc.) em um *container* Docker isolado. O Docker também i) permite limitar os recursos atribuídos a um *container* em termos de memória e CPU (por padrão, um *container* não possui restrições de recursos e utiliza todo recurso disponível pelo *host* hospedeiro), e ii) fornece funcionalidades para verificação e restauração de um *container* em execução(DOCKER, 2019b).

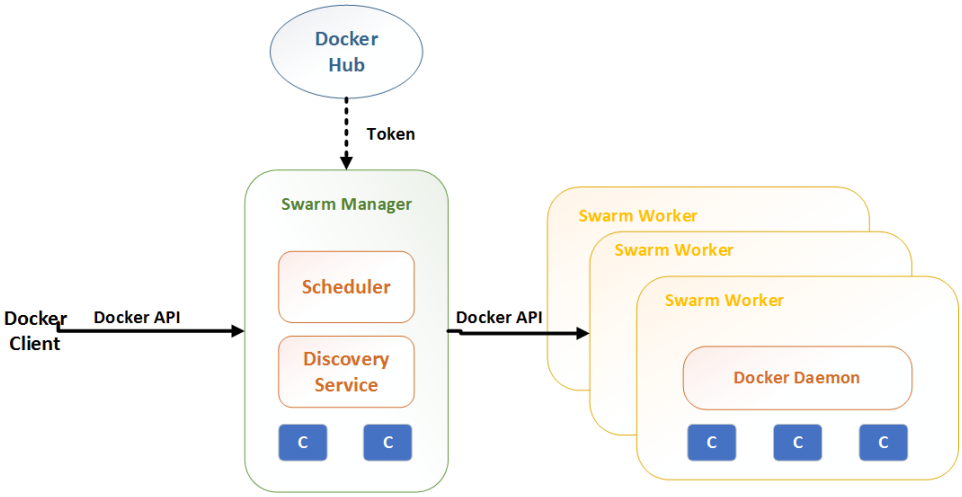
Um *container* do Docker é criado a partir de uma imagem Docker. A partir de uma única imagem Docker, um ou mais *containers* podem ser iniciados. O Docker também permite procurar imagens existentes em vez de criá-las a partir do zero. As imagens podem ser armazenadas em registros do Docker (por exemplo, Docker Hub(DOCKERHUB, 2019)), onde podemos deixar essas imagens privadas ou públicas onde outros usuários podem recuperá-las e usá-las. Os registros do Docker facilitam a distribuição de imagens com diferentes configurações e em diferentes ambientes.

[1] https://coreos.com/rkt/docs/latest/
[2] https://linuxcontainers.org/
[3] https://github.com/OpenVZ

Os *containers* do Docker podem se comunicar usando o *container* Docker em rede (DOCKER, 2019c). No momento na instalação do Docker são criados alguns *drivers* de rede por padrão onde um *container* conectado à uma rede poderá se comunicar com todos os outros da mesma rede. O Docker oferece vários *drivers* de rede, dependendo se os *containers* residem em um único *host* ou em um *cluster* de *hosts*. Soquetes ou *bridges* por padrão também podem ser usados como mecanismos de baixo nível para implementar um canal de comunicação entre os *containers*.

O Docker também possui ferramentas de orquestração incorporadas que permitem implantar aplicações em vários *containers*. Por exemplo, o Docker *compose* (DOCKER, 2019a) permite criar e gerenciar *containers* do Docker em um único *host* ou em um *cluster* de *hosts*. Existem outras diferentes técnicas avançadas para gerenciar os *containers* implantados em uma ou em um conjunto de máquinas, como *Swarm* [4], que é uma ferramenta do próprio Docker, Kubernetes [5], Apache Mesos [6] e assim por diante.

Figura 8 – Arquitetura Docker Swarm Alto Nível.



Fonte: O Autor (2019)

A Figura 8 mostra a arquitetura de alto nível da funcionalidade do orquestrador *Swarm*, bem como seus componentes principais. Com o Docker *Swarm* cada tarefa é mapeada para um *container*. Ao utilizar esse orquestrador podemos controlar as decisões de posicionamento do *container* usando rótulos (*tags*) definidos pelo usuário ou definidos pelo sistema e passar essas tarefas via API. O *scheduler* também leva em consideração restrições de CPU e memória ao agendar *containers* em determinados *hosts*. Além disso, o *Swarm* introduziu o Docker *Service*. Ao usar o Docker *Service*, o *scheduler* monitora continuamente o estado desejado. Por exemplo, ele pode monitorar o número de *containers* em execução e age quando o estado desejado está em violação, em caso de falha de um nó, reiniciando *containers* em outro *host*.

[4] https://docs.docker.com/engine/swarm/
[5] https://kubernetes.io/pt/
[6] http://mesos.apache.org/

A descoberta de serviço requer três processos importantes. Primeiro, é um tipo de serviço DHCP que permite que os serviços obtenham dinamicamente endereços (*Internet Protocol - IP*) uns dos outros e assim consigam se comunicarem na rede. A segunda é uma forma de verificação de integridade para garantir que apenas instâncias saudáveis recebam tráfego, e a terceira é o balanceamento de carga para garantir que todos os containers que executam uma determinada aplicação obtenham tráfego roteado para eles.

Docker *Hub* é a maior biblioteca e comunidade do mundo de imagens para *containers*, onde podemos encontrar imagens já pré-configuradas que atendam nossa necessidade, enviar imagens para esse repositório ou utilizar imagens locais em nossas máquinas. A conexão entre o gerente *Swarm*, e os nós *work* acontece via API, onde o mesmo faz uma checagem se a conectividade entre os nós está ocorrendo corretamente. Após isso ocorre um mapeamento dos serviços e tarefas que serão executadas. Esses serviços são compostos por tarefas, e podem receber comandos individuais e também comandos específicos para todos os *containers*. O *daemon* que está em execução no nós *work* retornam a confirmação que todos os serviços e tarefa foram criados e conectados.

O orquestrador escolhido para esse trabalho foi o *Swarm* pois o gerenciamento de *containers* através do *Swarm* é eficiente e leve quando comparado a outras técnicas, além de ser um projeto do próprio Docker. Ele converte todos os *hosts* do Docker em um único *host* virtual. Todos os nós criados no dispositivo de borda são controlados, programados, monitorados e limpos usando esse gerenciador de *clusters*. O gerenciador de *clusters* também nos ajuda a manter uma verificação da utilização de recursos, é capaz de criar um conjunto de *containers*, volumes de dados, redes e senhas (definição dada a proteção aplicada a informações sensíveis, como senhas e chaves privadas) por meio de um arquivo de configuração. Outras características do Docker *Swarm* são (DOCKER, 2019e):

- **Gerenciamento integrado ao Docker** – A partir de uma instalação Docker é possível iniciar um *cluster* Docker *Swarm*, não sendo necessário nenhum *software* adicional de orquestração para gerenciar o *cluster*;

- **Arquitetura Descentralizada** – Em vez de lidar com a diferenciação entre funções de nós no momento da implementação, o Docker lida com qualquer especialização em tempo de execução. Podemos implantar os dois tipos de nós, gerentes e os de processamento, usando o Docker *Engine*. Isso significa que podemos construir um *cluster* inteiro a partir de uma única imagem de disco.

- **Modelo de serviço declarativo** – O Docker usa uma abordagem declarativa para permitir que possamos definir o estado e funções desejadas dos vários serviços em sua pilha de aplicações.

- **Escalabilidade** – Para cada serviço, podemos declarar um número de tarefas que desejamos executar. Quando aumentamos ou diminuimos a escala de processamento

dessas tarefas, o gerenciador de *clusters* se adapta automaticamente adicionando ou removendo tarefas para manter o estado desejado.

- **Conciliação de estados desejada** – O nó do gerenciador de *clusters* monitora constantemente o estado do *cluster* e reconcilia quaisquer diferenças entre o estado real e o estado desejado expresso.
- **Rede *multi-host*** – Podemos especificar uma rede de sobreposição para novos serviços. O gerenciador de *clusters* atribui automaticamente endereços aos *containers* na rede de sobreposição quando inicializa ou atualiza a aplicação.
- **Descoberta de serviço** – Os nós gerentes do *cluster* atribuem a cada serviço em execução um nome DNS exclusivo e balanceadores de carga executando em *containers*. Onde podemos consultar todos os *containers* em execução no *cluster* por meio de um servidor DNS incorporado no mesmo.
- **Balanceamento de carga** – O Docker *Swarm* nos possibilita duas maneiras de configurarmos um balanceador de carga, onde podemos expor as portas de serviços a um balanceador de carga externo, ou internamente, o *swarm* permite especificar como distribuir a carga de trabalho entre os *containers* que fazem parte do *cluster*.
- **Seguro por padrão** – Cada nó no *swarm* impõe autenticação mútua e criptografia TLS para proteger as comunicações entre ele e todos os outros nós.
- **Atualizações contínuas** – No momento da distribuição, podemos aplicar atualizações de serviço aos nós de forma incremental. O gerenciador do *cluster* nos permite controlar as versões entre a implementação do serviço para diferentes conjuntos de nós. Se algo der errado, poderemos retroceder uma tarefa para uma versão anterior do serviço.

2.6 REDES PEER-TO-PEER (P2P)

Uma rede P2P (LUA et al., 2005);(ANDROUTSELLIS-THEOTOKIS; SPINELLIS, 2004);(PARAMESWARAN; SUSARLA; WHINSTON, 2001) é um sistema de auto-organização escalável no qual os nós cooperam para formar uma rede de entrega de conteúdo. Os nós em um sistema P2P são chamados de pares *peers*, pois podem atuar como clientes, ao fazerem o *download* de um determinado conteúdo de outras pessoas e também servidores, ao fornecer conteúdo a outros pares que fazem parte do *cluster* de compartilhamento. Geralmente, os pares são estações de trabalho pessoais de usuários comuns tais como computadores pessoais e *notebooks* conectados à internet. Os pares são consumidores e servidores ao mesmo tempo. Neste modelo de compartilhamento de conteúdo os usuários finais não acessam diretamente apenas poderosos servidores localizados em *data centers*, pois todos os pares

contribuem com partes de seus recursos como (largura de banda de rede, armazenamento, etc.) e geralmente têm a mesma capacidade e responsabilidade.

O precursor das redes P2P no que tudo indica é o *USENET*, desenvolvido em 1979. Era um sistema que permitia aos usuários ler e postar mensagens/notícias. Era um sistema de rede semelhante aos fóruns *on-line* que temos atualmente, mas com a diferença de que o *USENET* não dependia de um servidor ou administrador central. O *USENET* copiava a mesma mensagem/notícia para todos os servidores encontrados na rede. Da mesma forma que as redes P2P distribuem seu conteúdo.

O início da história para o grande sucesso do P2P ocorreu no ano de 1999 nos Estados Unidos, quando o *Napster*[7] ganhou vida. O *Napster* era um *software* de compartilhamento de arquivos que era usado por pessoas para baixar e compartilhar músicas. A música compartilhada no *Napster* era geralmente protegida por direitos autorais e, portanto, ilegal para sua distribuição. No entanto, isso não impediu que as pessoas conseguissem fazer tal ato. Embora o *Napster* tenha sido o que colocou as redes P2P em visibilidade para o mercado e para os usuários como um todo, esse projeto acabou fracassando e foi fechado pelas autoridades por causa de todo o conteúdo que foi compartilhado ilegalmente nele.

Diferentemente do (*Content Delivery Network - CDN*), um fato interessante sobre as redes P2P é a falta de necessidade de uma infraestrutura dedicada para seu funcionamento. Pois qualquer par pode participar da tarefa de entrega de conteúdo e, normalmente, não é necessário nenhum controlador central. O único requisito técnico é que todos os pares executem um programa específico para estarem preparados para ingressar em uma rede P2P. As redes P2P são escaláveis por natureza, já que sempre novos membros podem participar do *cluster* de compartilhamento. Contudo, nada impede que redes CDN e P2P trabalhem em conjunto.

Como não é esperado que estações de trabalho pessoais fiquem *on-line* 100% do tempo enviando determinado conteúdo, já que seus proprietários podem desligá-la ou pode ocorrer também das máquinas falharem, não há garantia de disponibilidade de recursos individuais. Assim, os sistemas P2P estão preparados para aumentar a probabilidade de disponibilidade de conteúdo em servidores tracker, esses servidores auxiliam na comunicação entre dois computadores que utilizam o protocolo P2P. Além de, manter cópias em diferentes pares que o solicitaram recentemente. Portanto, as redes P2P são tolerantes a falhas porque as falhas em um par são suprimidas por outros.

Essa abordagem de redes P2P mostra-se ser bastante apropriada para as VANETs, dada sua topologia altamente dinâmica. No entanto, outras investigações devem ser conduzidas para propor soluções eficientes para a descoberta e entrega de conteúdo com o objetivo de fornecer um serviço que atenda aos requisitos mínimos de latência e qualidade de serviço.

[7] https://br.napster.com/

2.7 CONSIDERAÇÕES FINAIS

Este capítulo apresentou um resumo dos principais conceitos que fundamentam a criação da plataforma que será apresentada nessa proposta. Inicialmente foram apresentados os principais conceitos relativos as tecnologias constituintes da proposta, seguido dos conceitos que envolvem as ferramentas e o ambiente possibilitando ao leitor conhecer de forma básica os conceitos que fundamentam esse trabalho. O Capítulo 3 apresentará um resumo de alguns trabalhos relacionados aos temas expostos neste Capítulo.

3 TRABALHOS RELACIONADOS

Este capítulo descreve trabalhos relacionados encontrados na literatura durante a pesquisa desta dissertação. Os trabalhos foram divididos em três categorias que são apresentadas em Seções distintas deste Capítulo. Na Seção 3.1 são discutidas as propostas que tratam de esquemas de implementação da arquitetura MEC. A Seção 3.2 apresenta trabalhos com estratégias de implementação de Fog computacional tanto para redes veiculares quanto para dispositivos que não necessitam de mobilidade. A Seção 3.3 discute os trabalhos que abordam arquiteturas usando MEC e fog para VANETs e IoT. Por fim, a Seção 3.4 apresenta as considerações finais deste capítulo.

3.1 SOLUÇÕES BASEADAS EM MEC

A computação MEC surge como uma evolução da computação em nuvem cujo objetivo é levar a hospedagem de aplicações de *data centers* centralizados para a borda da rede, mais próxima dos consumidores e dos dados gerados pelos aplicativos. Esse modelo de arquitetura é reconhecido como um dos principais pilares para atender aos exigentes indicadores para a tecnologia 5G, especialmente em termos de baixa latência e eficiência de largura de banda. O MEC oferece aos desenvolvedores de aplicações e aos provedores de conteúdo recursos de computação em nuvem e um ambiente de serviços de TI na borda da rede gerenciado e orquestrado para facilitar a criação e integração de novos produtos.

Em busca de usufruir desse novo padrão, os autores (HSIEH; LEE; CHEN, 2018) adotaram cenários de IoT baseados em infraestrutura MEC para análise de desempenho do mecanismo de controle de fluxo de dados. O objetivo dessa proposta é reduzir o congestionamento da rede, a latência ao fazer requisições à aplicação e fornecer aos usuários uma *QoS* mais adequada. Foi implementado um *gateway* IoT para fazer comunicação entre um dispositivo IoT (*Raspberry Pi*) e as aplicações em execução na MEC através de funções de rede virtualizadas. As informações geradas e repassadas ao *gateway* IoT determinam a decisão de política e ação tomada no tráfego que contribuiu para a execução das regras. Os autores seguiram o padrão ETSI-MEC para desenvolver uma plataforma virtualizada de rede com duas funções principais: (*Traffic Offloading Function - TOF*) para controle de sessão e tráfego, e (*Radio Network Information Service - RNIS*) para monitoramento e utilização eficiente da força do sinal da rede sem fio. A plataforma virtualizada com funções TOF e RNIS foi integrada como mecanismo de controle de fluxo, o que ofereceu mais flexibilidade à rede. A implementação proposta de programação de recursos contribuiu para o modo de serviço e estratégias de controle de tráfego das aplicações, o que melhorou a latência de rede de entrega e experiência do usuário. Através do OVS-DPDK, onde os autores utilizaram o *OpenvSwitch*. No entanto, na proposta os autores não explicam

ou implementam a arquitetura NFV seguindo formalmente os padrões ETSI MANO, dividido em três blocos funcionais (NFV-MANO). **NFV Orchestrator:** Responsável pela integração de novos pacotes de serviços de rede (*Network Service - NS*) e VNF; Gerenciamento do ciclo de vida do NS; gestão global de recursos; validação e autorização de solicitações de recursos de NFVI. **VNF Manager:** Supervisiona o gerenciamento do ciclo de vida de instâncias de VNF; cumpre o papel de coordenação e adaptação para configuração e relatório de eventos entre NFVI e **Virtualized Infrastructure Manager (VIM):** Controla e gerencia os recursos de computação, armazenamento e rede do NFVI. A arquitetura MEC proposta pelos autores também não tem nenhum virtualizador de infraestrutura que faz com que não sigam completamente o padrão ETSI. Como também não utilizam nenhum virtualizador ou orquestrador no nível de infraestrutura de nuvem, pois fazem uso de containers diretamente em máquinas físicas, o que não torna uma arquitetura escalável à nível de recursos físicos. Esse problema de escalabilidade poderia ser resolvido caso fizessem o uso de máquinas virtuais. Também não utilizam nenhum controlador SDN na arquitetura.

No trabalho de (AL-BADARNEH et al., 2018), os autores tiveram como objetivo fornecer um suporte de entrega de conteúdo definido por software para sistemas com infraestruturas MEC em ambientes onde a mobilidade é considerada muito importante. Eles propuseram uma abordagem para permitir a prestação de serviços e conexão à aplicações com latência reduzida, utilizando a estratégia de busca cooperativa da MEC para comunicação entre veículos e infraestrutura V2I, bem como o uso de cache no nível veicular para comunicações entre os pares V2V. O protótipo do framework foi implementado como uma extensão do emulador Mininet-WiFi onde o conteúdo requisitado pelo veículo pode estar localizado na estação base. O suporte à arquitetura MEC proposta permitiu que as estações base forneçam serviços para veículos em movimento na rodovia com baixa latência e alta largura de banda. Um dos problemas que pode ser percebido no trabalho dos autores é que eles não levaram em consideração a padronização da arquitetura MEC fornecida pelos documentos da ETSI, onde uma solução de orquestração de três camadas precisa ser totalmente adaptada à implantação do MEC: infraestrutura, plataforma e aplicativos. Ainda sobre os documentos de padronização, os autores citam que para que todos os padrões sejam utilizados corretamente são fornecidas algumas considerações preliminares de alto nível para o desenvolvimento e a implantação de uma aplicação na infraestrutura MEC, tais como: solução baseada em DNS, nome do domínio, empacotamento da aplicação (em uma VM ou *container*) e que forneça metadados com requisitos da aplicação como (cpu, memória, disco, rede).Uma limitação da proposta dos autores é que a busca por veículos que detém o conteúdo requisitado ficará restrito ao raio de cobertura das RSUs ou do controlador SDN.

3.2 SOLUÇÕES BASEADAS EM FOG

Essa arquitetura consiste na alocação do poder de processamento mais perto do limite da rede ou nos próprios dispositivos que a compõe. Portanto, é uma arquitetura de computação descentralizada onde dados, cálculos, comunicações, armazenamentos, medições, aplicações e gerenciamentos são distribuídos no local mais adequado e eficiente: entre a fonte de dados, infraestruturas intermediárias e a nuvem.

Na pesquisa de (NOBRE et al., 2019), os autores investigaram os princípios arquitetônicos de uma arquitetura de rede veicular, (*Veicular Networking - VN*) baseada em SDN chamada (*Vehicle Software Defined Networking - VSDN*). Essa arquitetura é habilitada para fog computacional onde a nuvem é responsável por gerenciar e controlar os nós fog de maneira centralizada. Exploraram os princípios de design da arquitetura SDN, que foi projetada na fog computacional habilitada para redes veiculares. Os autores fizeram um processamento de vídeo na RSU que foram utilizadas como nós de fog computacional, onde recebem o vídeo do acidente e o repassaram pelas RSUs até o centro de atendimento médico mais próximo. Finalmente, a rota de resgate é definida pelo controlador e os veículos que estão respeitando a política de resgate de emergência são alertados para manter uma rota alternativa para cooperarem com os veículos de emergência. Os testes ocorreram tendo como base uma certa porcentagem de carros que aceitassem fazer uso dessa nova regra que foi passada a eles. Ou seja, os carros apenas decidem se irão utilizar uma determinada regra ou não, esse será seu poder de decisão. Por exemplo se somente 25% dos carros aceitassem essa nova regra seria uma certa velocidade no atendimento, e assim sucessivamente para 50% e 75% dos veículos. O problema é que os nós fog, que no caso são as RSUs foram colocadas sempre na mesma posição para que pudessem cobrir todo o cenário de testes. Os autores não levaram em consideração casos em que essas RSUs ficasse sem comunicação.

No trabalho de (SUN; ANSARI, 2016), os autores propuseram uma arquitetura IoT flexível, chamada edgeIoT, onde eles fizeram o uso do paradigma de fog computacional e de redes definidas por software SDN para coletar, classificar e analisar os fluxos de dados da IoT na borda móvel. O objetivo principal da edgeIoT foi aproximar os recursos de computação que estavam conectados às antenas celulares para que pudessem ser executados o processamento dos dados próximo dos dispositivos IoT, o tráfego na rede principal pudesse ser aliviado e o atraso de ponta a ponta entre recursos de computação e dispositivos de IoT fosse minimizado. Dessa forma, projetaram uma arquitetura de computação de fog computacional de maneira hierárquica para fornecer provisionamento de recursos de computação flexível e escalável para cada usuário, bem como para cada provedor de serviços de IoT, propuseram também um esquema de migração de máquina virtual de *proxy* para minimizar o tráfego na rede principal. O problema desta proposta é que para fazer esse processamento dos dados eles utilizaram dois modelos de VM, e os dois modelos acabam não sendo algo tão funcional por eles não deixar claro como isso poderá ser feito,

passando assim a responsabilidade para o cliente ter em execução essas VMs próximas de sua residência. A primeira VM (*proxy*) estará em execução localmente nas residências ou próximas dos dispositivos IoT, ou seja, acarretará custos a quem quiser sempre manter uma VM em execução para responder às requisições e fazer o processamento ou pré-processamento dos dispositivos. O segundo modelo de VM (*proxy*) será para responder às requisições de dispositivos com mobilidade onde elas ficarão em execução próximas à estações celulares. O problema é que os autores trabalham apenas com máquinas virtuais e não com arquiteturas orquestradas assim como a proposta de nosso trabalho, pois migrar VM torna-se um processo custoso, e esse problema poderia ser resolvido migrando ou replicando apenas a aplicação, como pode ser feito em microsserviços de uma infraestrutura à outra, e não a máquina virtual por completa.

Em (LI et al., 2019), os autores propuseram um esquema de melhoria na qualidade de serviço baseado nos procedimentos de transmissão existentes para apoiar os serviços veiculares em tempo real, através da migração de serviço nas redes celulares habilitadas para fog computacional. Os autores realizaram um estudo de caso baseado em um padrão realista de mobilidade de veículos para o cenário da cidade de Luxemburgo. A proposta é comparada com valores de referência e as análises consideram latência e confiabilidade, bem como o custo de migração das máquinas virtuais onde as aplicações estavam em execução. Em resumo os autores propuseram uma estrutura em redes celulares habilitada para veículos conectados, um esquema de *QoS* aprimorando o procedimento de transferência de dados, e fizeram um estudo sobre a troca de informações para a migração de serviço bem como o custo de migração onde foram investigados em termos de frequência de migração e tempo de migração. A migração de serviço foi executada através da realocação de serviços de um servidor de Fog computacional para outro de maneira dinâmica, onde assumiram essa proposta como uma solução eficaz para o problema de mobilidade. Na maioria das vezes a migração de uma VM, implica em parar a VM e consequentemente o serviço que está em execução para que assim possa fazer a migração. Além disso, é necessário uma boa taxa de transmissão para que possa migrar a máquina virtual de uma infraestutura para outra. Os próprios autores citam que para realizar a migração de serviço contínuo e sem interrupções, a conectividade entre os nós de fog computacional, no caso a (*Base Station - BS*) deve ser aprimorada por meio de links de alta capacidade como os baseados em fibra. Além disso, para melhorar o desempenho da rede, são necessárias outras técnicas de virtualização para reduzir o tamanho dos dados relacionados à migração.

A arquitetura de (SANTORO et al., 2017) denominada Foggy tem como objetivo gerenciar o posicionamento da carga de trabalho entre uma infraestrutura de nuvem e uma fog computacional satisfazendo os requisitos de aplicações implantadas nos dispositivos finais da rede que, por sua vez, solicitam acesso e usam recursos e serviços oferecidos por essa infraestrutura. Os autores descrevem Foggy como uma plataforma de *software* para

orquestração de carga de trabalho e negociação de recursos em um sistema de computação em nuvem multicamada, altamente distribuído, heterogêneo e descentralizado, onde o foco principal é em dispositivos IoT. Contudo, tal arquitetura não provê mecanismos de gerenciamento para todas as camadas, chegando apenas até os nós de computação que fazem parte da própria infraestrutura de nuvem que foram realocados para a borda. Essa configuração acarretará em muita troca de mensagens entre os nós de computação e o controlador da infraestrutura, ocasionando uma sobrecarga de dados na rede. Além de deixarem de lado os dispositivos finais (sensores, *smartphones* e câmeras), como conforme as principais definições que prevêem de fog (YI; LI; LI, 2015), pois esses dispositivos também podem fazer parte do processamento. Os autores propõem uma arquitetura para orquestração de carga de trabalho entre as camadas de nuvem e *edge* baseada nas principais tecnologias do mercado (*Openstack e Kubernetes*).

Em uma proposta de arquitetura participativa dos usuários denominada User-Participatory Fog (KIM; CHUNG, 2018), os clientes podem adquirir dispositivos fog como *Cloudlets* e pontos de acesso (*Access Point - AP*), registrá-los em um serviço online e, assim, prover computação, armazenamento e rede para aplicações descentralizadas. Na arquitetura proposta, a otimização de posicionamento de instância de serviço de fog computacional é executada com base no uso de serviço dos usuários participantes, que é formulado em um problema de programação não linear inteira mista (MINLP) e, em seguida, linearizada. A arquitetura proposta e o método de posicionamento do serviço de fog são avaliados com base em simulações levando em consideração os parâmetros reais dos serviços e dispositivos da IoT. Apesar disso, esse modelo não descreve claramente qual seria a integração entre os vários níveis fog (núcleo e borda).

3.3 SOLUÇÕES INTEGRADAS MEC E FOG

Na pesquisa de (VILALTA et al., 2017), os autores propuseram uma arquitetura chamada TelcoFog como uma infraestrutura de fog computacional, onde a mesma propõe ser uma arquitetura segura, altamente distribuída e densa, que pode ser alocada na extremidade de uma rede com ou sem fio para que uma operadora de telecomunicações forneça múltiplos recursos unificados e econômicos para os novos serviços 5G, como NFV, MEC e serviços para terceiros (por exemplo, cidades inteligentes e IoT). Essa arquitetura tenta demonstrar que por seu modelo, é capaz de integrar um ecossistema para operadoras de rede dispostas a fornecer serviços NFV, MEC e IoT, onde o principal benefício da TelcoFog é a implantação dinâmica de novos serviços distribuídos de baixa latência. Em resumo, este trabalho apresenta os componentes básicos de uma nova arquitetura, que fornece a possibilidade de implantação dinâmica de serviços IoT na borda da rede. Diferentemente da proposta Edge-On, Telcofog foca em dispositivos IoT fixos em vez de veículos. Outra característica da arquitetura proposta é que não fazem qualquer tipo de processamento nos dispositivos finais, pois fazem uso de um *gateway* IoT ao qual enviam os dados até

esse dispositivo da rede. Outra grande diferença é que em nossa proposta concebemos e exemplificamos toda a sinergia que existe entre o padrão das arquiteturas NFV e MEC bem como toda a orquestração à nível de infraestrutura, serviços de rede e aplicações que poderão ser executados na borda da rede, tanto na infraestrutura MEC quanto nos próprios dispositivos/veículos.

No trabalho de (NING; HUANG; WANG, 2019), inicialmente os autores criaram um modelo baseado na arquitetura de computação veicular na névoa (*Vehicular Fog Computing - VFC*) de três camadas (*cloud – cloudlet – fog*) para gerenciar o tráfego de uma cidade de maneira distribuída. Depois disso, investigaram um esquema para o gerenciamento de tráfego através de VFC em tempo real. Especificamente, utilizaram a camada de fog computacional através de veículos estacionados e em movimento, onde utilizaram os veículos para oferecer recursos de computação para processamento de informações em busca de trazer melhoria no gerenciamento de tráfego e segurança no trânsito. Para os nós de fog computacional baseados em veículos móveis, o modelo correspondente seguiu um sistema de enfileiramento, onde as trocas de mensagens no sistema segue um procedimento de *Poisson*. Uma das críticas ao trabalho dos autores é que eles não citam ou trabalham com tecnologias facilitadoras do 5G, como NFV, SDN além de utilizarem paradigmas computacionais como cloudlets, que é um paradigma precursor aos atualmente estudados como MEC e possível facilitador juntamente com outras tecnologias.

O trabalho de (LE et al., 2017) apresenta uma proposta de computação na borda que abrange uma infraestrutura *Edge* e continua até os dispositivos finais que fazem uso da tecnologia P2P. O principal elemento da proposta é apresentar uma arquitetura com tolerância a falhas na rede, que podem ser comuns em ambiente dinâmicos das redes veiculares. Seu maior trunfo, entretanto, é a maior fragilidade do trabalho, pois a resiliência pretendida, assim como boa parte do controle de requisição-resposta, fica sob responsabilidade de um *middleware publisher-subscriber* específico que se encontra na infraestrutura *Edge*. A arquitetura como um todo não faz uso de tecnologias e padrões especificados no mercado e na academia para tais infraestruturas. Uma das especificações citadas por (SABELLA et al., 2019) é que nestes modelos de arquitetura que façam uso de Edge, exista também uma camada superior, ou seja, uma infraestrutura de cloud para que exista uma redundância caso ocorra uma falha nesse servidor intermediário (MEC). Na proposta, a tolerância a falhas da arquitetura existe porque após a tentativa de requisição ao servidor de borda (*edge*) não ser bem sucedida, os próprios nós através de uma rede P2P fornecem uma forma de processamento até que a conexão com a infraestrutura *Edge* seja restabelecida. Um problema pode ser que para os requisitos de determinadas aplicações não seja possível elas serem executadas nos dispositivos finais através de P2P.

Por fim, podemos observar na Tabela 2 uma comparação dos trabalhos relacionados que foram utilizados nesta dissertação.

Neste trabalho, o uso do paradigma SDN mostra ganhos significativos na programa-

Tabela 2 – Comparação dos Trabalhos Relacionados

Proposta	Problema	SDN	NFV	Edge (MEC, MCC)	Containers	Computação nos dispositivos finais	Suporte à Mobilidade	Orquestração	Ambiente de Testes
(HSIEH; LEE; CHEN, 2018)	QoS	Sim	Sim	Sim	Sim	Não	Não	Sim	Simulação
(AL-BADARNEH et al., 2018)	Latência	Sim	Não	Sim	Não	Sim	Sim	Não	Emulação
(NOBRE et al., 2019)	Congestionamento em acidentes	Sim	Não	Não	Não	Não	Sim	Não	Simulação
(SUN; ANSARI, 2016)	Mobilidade e Latência	Sim	Não	Sim	Não	Não	Sim	Não	Simulação
(LI et al., 2019)	Mobilidade e Latência	Não	Não	Não	Não	Não	Sim	Não	Simulação
(SANTORO et al., 2017)	Balanceamento	Não	Não	Não	Sim	Não	Não	Sim	Simulação
(KIM; CHUNG, 2018)	Posicionamento	Sim	Não	Não	Sim	Não	Não	Sim	Simulação
(VILALTA et al., 2017)	Latência	Sim	Sim	Sim	Sim	Não	Não	Sim	Simulação
(NING; HUANG; WANG, 2019)	Gerenciamento de Tráfego	Não	Não	Sim	Não	Sim	Sim	Não	Modelagem
(LE et al., 2017)	Tolerância a Falha	Não	Não	Sim	Não	Sim	Sim	Não	Simulação
Edge-On	Mobilidade Balanceamento e Latência	Sim	Sim	Sim	Sim	Sim	Sim	Sim	Emulação/ (Testbed)

Fonte: O Autor (2019)

bilidade e flexibilidade no gerenciamento da mobilidade dos veículos e em suas aplicações executadas internamente. A arquitetura NFV é utilizada para orquestrar, virtualizar as funções dos controladores, *switches*, reduzir custos e permitir a execução de vários serviços VNFs em infraestrutura construída por meio da computação em nuvem e MEC. A utilização da tecnologia em containers demonstra maior agilidade na criação, replicação ou migração de aplicações, e a fog computacional através da arquitetura P2P demonstra excelentes resultados na diminuição da latência em comparação a nuvem, bem como tolerância a falhas em uma possível desconexão no momento que estiver adquirindo um conteúdo, pois esse conteúdo muito provavelmente estará contido em outros veículos na rede ao qual o veículo poderá se comunicar.

3.4 CONSIDERAÇÕES FINAIS

Este capítulo apresentou uma discussão sobre os trabalhos relacionados à criação de novas arquiteturas e plataformas envolvendo *cloud, edge e fog computing* para solucionar problemas de latência, suporte ao caráter dinâmico, *QoS* e balanceamento de carga de trabalho entre os diferentes níveis de arquitetura para redes veiculares. Os vários trabalhos mostraram a relevância desta pesquisa. Onde foi possível visualizar que o uso de tecnologias facilitadoras do 5G poderão contribuir para evolução tecnológica e arquitetural que dê suporte as redes VANETs.

4 PLATAFORMA PROPOSTA

Este capítulo tem por finalidade descrever a proposta da plataforma *Edge-On*. Uma plataforma que faz uso de de *softwares* para orquestração e gerência de recursos em um sistema de computação em nuvem/borda/névoa (*Cloud/Edge/Fog Computing*) multicamada. A Seção 4.1 apresenta uma descrição da proposta, Seção 4.2 instanciação dos serviços de rede e das aplicações na nuvem, Seção 4.2.1 o fluxo de instanciação dos serviços de rede e das aplicações na nuvem, Seção 4.3 fluxos de instanciação dos serviços de rede e das aplicações na MEC, Seção 4.4 comunicações interveicular e proposta de uma OBU para comunicação e processamento de dados, e a Seção 4.5 apresenta as considerações finais.

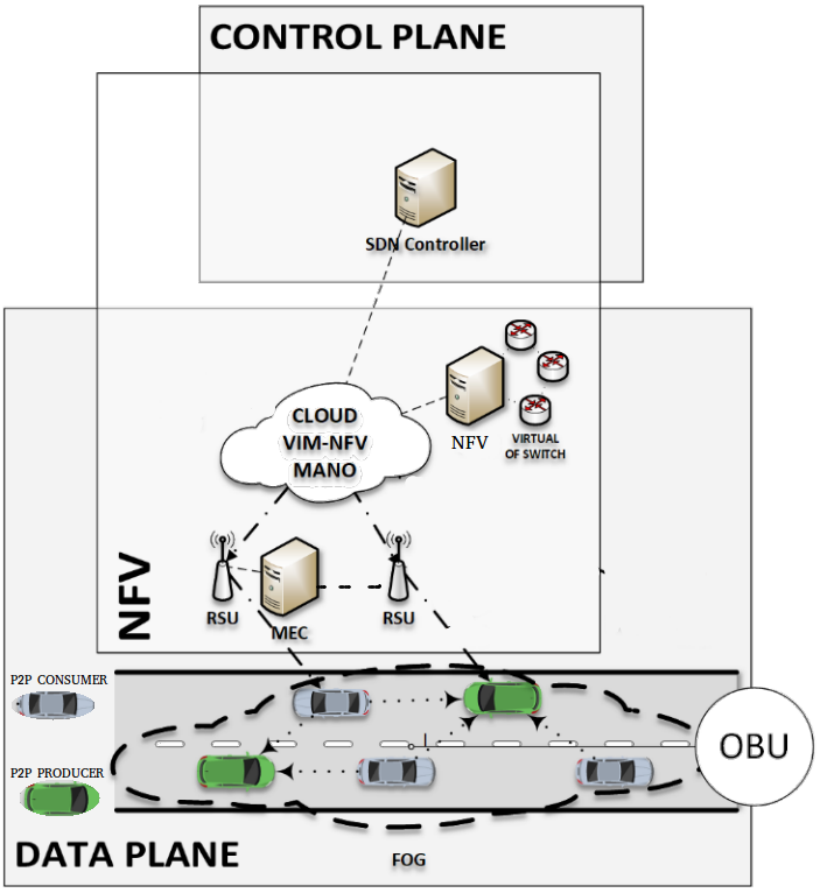
4.1 DESCRIÇÃO DA PLATAFORMA

A plataforma Edge-On estende o modelo baseado em infraestrutura para comunicações veiculares seguindo os princípios de computação *Fog*. Edge-On está distribuída em três níveis de computação: plataforma em nuvem, MEC e Veículos. Aproveitamos os recursos computacionais dos sistemas finais, ou seja, os veículos, para permitir o balanceamento da carga de trabalho entre estes, a infraestrutura do MEC e a nuvem. Serviços e aplicações funcionam como microsserviços que são executados em containers, onde esses serviços poderão ser migrados, replicados ou reposicionados com maior facilidade. *Switches* virtuais são criados onde seu ciclo de vida é orquestrado pelo (*NFV-Orchestrator - NFVO*) e a gerencia do fluxo de trabalho fica por parte do controlador SDN. O orquestrador NFV é responsável pela gestão da arquitetura virtualizada e ciclo de vida das instâncias, e o orquestrador de container gerencia a comunicação e implantação de aplicativos na nuvem, na MEC bem como, nos próprios veículos. SDN também foi explorado nas RSUs e *Switches*, habilitados para *OpenFlow*.

A plataforma provê compartilhamento de arquivos e processamento distribuído visando melhorar a execução de aplicativos e reduzir restrições temporais, tais como latência para determinadas aplicações. Em vez de trabalhar como pontos passivos, ou seja, apenas como consumidores, os componentes localizados nos veículos atuam como microsservidores capazes de executar serviços em containers, tornando-se produtores de informação. Eles também podem responder à solicitações de computação geradas de outros veículos, de usuários ou de dispositivos IoT. Essas camadas de computação(FOG e MEC) reduzem as distâncias físicas e lógicas entre veículos e aplicativos. O estágio atual da plataforma nos permite balancear e distribuir a carga de trabalho entre os vários provedores do mesmo serviço, formando uma infraestrutura de compartilhamento e processamento de conteúdo distribuído.

Em busca de uma melhor visualização de todas as camadas de nossa proposta, dividi-

Figura 9 – Modelo de Alto Nível Edge-On

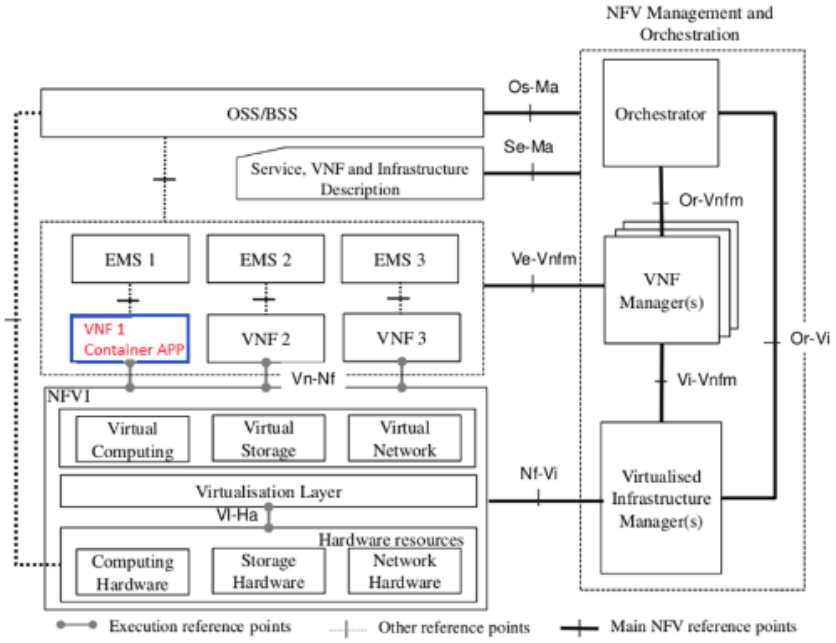


Fonte: O Autor (2019)

mos essas camadas para mostrarmos a sinergia entre as tecnologias utilizadas. A Figura 9 mostra a arquitetura em alto nível. O controlador SDN(SDN Controller) é centralizado como uma VNF na nuvem, bem como o *switch* virtual ao qual os veículos estarão conectados. No nível da visão de redes veiculares definidas por *software*, o controlador SDN também será instanciado como uma VNF na infraestrutura de nuvem a qual irá pertencer ao plano de controle e ficará responsável por gerenciar e orquestrar as comunicações veiculares. O plano de dados é composto das comunicações entre veículos, RSUs, infraestrutura MEC e infraestrutura de nuvem. A proposta da plataforma virtualiza o plano de controle para uma maior flexibilidade caso tenha a necessidade de aumentar os recursos da VNF ou na criação de uma outra. Na arquitetura, o orquestrador (*NFV-Orchestrator - NFVO*) é responsável por consultar os recursos que o (*Virtualized Infrastructure Manager - VIM*) possui e quais políticas de VNFs foram definidas, que assegurem uma instanciação eficiente de acordo com o número mínimo e máximo de máquinas a serem criadas. No nível intermediário está a infraestrutura MEC que também estará fazendo uso do padrão NFV para orquestrar seus recursos físicos bem como o ciclo das instâncias que forem criadas nesse ambiente. Por fim, podemos enxergar a camada da FOG computacional composta pelos próprios veículos em uma arquitetura P2P, distribuída onde os mesmos poderão es-

tar fazendo requisições a infraestrutura de nuvem, MEC ou aos próprios veículos vizinhos que tenham o conteúdo requisitado em *cache*.

Figura 10 – Implementação do padrão NFV na arquitetura de nuvem

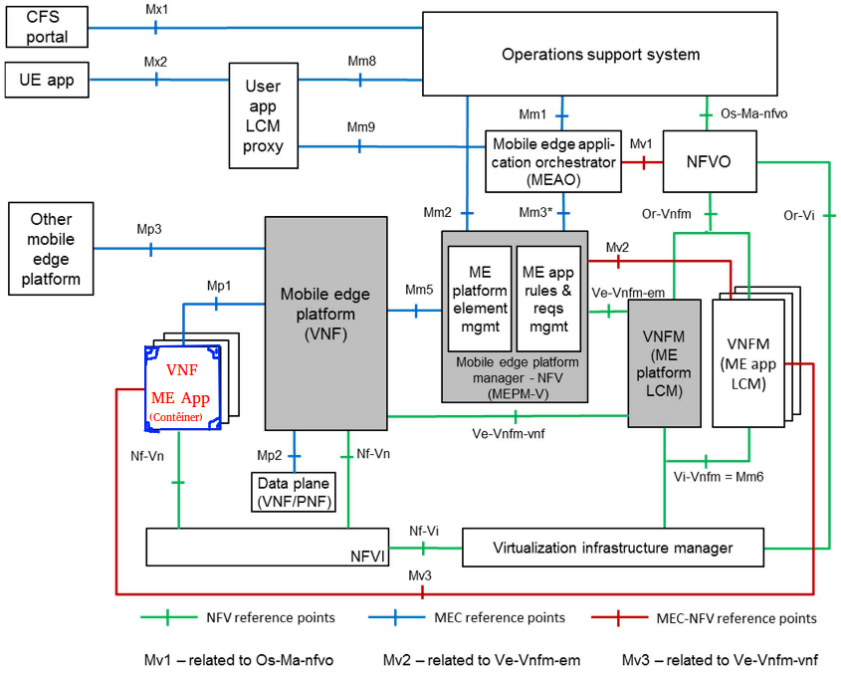


Fonte: Imagem moficada pelo Autor (ETSI-GS-NFV, 2013)

Para a Implementação da infraestrutura de nuvem privada, foi utilizada como ferramenta de gerenciamento de nuvem a plataforma *OpenStack* em sua versão *Ocata*. Essa plataforma nos possibilitou fazer uso do orquestrador NFV. O padrão NFV foi integrado através do orquestrador *Tacker*, que é um projeto interno comumente chamado de serviço do próprio *OpenStack*. Com esse ambiente foi possível gerenciar a infraestrutura física (rede, processador, disco) e prover recursos às instâncias, bem como orquestrar a criação de nossas funções de redes virtuais (controlador,*swithes*). Também podemos visualizar na caixa azul da Figura 10 que nesse ambiente foi possível orquestrar as aplicações que estavam sendo executadas em containers através da ferramenta de *software Docker*.

A Figura 11 mostra a sinergia do padrão da infraestrutura MEC quando gerenciada pelo orquestrador NFV(ETSI-GR-MEC-017, 2018). Podemos visualizar de forma bem definida a comunicação entre os orquestradores NFV e de *containers*(caixa azul), onde conseguimos integrar e fazer uma interação entre eles, permitindo o balanceamento de carga entre os diferentes níveis da arquitetura. Os orquestradores estão estrategicamente posicionados na infraestrutura de nuvem e na MEC. Assim, eles podem controlar a distribuição da aplicação entre os três níveis de computação. Utilizamos mecanismos de NFV e *containers* para que a orquestração se estendesse aos veículos de forma transparente. Assim, os veículos precisam atender a um requisito especial: eles precisam executar *switches* virtuais (*OpenFlow*) gerenciáveis com interfaces que sejam conectadas aos orquestradores, para que possamos gerenciar os veículos, aplicações e manter a rede sob controle.

Figura 11 – Implementação NFV e container na arquitetura de MEC



Fonte: Imagem modificada pelo Autor (ETSI-GR-MEC-017, 2018)

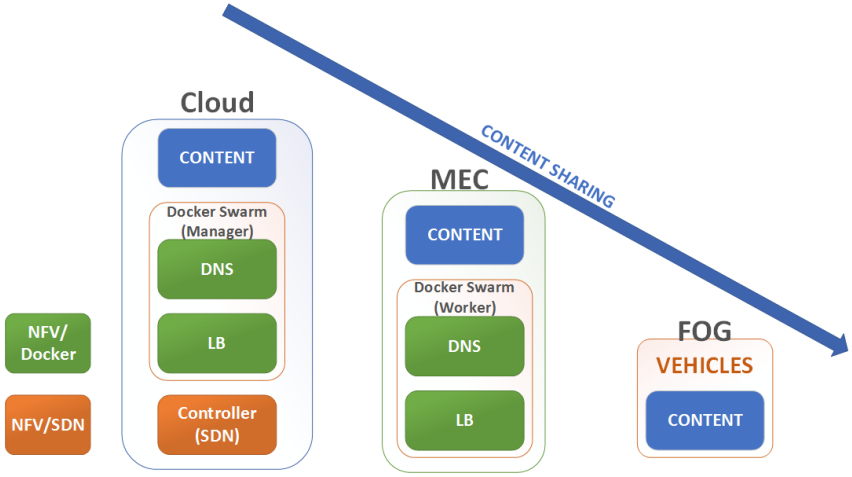
Como podemos ver na Figura 12, inicialmente as aplicações e os serviços complementares, como o balanceador de carga e serviço DNS, são executados na nuvem. Em certas situações, como novos requisitos, novas aplicações ou um novo acordo de nível de serviço (*Service Level Agreement - SLA*), a aplicação precisará ser executada em um servidor MEC. Assim, a arquitetura Edge-On transferirá o *template* NFV juntamente com as configurações do *Docker Swarm* da infraestrutura de nuvem para a MEC.

Inicialmente, o NFVI transforma os nós de processamento, de armazenamento e a infraestrutura de rede da infraestrutura física em uma camada de virtualização que aloca os recursos necessários para cada função virtualizada da arquitetura da nuvem e da arquitetura da MEC.

A camada MEC trabalha em conjunto com a infraestrutura de nuvem. A Figura 9 apresenta o host MEC conectado à RSU. A principal função da MEC é aproximar os serviços e a camada de aplicativos às camadas de veículos. Assim, o orquestrador NFV da nuvem pode enviar um template contendo as informações necessárias para que o aplicativo seja replicado e executado na borda da rede MEC. Neste momento, a aplicação que está sendo executada e os serviços complementares como balanceador de carga e DNS passarão a ser executados na MEC e trocarão informações com quem está sendo executado na nuvem.

Podemos observar de forma clara e objetiva a utilização de microsserviços gerenciados pelo orquestrador de containers em nossa arquitetura na Figura 12, onde podemos dividir as aplicações em pequenas partes e assim replicar, migrar apenas o serviço específico que

Figura 12 – Migração/Replicação de Aplicações e Serviços



Fonte: O Autor (2019)

quisermos. Esses serviços podem ser balanceadores de carga, serviços de nomes (DNS), *dashboards* para a visualização e gerenciamento de recursos do *cluster*, bem como tornar decisões de tarefas e regras comumentes difíceis em outros ambientes em algo simples, tais como balanceamento e redirecionamento de tráfego para *hosts* específicos que compõem o *cluster* ou passagem de *tags*/parâmetros para criação de novos containers/serviços em uma localidade específica da infraestrutura.

Inicialmente, o *Swarm Manager* supervisiona a aplicação Web (*content*), balanceador de carga e DNS em diferentes containerss em execução em uma única VNF na nuvem. A qualquer momento, quando o veículo solicitar o aplicativo, o mesmo template NFV que está na nuvem migra para o MEC juntamente com as configurações do Docker. Dessa forma, o conteúdo solicitado ganha proximidade física com o veículo e a infraestrutura MEC faz o balanceamento de carga com a nuvem.

O gerenciador de *clusters* de container *Swarm* surge como uma camada de *software* que organiza os containers do Docker distribuídos por *hosts* virtuais (VNFs). Edge-On apresenta cada VNF como um *host* do Docker. Para fins de gerenciamento, um *cluster Swarm* sempre requer dois tipos diferentes de *hosts*: Gerentes e nós de processamento (*workers*). Os gerentes lidam com o agendamento de serviços, gerenciam APIs HTTP e mantêm informações sobre o estado do *cluster*. Ele monitora todos os nós no *cluster*, observa as rotinas internas quando um novo nó ou serviço junta-se ao *cluster*, recebe e aplica as atualizações impostas pelo administrador, entre muitas outras operações. Dentro da arquitetura Edge-On, o gerenciador também recebe e executa containers. Os *workers* recebem apenas containers para execução de tarefas sem função de gerência.

4.2 INSTANCIAÇÃO DOS SERVIÇOS DE REDE E DAS APLICAÇÕES NA NUVEM

Em um ambiente real, a nuvem *OpenStack* foi configurada com seus principais serviços: autenticação (*Keystone*), Imagem (*Glance*), computação (*Nova*), rede (*Neutron*), armazenamento(*Cinder*), orquestração (*Heat*) e Dashboard (*Horizon*). O NFV baseado na arquitetura *ETSI MANO* foi integrado à nuvem por meio do serviço *tacker* para gerenciamento e orquestração de VNFs e serviços de redes usando a arquitetura NFV.

Em ambas infraestruturas (Nuvem e MEC) o *OpenStack* é usado como gerenciador de infraestrutura virtual, de acordo com os padrões NFV chamado de VIM. O *tacker* usa uma topologia e especificação de orquestração para serviços de nuvem conhecida como *TOSCA*. O *tacker* possui uma API responsável pelo ciclo de gerenciamento das VNFs. As configurações do controlador SDN bem como das outras aplicações são definidas em *templates*, onde *CPU*, memória, disco, rede, políticas de segurança, e principalmente as imagens pré-configuradas com cada serviço e aplicação são passadas como parâmetros.

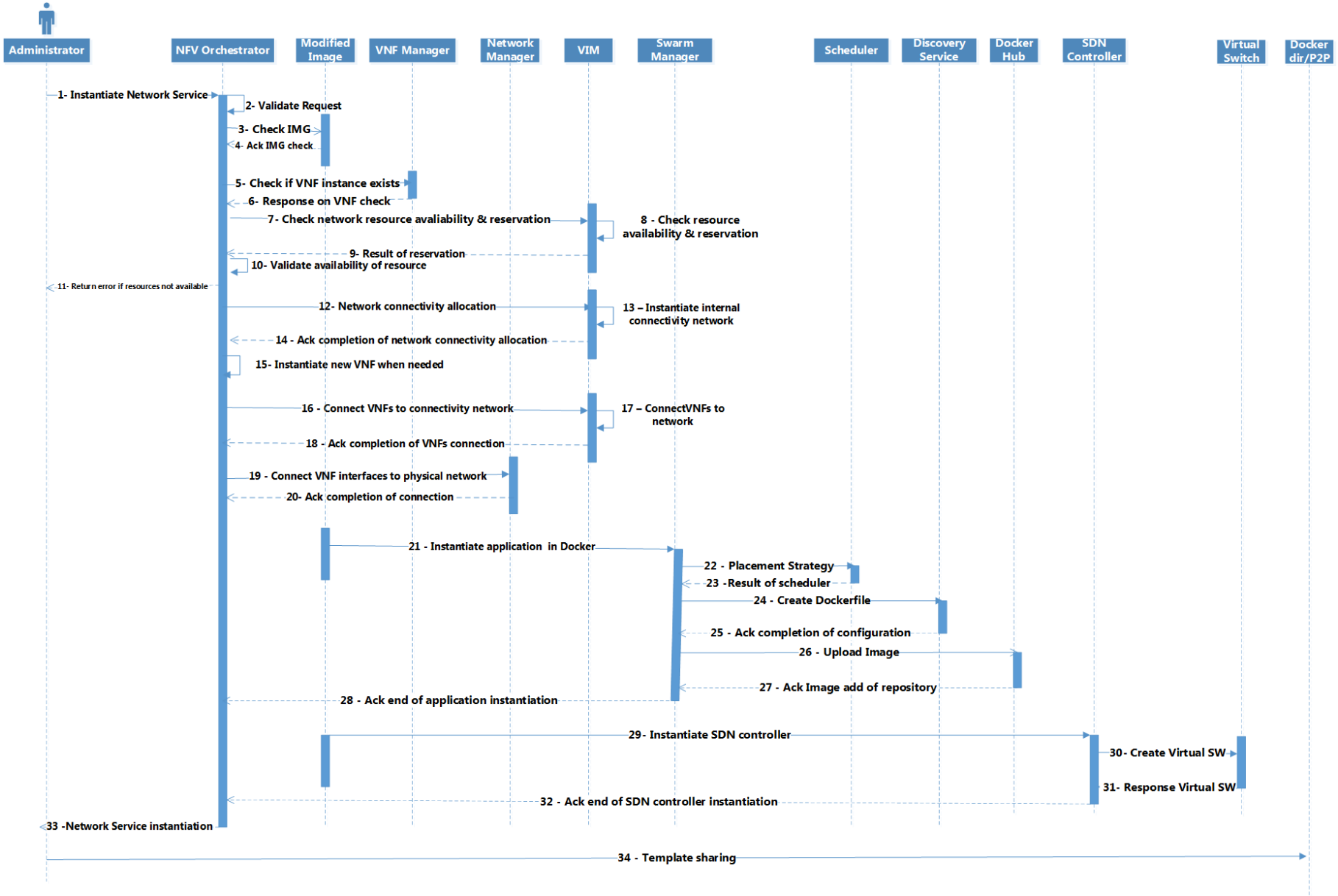
4.2.1 Fluxos de instanciação dos serviços de rede e das aplicações na nuvem

Na Figura 13 é mostrado um cenário mais detalhado do que acontece quando o (*NFV-Orchestrator - NFVO*) recebe uma solicitação para instanciar um novo serviço de rede. Essa solicitação pode vir de um administrador da nuvem, que faz uma solicitação de instanciação de um serviço de rede. O NFVO é o primeiro ponto de acesso para instanciação do serviço de rede simplificando a interface e a comunicação com o administrador.

O NFVO recebe uma solicitação para instanciar um novo serviço de rede usando a operação *Instantiate Network Service*, checa se existe a imagem que foi repassada no template e verifica com o Gerenciador de VNF se já existe uma instância que corresponda aos requisitos usando a operação de consulta VNF da interface do VNF *Lifecycle Management* (1-6). Se tal instância VNF existir, ela será usada como parte do Serviço de Rede. O NFVO solicita ao VIM disponibilidade dos recursos de rede necessários para a interconexão de uma nova VNF e reserva os recursos usando a operação de criar reserva da interface de gerenciamento de recursos virtualizados (7-18). Se necessário, o NFVO solicita que o *Network Manager* conecte as interfaces externas das VNFs às interfaces da rede física (19-20).

Será feito a inicialização dos serviços específicos que foram pré-configurados na imagem para utilizar serviços de rede e aplicações em containers (21-28). Além do mapeamento das tarefas que serão executadas. Com o Docker Swarm cada tarefa é mapeada para um container específico da tarefa. Após o passo anterior é feito a inicialização dos serviços específicos que foram pré-configurados na imagem para utilizar serviços do controlador SDN (29-32).

Figura 13 – Instanciação dos Serviços de Rede e das Aplicações na Nuvem



Fonte: O Autor (2019)

Por fim, o NFVO reconhece a conclusão da instanciação dos novos serviços de rede e é enviado o template pré-configurado para que outras instâncias possam ser criadas na infraestrutura MEC com as mesmas configurações que a Nuvem (33-34).

O passo 34 de nossa plataforma é o ponto chave de nossa proposta, pois como nosso principal objetivo é diminuir o tempo de resposta para que os veículos possam ter acesso ao conteúdo, não nos preocupamos em carga de trabalho ou sobrecarregar a aplicação. Pensamos na melhor forma que poderíamos migrar os serviços que estariam sendo executados na nuvem para a infraestrutura MEC e entre os próprios veículos. Sendo assim a estratégia criada por nós foi compartilhar apenas o template que foi utilizado para criar as instâncias e as aplicações. Assim, transferimos um template "*yaml*"que poderia ser utilizado pela MEC para iniciar rapidamente uma instância na MEC que será utilizada para fazer um balanceamento de carga entre os dois ambientes.

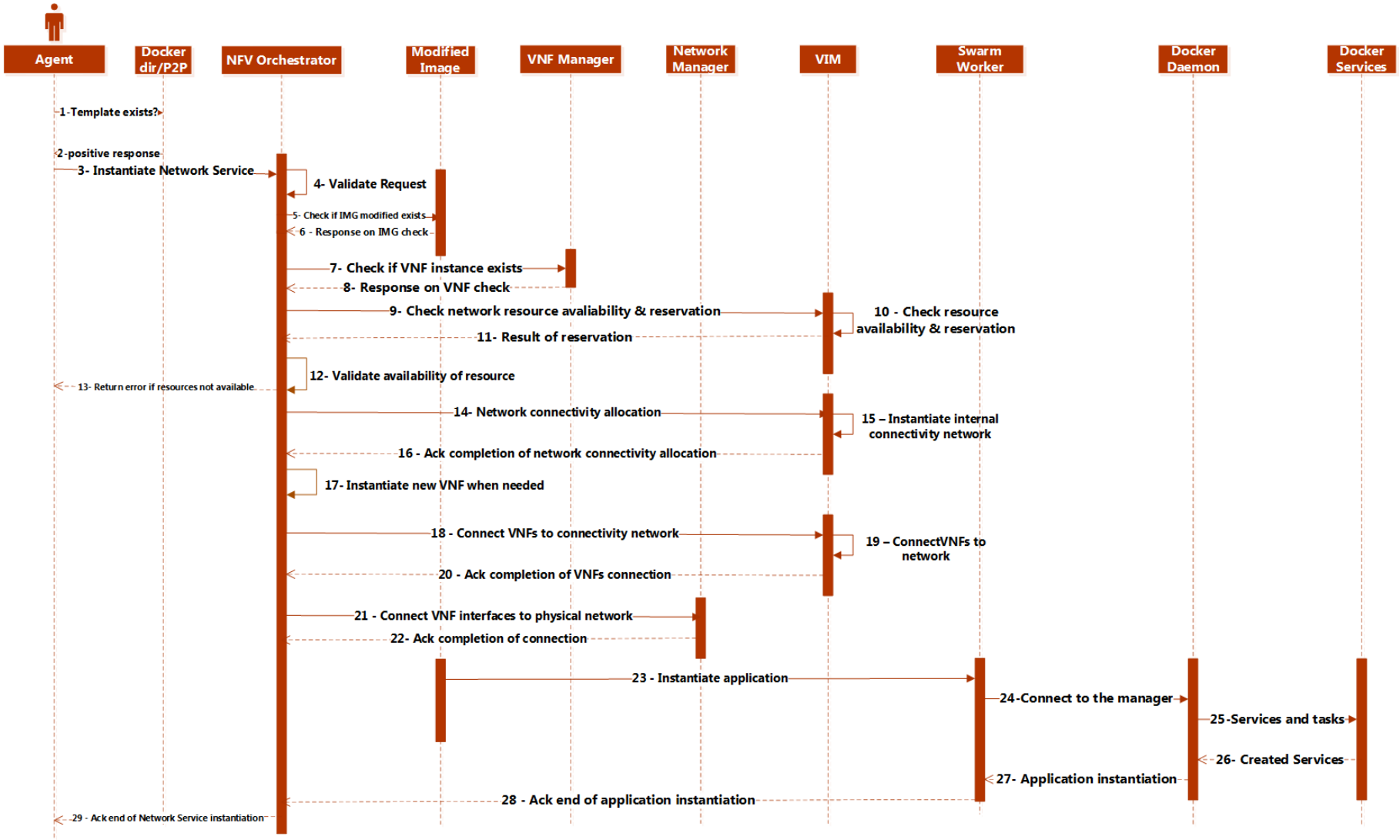
4.3 FLUXOS DE INSTANCIAÇÃO DOS SERVIÇOS DE REDE E DAS APLICAÇÕES NA MEC

Para a realização dos experimentos, uma infraestrutura MEC foi configurada com o *OpenStack* em sua versão *Ocata*. A arquitetura NFV foi integrada através do orquestrador Tacker. Com esse ambiente é possível orquestrar a criação dessas funções de redes virtuais. Nesse ambiente também é possível orquestrar as aplicações que estarão sendo executadas em containers através da ferramenta de software Docker.

Na Figura 14 é mostrado um cenário mais detalhado do que acontece quando o NFVO recebe uma solicitação para instanciar um novo serviço de rede. Essa solicitação vem a partir de um agente que estar monitorando um diretório que estar compartilhado com a nuvem. Quando esse agente encontra um novo template de configuração que foi compartilhado pela nuvem, ele repassará esse template para o NFVO. O NFVO é o primeiro ponto de acesso para instanciação do serviço de rede simplificando a comunicação. O cenário a seguir irá demonstrar como é feita a criação da instância e conexão dos serviços com ambiente da nuvem, em que é descrito da seguinte forma:

O agente monitora o diretório para checar se existe algum template disponível, verificando se é necessário ou não repassar alguma tarefa para o NFVO e confirma que um template foi compartilhado, então ele envia uma nova solicitação para o NFVO (1-3). O NFVO solicita ao VIM disponibilidade dos recursos de rede necessários para a interconexão de uma nova VNF e reserva os recursos usando a operação de criar reserva da interface de gerenciamento de recursos virtualizados (7-20). Se necessário, o NFVO solicita que o Network Manager conecte as interfaces externas das VNFs às interfaces da rede física (21-22).

Figura 14 – Instanciação dos Serviços de Rede e das Aplicações na MEC



Fonte: O Autor (2019)

Será iniciado os serviços específicos que foram pré-configurados na imagem para utilizar serviços de rede e aplicações em containers. Os containers conectam-se ao gerente Swarm, mapeando as tarefas que serão executadas e retorna ao *Daemon* a confirmação que todos os serviços e tarefas foram criados e conectados (23-28). Por fim, o NFVO reconhece a conclusão da instanciação dos novos serviços de rede, e finaliza a operação.

4.4 COMUNICAÇÕES INTERVEICULAR E PROPOSTA DE UMA OBU PARA COMUNICAÇÃO E PROCESSAMENTO DE DADOS

Na camada Fog, a mais baixa, refere-se ao plano de dados e consiste nos carros conectados em um ambiente de rede veicular.

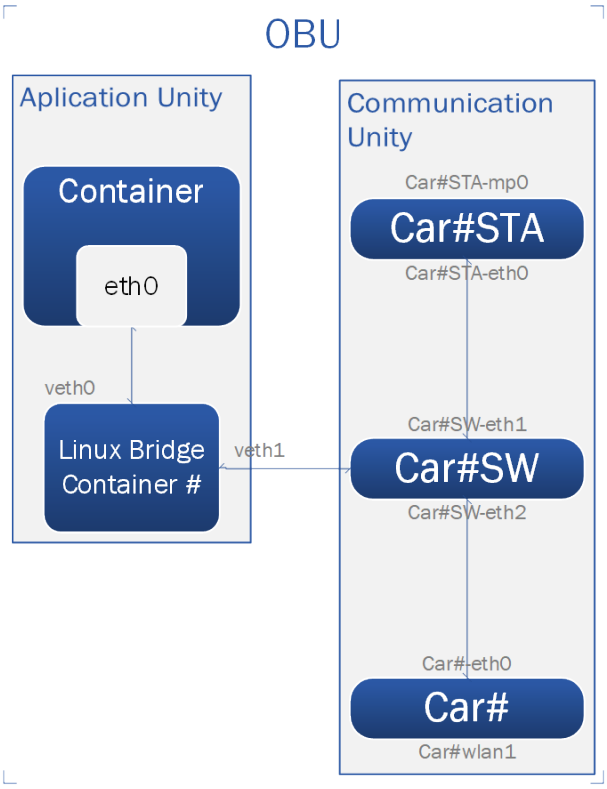
Nesta camada Fog estão os carros da rede VANETs. Foi desenvolvida uma camada de *software* em container para que os veículos pudessem suportar algumas aplicações. Essas aplicações desenvolvidas, foram aplicações que se adequaram a arquitetura existente para comunicação veicular no Mininet-Wifi. Foi implementado uma aplicação cliente para acesso a servidores HTTP e uma interface de comunicação via protocolo P2P, onde os veículos passaram a fazer parte ativa no processo de difusão de informação na rede para outros veículos, aumentando assim o número de nós servidores que continham e compartilhavam conteúdo.

Os carros são habilitados com *OpenFlow* e, dessa forma, podem realizar a comunicação com o controlador SDN da rede veicular. Essa aplicação consulta e administra as tabelas de regras *OpenFlow* alocadas nos veículos para encaminhamento de pacotes. As regras SDN definidas para este cenário são adicionadas utilizando uma aplicação padrão no controlador *Ryu*, onde são identificados os endereços de origem e destino e portas de acesso.

A Figura 15 descreve como o Mininet-Wifi implementa a OBU e nossas contribuições para essa implementação. A parte mais à direita da Figura 15 representa a arquitetura interna do veículo, que já está implementada no Mininet-Wifi, e é baseada em *switches* virtuais (*OpenVSwitch*) e permite que os controladores *SDN* gerenciem as comunicações V2V e V2I. Assim, se um nó de extremidade é um nó na rede mesh sem fio, ou seja, V2V, todo pacote sai da interface de um veículo (Car#), onde o símbolo # indica o id do veículo. O pacote passa pelo *switch* virtual interno (Carro# SW) até chegar ao outro veículo. Na comunicação entre carros e RSU, ou seja, V2I, acontece o mesmo processo, e o *switch* também está ciente dessa comunicação.

Em nossa implementação, adicionamos uma unidade de aplicação representada na parte mais à esquerda da Figura 15, criando uma OBU completa que contém elementos de aplicação e comunicação. Como o *switch* virtual interno, por padrão, é responsável pelo gerenciamento de todas as comunicações do veículo, adicionamos containers conectados ao *switch* de cada veículo que participaria do compartilhamento de arquivos por meio de tunelamento virtual. Assim, todo o tráfego que se origina dos fluxos de aplicativos é

Figura 15 – Proposta de implementação da OBU



Fonte: O Autor (2019)

encaminhado diretamente através do *switch* virtual interno. Neste container, nós executamos o aplicativo P2P *BTSync*. Assim que a conexão é estabelecida, o controlador SDN identifica e controla a comunicação entre as interfaces dos veículos.

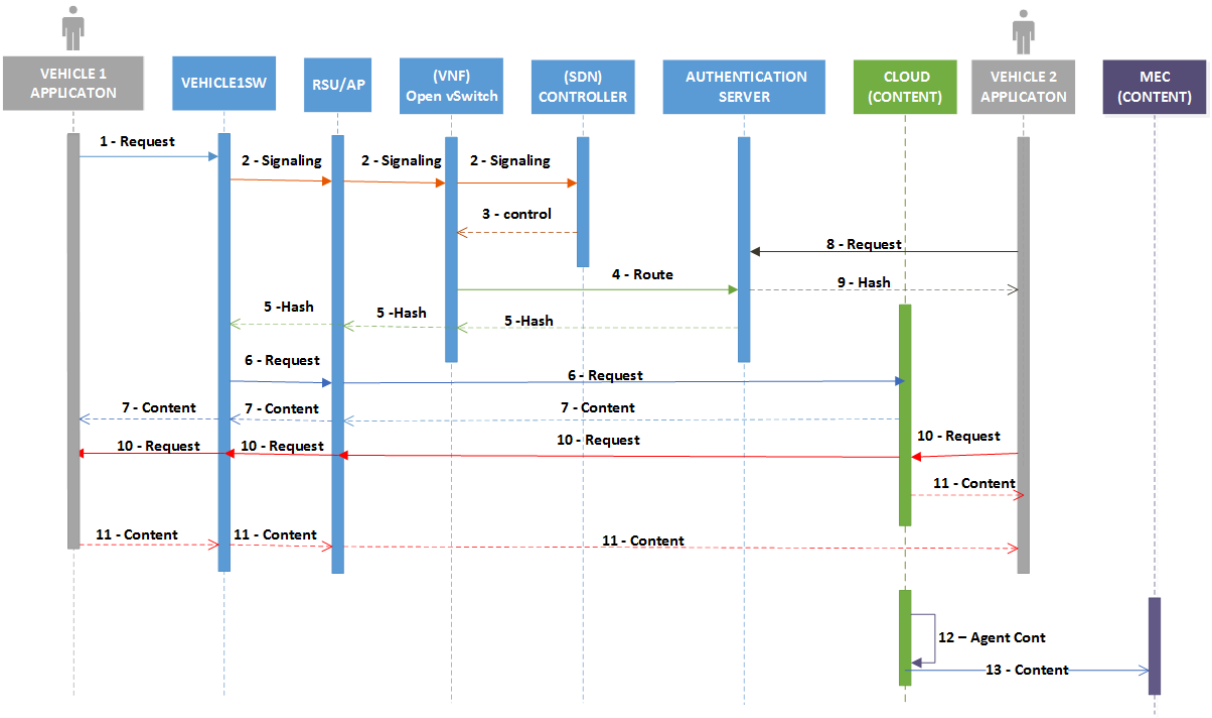
Na Figura 16 é mostrado um cenário mais detalhado do que acontece na comunicação veicular Edge-On integrada com o controlador na nuvem e orquestrada pelo tacker. Implementamos uma aplicação de compartilhamento de conteúdo totalmente distribuído em execução em nossa arquitetura. Ele usa os recursos disponíveis do SDN, NFV e contêineres e funciona de acordo com a Figura 16. Inicialmente, consideramos que não há rotas predefinidas para os dados na rede para que o SDN possa funcionar. O cenário é descrito da seguinte forma:

1. A distribuição de conteúdo começa quando o veículo solicita algum conteúdo para a infraestrutura da nuvem. A aplicação que está executando na OBU se comunica diretamente com o *switch* virtual interno para acessar as comunicações V2V e V2I.

2. Após uma solicitação, o sistema inicia os procedimentos padrão de sinalização SDN, com base nas mensagens *PACKET-IN* entre os dispositivos habilitados para SDN, ou seja, RSU, Switch OpenFlow e controlador SDN.

3. Em seguida, o controlador SDN calcula rotas com base em uma visão global do tráfego da rede e envia as regras para o *switch* interno dos veículos. Assim, ele pode

Figura 16 – Diagrama de sequência Edge-On



Fonte: O Autor (2019)

controlar os fluxos de dados com base na melhor rota para o conteúdo solicitado que inicialmente está sendo executado na nuvem.

4. Inicialmente, o Controlador SDN define as rotas para o Servidor de Autenticação, onde o veículo pode encontrar o hash apontando para o conteúdo de interesse.

5. O controlador encaminha o *hash* até a OBU do veículo.

6. O veículo usa o *hash* para encontrar o conteúdo que em seu primeiro momento encontra-se apenas na infraestrutura de nuvem.

7. O veículo faz o *Download* do conteúdo.

8. Um outro veículo faz uma requisição ao mesmo conteúdo passando por todos os passos anteriores do primeiro veículo.

9. Esse veículo que fez a requisição recebe o mesmo *hash* do servidor de autenticação que está na nuvem.

10. Devido ao *hash* recebido, o veículo 2 faz uma requisição tanto para a nuvem quanto para o veículo 1.

11. Neste momento o segundo veículo passa a receber o conteúdo de ambas as parte que contêm o conteúdo, ou seja, tanto da infraestrutura de nuvem quanto do veículo 1.

12. Um agente é usado na nuvem para colher resultados de tempos em tempos de quais conteúdos estão sendo mais acessados.

13. O conteúdo que estiver sendo mais acessado é enviado para a infraestrutura MEC, fazendo assim com que mais um nó da rede possa compartilhar desse conteúdo.

Usamos *containers* Docker devido à facilidade de promover escalabilidade e portabilidade para as aplicações. O Docker empacota uma aplicação ou ambiente inteiro em um *container* e, a partir de então, todo o ambiente se torna portátil ou replicável para qualquer outro nó da rede que contenha o Docker instalado. Isso reduz drasticamente o tempo de implantação de aplicações ou até de infraestrutura, pois não há necessidade de configurações extras. Ao incluir apenas o aplicativo e suas dependências, os containers tornam-se mais leves e eficientes quando comparados aos recursos disponíveis. Isso é bem útil quando é necessário usar dispositivos que não têm muitos recursos disponíveis, como veículos ou até mesmo alguns dispositivos de IoT. Além disso, fornece ambientes de acordo com as necessidades de recursos estritamente controlados.

O Docker, criam-se *containers*s com base em imagens. Para construir uma aplicação de *container* P2P distribuído e do servidor *web*, usamos uma imagem de um sistema operacional chamado alpine [1] como ponto de partida e a ajustamos para se adequar ao nosso caso de uso. Esta imagem sem nenhuma configuração tem 5 MB, que, quando adaptada à nossa aplicação, fica no total com apenas 30MB de tamanho. Isso apresenta vantagens no tempo de download, no tempo de execução e requer espaço de armazenamento mínimo em dispositivos com pouca capacidade de computação. Para comparação, uma imagem do Ubuntu OS para execução em *container* tem, no mínimo, 84MB e o Centos, 200MB.

4.5 PSEUDOCÓDIGO DE OPERAÇÃO DA PLATAFORMA

O pseudocódigo 1 resume a operação da plataforma. Ele descreve o envio do template da aplicação *web* em execução na nuvem para a MEC, bem como a operação de distribuição de conteúdo P2P entre a nuvem, a MEC e os veículos.

A função *ApplicationWeb* descreve como a nuvem sempre recebe solicitações iniciais e como isso aciona a distribuição de aplicações na infraestrutura MEC. O momento da replicação do ambiente é uma etapa essencial, pois nosso principal objetivo é diminuir o tempo de resposta para que os veículos possam ter acesso ao conteúdo. O pseudocódigo indica que nossa plataforma distribui o template em vez de migrar a máquina virtual completamente. Por isso, transferimos um template, arquivo no formato *YAML*, com um tamanho de poucos *KB*. Esse template já foi usado pelo orquestrador NFV para iniciar rapidamente uma instância executando o aplicativo no MEC, que já continha uma imagem pré-configurada usada para fazer o balanceamento de carga entre os dois ambientes. Assim, demos prioridade ao MEC para receber as requisições.

[1] https://hub.docker.com/_/alpine

Algoritmo 1: Operação da Plataforma

```
template: NFV and container template configuration;
t: set time for migration;
ApplicationWeb(c: cloud, p: index);
while content True in cloud do
    for s in c.S;
    q ← content(s,p);
    Starting;
    Time();
    if Time > t then
        Cloud.send(template);
        MEC ← template ;
        MEC.connect in Cloud;
        ScaleApplication ← p;
        increase the priority of < p,q > in MEC;
        return q
    else
        return ERROR File p not found ;
    end
end
PeerSearch(p: peer, c: ContentName);
while content True in cloud do
    for content in Server;
    q ← super-peer-local-search(content,c);
    if super-peer-local-search-succeded then
        vehicle ← content;
        breack;
    else
        insert content into vehicle;
        merge-super-peer-caches(Cloud,MEC);
        return q
    end
end
Super-peer-local-search(vehicle: super-peer, c: ContentName);
if an entry < c,q > exists in vehicle.F then
    download < c,q > in cloud and vehicle.F;
    return q
else
    return ERROR File c not found
end
merge-super-peer-caches(Cloud:peer, MEC:peer);
for content in Cloud.F do;
if MEC.F contains content then
    increase the priority of content in MEC.F
else
    insert content into MEC.F
end
```

Fonte: O Autor (2019)

As funções *PeerSearch*, *Super-Peer-local-search* e *Merge-super-peer-caches* descrevem como as solicitações de conteúdo inicialmente vão para um servidor executando apenas na infraestrutura de nuvem. Após o primeiro veículo solicitar e recuperar o conteúdo, todos os outros veículos ou dispositivos que solicitarem o conteúdo acessarão diretamente da Cloud, MEC ou veículos usando P2P. Há também um agente que monitora qual conteúdo foi mais procurado pelos veículos. Esse conteúdo também pode ser replicado para a infraestrutura MEC, que surge como mais um local para armazenamento e compartilhamento próximo dos veículos da região.

4.6 CONSIDERAÇÕES FINAIS

Este Capítulo apresentou as funcionalidades dos elementos que compuseram a implementação da proposta dessa dissertação, incluindo *softwares* de gerenciamento de ambientes

de nuvem pública e privada, bem como mostramos de forma detalhada as operações realizadas entre os três níveis da plataforma (*Cloud/MEC/Fog*). Onde foi descrito desde o envio do *template* da aplicação *web* em execução na nuvem para o MEC, bem como a operação de distribuição de conteúdo P2P entre a nuvem, a MEC e os veículos. O Capítulo 5 apresentará os resultados obtidos em testes realizados utilizando nossa plataforma.

5 AVALIAÇÃO EXPERIMENTAL

Neste Capítulo são realizadas as avaliações com o objetivo de validar a plataforma proposta. A Seção 5.1 descreve os recursos e ferramentas utilizados no *testbed* bem como o ambiente desenvolvido para a realização dos experimentos. A Seção 5.2 apresenta parâmetros e métricas utilizados. A Seção 5.3 apresenta os resultados dos experimentos contendo especificações de como foram realizados os testes e a Seção 5.4 apresenta as considerações finais.

5.1 SOFTWARES UTILIZADOS

As plataformas de *softwares* escolhidas como base para a construção da arquitetura Edge-Ful e do cenário das simulações desta dissertação foram o *OpenStack*[1], Docker[2] e Mininet-WiFi[3].

OpenStack é responsável por controlar grandes conjuntos de recursos de computação, armazenamento e rede em todo o *datacenter*, gerenciados por meio de uma interface gráfica ou por meio da API.

Docker fornece uma camada adicional de abstração e automação de virtualização de nível de sistema operacional tanto no *Windows* quanto no Linux. O Docker possibilita o empacotamento de uma aplicação ou ambiente inteiro dentro de um container, e a partir desse momento o ambiente inteiro torna-se portável para qualquer outro Host que contenha o Docker instalado.

BTSync[4] é uma aplicação que usa o protocolo *BitTorrent* para sincronização de dados. Consequentemente, é possível sincronizar pastas e arquivos entre diferentes plataformas com segurança e rapidez. Em nossos experimentos, simulamos a sincronização de dados ou diretórios entre containers, permitindo que vários containers compartilhem o mesmo conteúdo para aumentar a disponibilidade. Este tipo de técnica de sincronização é promissor no que diz respeito às aplicações de redes veiculares e até aplicações de IoT, pois o conteúdo requisitado estará nos próprios dispositivos finais que em determinado momento podem ser vizinhos. Portanto, será possível criar estratégias para a disponibilidade e acesso a dados para uma latência mínima.

Traefik[5] é um proxy reverso HTTP e balanceador de carga arquitetado e desenvolvimento para ser totalmente dinâmico e orientado a microsserviços. Sua configuração é feita de forma automática e dinâmica. Além disso, ele suporta nativamente diversos tipos

[1] https://www.openstack.org/
[2] https://www.docker.com/
[3] https://github.com/intrig-unicamp/mininet-wifi
[4] https://www.resilio.com/
[5] https://traefik.io/

de orquestradores, como é o caso do *Docker Swarm, Kubernetes, Mesos*, apenas *Docker, Consul*, dentre muitos outros.

Mininet-WiFi[6] é um aprimoramento do emulador de rede SDN Mininet, adicionando estações *Wi-Fi* virtualizadas e pontos de acesso baseados nos drivers sem fio padrão do Linux e no driver de simulação sem fio 80211_hwsim. Os criadores da ferramenta adicionaram classes para suportar a adição desses dispositivos sem fio em um cenário de rede Mininet e para emular os atributos de uma estação móvel, como posição e mobilidade em relação aos pontos de acesso.

SUMO[7] é um simulador de tráfego rodoviário de código aberto. Ele permite simular como uma determinada demanda de tráfego, que consiste em veículos individuais, se move através de uma determinada rede rodoviária. A simulação permite abordar um grande conjunto de tópicos de gerenciamento de tráfego. A simulação é puramente microscópico, ou seja, cada veículo é modelado explicitamente, tem uma rota própria e se move individualmente pela rede.

Ryu[8] é uma estrutura de rede definida por *software* baseada em componentes. O *Ryu* fornece componentes de *software* com API bem definidas que facilitam para os desenvolvedores criar novos aplicativos de gerenciamento e controle de rede. Ele suporta vários protocolos para gerenciar dispositivos de rede, como *OpenFlow, Netconf, OF-config*. Sobre o *OpenFlow*, o *Ryu* suporta da versão 1.0 à 1.5.

5.1.1 Especificação dos Equipamentos

O ambiente de testes da plataforma proposta desta dissertação, onde os experimentos foram implementados e realizados, será discutido nesta subseção. Todo o ambiente de testes foi montando em dois laboratórios de pesquisa localizados no Centro de Informática, campus da UFPE (Universidade Federal de Pernambuco).

Para a realização dos experimentos, uma nuvem privada foi configurada com o *OpenStack* em um *Data Center* da própria universidade juntamente com a arquitetura NFV que foi integrada através do orquestrador *Tacker*. Uma infraestrutura MEC também foi configurada com o *OpenStack* juntamente com o *Tacker* em um servidor diretamente conectado a um cenário veícular emulado através do Mininet-WiFi.

O ambiente de nuvem foi configurado em um servidor Linux com a distribuição Ubuntu 16.04, onde foi configurado o ambiente da MEC. Em um *notebook* o cenário VANET foi criado para caracterizar a mobilidade dos carros em conjunto com o (*Simulation of Urban Mobility - SUMO*), responsável por simular o tráfego dos veículos. A Tabela 3 especifica a configuração de *hardware* e *software* utilizada nos testes.

[6] https://github.com/intrig-unicamp/mininet-wifi
[7] http://sumo.sourceforge.net/
[8] https://osrg.github.io/ryu/

Tabela 3 – Configuração de *Hardware* e *Software* do *testbed*

Configuração de Hardware e Software do testbed		
Ambiente de Execução	Configurações de Hardware	Configurações de Software
Nuvem - NFV - SDN - Container	Servidor Processador: Intel(R) Xeon(R), Memória RAM: 16GB	Ubuntu 16.04, OpenStack Ocata, Tacker, Controlador Ryu v4.21, Docker v18.06.1-ce, Traefik, BTSync.
MEC - NFV - Container	Servidor Processador: Intel(R) Core(TM) i7, Memória RAM: 12GB	OpenStack Ocata, Tacker, Docker v18.06.1-ce, Traefik, BTSync.
FOG - Mininet-WiFi - Container	Notebook Dell: Intel(R) Core(TM) i7, Memória RAM: 8GB	Mininet-WiFi, Open vSwitch 2.5.5, Open Flow 1.3, Docker v18.06.1-ce, BTSync.

Fonte: O Autor (2019)

5.1.2 Ambiente de Testes

Conforme descrito na descrição da plataforma, ela é totalmente integrada ao SDN, pois é baseada no *OpenFlow* e no *Open vSwitch*. Cientes disso, criamos um ambiente para avaliar o funcionamento da aplicação em um ambiente com mobilidade, examinando como o processamento e a distribuição de conteúdo podem se comportar quando dependem da nuvem, da MEC e quando o conteúdo for compartilhado entre os próprios veículos.

Cada veículo é emulado em uma VM pois foi a melhor maneira que conseguimos encontrar para isolar o ambiente, para que quando a troca de conteúdo fosse entre os próprios veículos ocorresse da maneira mais real possível. A mobilidade é definida pelo arquivo de configuração do SUMO com as posições dos carros de acordo com o mapa da ilha de Manhattan. Os carros fazem seus trajetos conforme o mapa estabelecido, para cada movimentação do veículo, a posição desse veículo é atualizada no Mininet-WiFi. O tempo de simulação definido para cada execução foi de 180 segundos, período suficiente para a troca de conteúdo entre eles.

Todas as RSUs estão conectadas a um único controlador SDN remoto que está em execução na nuvem, nesse ambiente elas funcionam como (*Access Point - AP*). Com a mobilidade dos carros, o tráfego é gerado no código do cliente "nó", também chamado carX, que envia requisições a um servidor que inicialmente encontra-se na infraestrutura de nuvem. Para avaliar o ganho na diminuição do tempo de resposta e no ganho de tempo da distribuição do conteúdo criamos nossas próprias aplicações. A Tabela 4 apresenta os principais parâmetros utilizados na configuração.

A Figura 17 ilustra o cenário físico produzido no Mininet-WiFi com RSUs conectadas ao controlador que está em execução na nuvem, e todas as RSUs criadas são conectadas

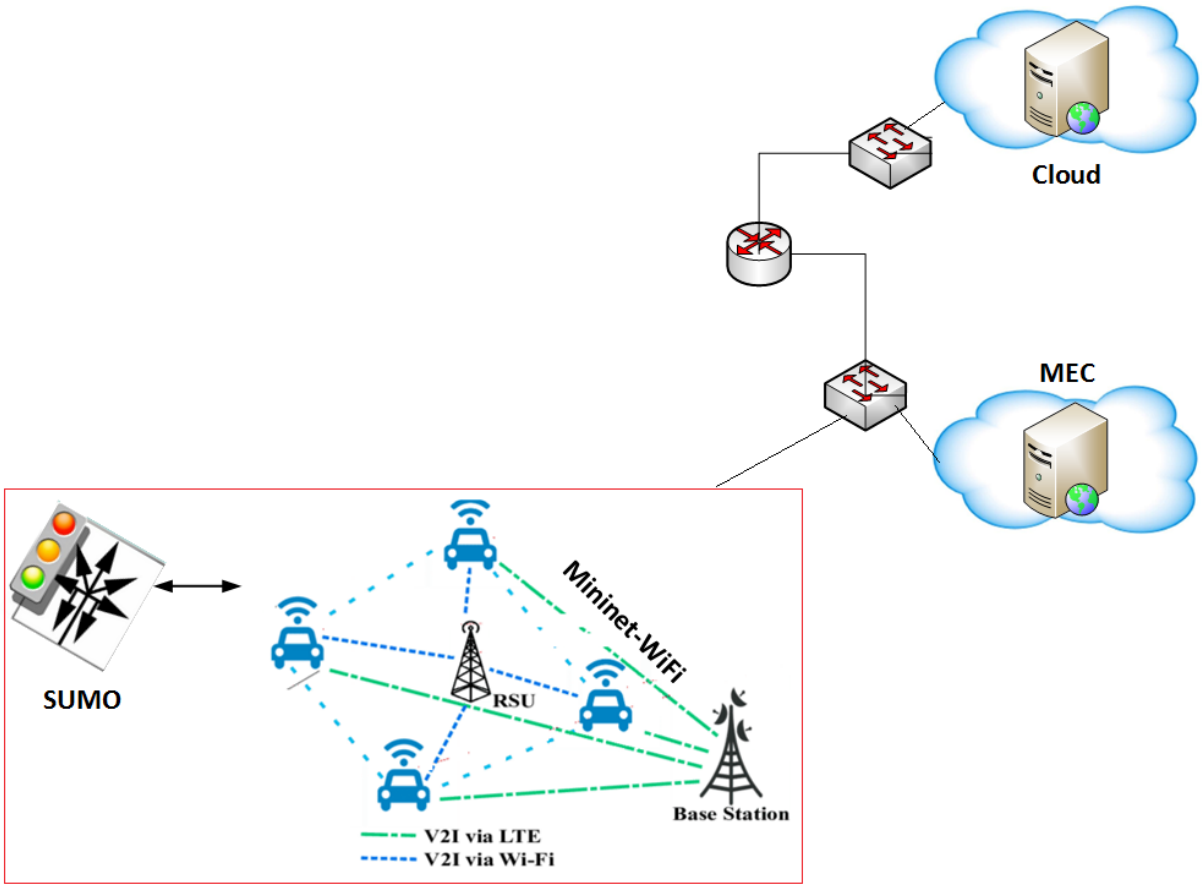
Tabela 4 – Parâmetros de Avaliação

Tempo de Execução	180s
Número de Veículos	10
Velocidade Máxima	11,18 m/s
Tamanho do Arquivo	4KB(HTML) e 64MB(P2P)
Número de RSUs	6

Fonte: O Autor (2019)

a infraestrutura MEC que em um determinado momento irá passar a receber requisições oriundas dos veículos. Para efeitos de diminuição da latência os veículos terão a infraestrutura da MEC para fazer suas requisições, e para efeito de distribuição de cargas em determinado momento as requisições feitas pelos veículos poderão ser atendidas pelas nuvem, MEC e pelos próprios veículos. Tanto as comunicações V2V, V2I e a conexão com o controlador remoto são configuradas no cenário.

Figura 17 – Cenário Físico



Fonte: O Autor (2019)

O Edge-On possui os elementos para o desenvolvimento e a operação de aplicações

com requisito de latência próximo aos 10m/s para aplicações fog no contexto da IoV. Isso pode ser alcançado graças à integração entre *containers* e SDN em uma infraestrutura de suporte a NFV. De acordo com o que foi demonstrado, as regras de encaminhamento das aplicações de qualquer tipo, incluindo aquelas totalmente distribuídas, podem ser gerenciadas a partir de um controlador SDN. Além disso, no caso de indisponibilidade de acesso ao conteúdo na nuvem remota, especialmente nos cenários de conectividade da IoV, as aplicações que suportam P2P podem continuar a funcionar de forma independente. Essa análise serve como uma recomendação para os desenvolvedores prepararem suas aplicações usando componentes P2P ou com suporte de distribuição para IoV.

5.2 MÉTRICAS E PARÂMETROS

Para avaliar o desempenho da aplicação veicular utiliza-se o tempo de resposta à aplicação e o tempo que é necessário para obter o conteúdo durante o período do experimento. Todas as requisições feitas são caracterizadas pelo cenário VANET emulado. Os fatores e parâmetros são definidos na emulação, para alterações na quantidade de carros que contém o conteúdo requisitado para cada tipo de experimento. Estas alterações são necessárias para gerar mais veículos que passarão a ser servidores de conteúdo para outros veículos que fizerem requisições para o mesmo conteúdo. A descrição mais detalhada dessas métricas poderão ser visualizada na Tabela 5.

Tabela 5 – Projeto de experimentos

Características	**Descrição**
Sistema	Ambiente de Computação em Nuvem e MEC: OpenStack Mitaka, Tacker, Controlador Ryu v4.21, Docker v18.06.1-ce, Traefik e BTSync. Emulador Mininet-WiFi: Open vSwitch 2.5.5, Open Flow 1.3, Docker v18.06.1-ce e BTSync
Métricas	Tempo de Resposta e Tempo Compartilhamento de Conteúdo
Parâmetros	Quantidade de Veículos e Tempo de Simulação.
Fatores	Tempo de Simulação e Quantidade de nós compartilhando conteúdo
Projeto de Experimentos	Experimento 1: Avaliação do tempo de resposta para requisições V2V, V2I e de veículo para nuvem. Experimento 2: Avaliação de requisição e balanceamento HTTP para a infraestrutura nuvem e MEC. Experimento 3: Avaliação do uso da infraestrutura completa (nuvem, MEC e fog)

Fonte: O Autor (2019)

5.3 RESULTADOS E DISCURSSÕES

Os experimentos usando as configurações padrões do emulador fazendo requisições à arquitetura de nuvem e os demais testes com os acréscimos da infraestrutura MEC e compartilhamento do conteúdo entre os próprios veículos, serão apresentados e discutidos. Fizemos testes para a execução de aplicações em containers em IoV para avaliar o desempenho da arquitetura proposta em um ambiente emulado composto por OBUs, *switches* habilitados com *OpenFlow* e *switches* virtuais (*Open vSwitch*). O padrão ETSI para ITS (REZNIK et al., 2018), define que a comunicação de V2I, sugere requisitos de menos de 10 ms para a maioria das aplicações. O ETSI também recomenda o MEC para redes veiculares. Assim, esse requisito de latência motiva a execução de experimentos.

5.3.1 Experimentos sem a proposta

Esse primeiro experimento que avaliou o tempo de resposta foi realizado em três cenários. Quando os veículos se comunicam diretamente entre si - V2V, quando retransmitem a comunicação por meio de um RSU-V2I e quando se comunicam com a nuvem. O Mininet-Wifi suporta os dois primeiros tipos de avaliação. Para o terceiro tipo, integramos a ferramenta de simulação à infraestrutura real da nuvem, adicionando uma interface de rede no servidor que executa o Mininet-Wifi e criando conexões com o *switch* virtual que é integrado ao veículo. Então, esse veículo acessaria a Internet através dessa nova porta que foi adicionada.

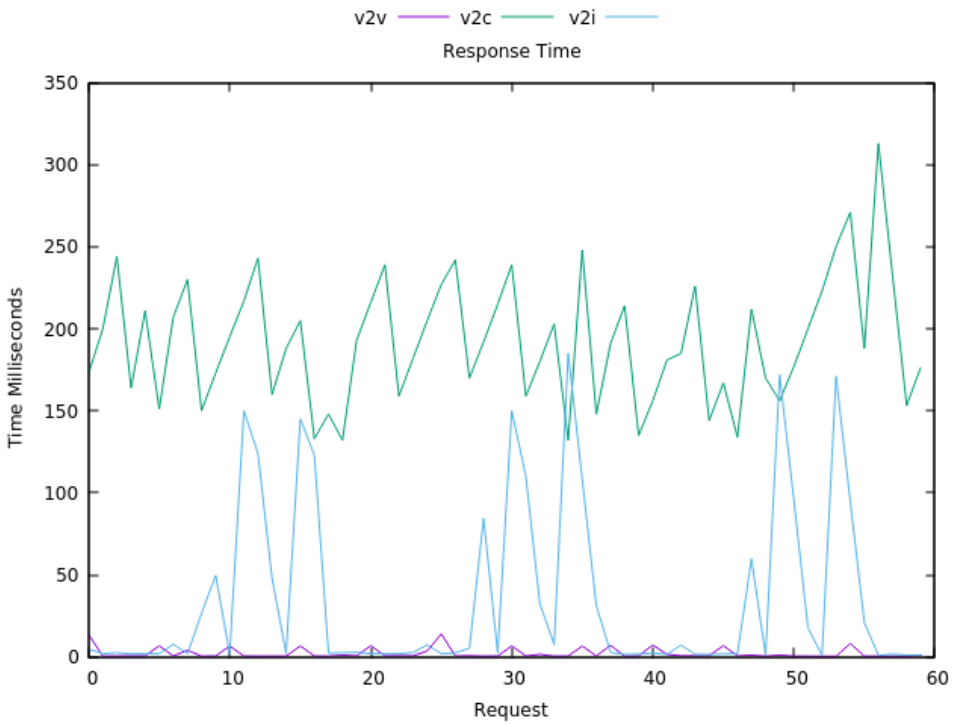
5.3.1.1 Tempo de Resposta

Inicialmente, empregamos o ICMP para testar o tempo de resposta entre os veículos. Em seguida, utilizamos o modelo V2I onde os veículos se comunicam através das RSUs e por fim realizamos testes no modelo tradicional cliente-servidor através do (*Vehicle-to-Cloud - V2C*).

A Figura 18 apresenta os resultados preliminares dos três cenários, sendo possível verificar a latência entre apenas dois *endpoints*, onde um deles pode ser um veículo de acesso direcionado, um veículo acessado através de uma RSU, ou com a infraestrutura de nuvem. Para esse teste inicial não estamos avaliando a arquitetura Edge-On, mas sim os atrasos individuais em um experimento que testa o acesso às três camadas que serão integradas na arquitetura Edge-On. Observamos os comportamentos discutidos na literatura: variações significativas na latência ao acessar a nuvem e ao usar a infraestrutura, bem como a redução de latência no ambiente V2V, demonstrando a necessidade de investigar minuciosamente os mecanismos e protocolos para alavancar a comunicação direta entre veículos em várias condições de rede.

Os resultados dos testes entre veículos, isto é, V2V, apresentam picos de 10 ms, o que é adequado para aplicações reais de acordo com os requisitos estabelecidos pelo ETSI

Figura 18 – Resultados Preliminares



Fonte: O Autor (2019)

(REZNIK et al., 2018). A latência para conectividade entre veículos através de RSU exibe variações significativas. Há resultados muito baixos em determinados momentos, que permanecem abaixo dos limites estabelecidos pelo ETSI. No entanto, há situações em que o tempo de acesso sobe um pouco mais, muito devido a troca de conexão de uma RSU para outra. A latência para se conectar à nuvem tem em média 200ms, mas também tem uma variação alta, até 100ms. Nos dois últimos casos, V2I e *Vehicle-to-Cloud* (V2C), a variação resulta da mobilidade dos veículos e das características da criação de regras pelo controlador.

5.3.2 Plataforma Edge-On

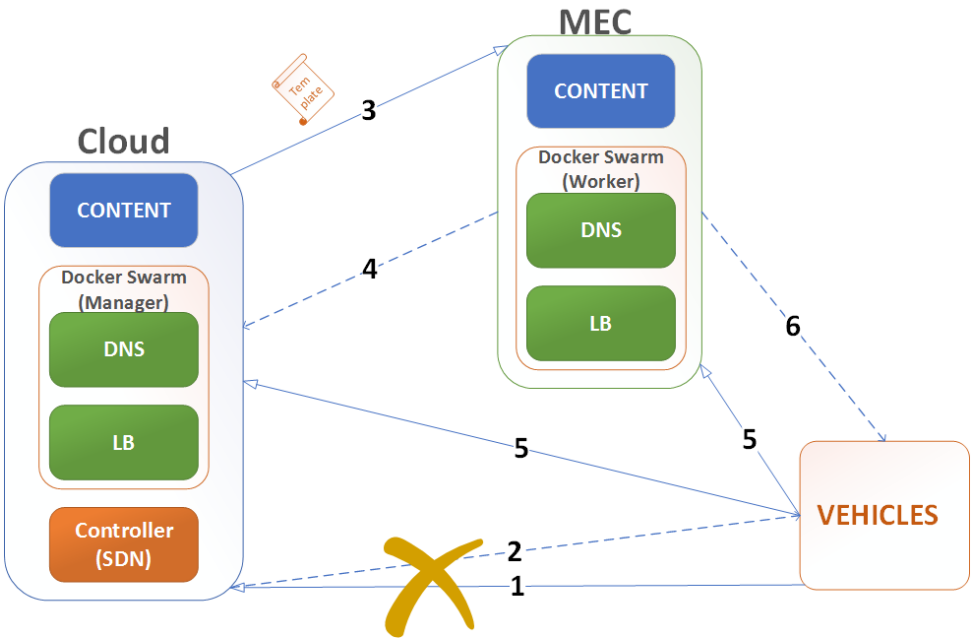
Para esta subseção serão discutidos alguns cenários e resultados obtidos ao realizar um estudo de caso da arquitetura Edge-On que tem como objetivo diminuir a latência da rede buscando formas de distribuir o conteúdo na borda próximo aos dispositivos finais.

5.3.2.1 Sinergia entre nuvem e MEC

Tendo como ponto de partida os resultados obtidos no primeiro experimento onde podemos observar que o tempo de comunicação até a nuvem acabou sendo um pouco maior, o que pode ter um impacto negativo em um cenário IoV, nossa proposta é adicionar uma infraestrutura MEC para que possamos distribuir as aplicações.

Como podemos ver na Figura 19, no primeiro momento as requisições irão ser feitas a uma aplicação somente na infraestrutura de nuvem, e logo depois seguindo nossa proposta essa mesma aplicação irá ser distribuída entre a infraestrutura de nuvem e MEC para que possa ser acessíveis aos veículos. O momento de replicação do ambiente é um passo importante pois como nosso principal objetivo é diminuir o tempo de resposta para que os veículos possam ter acesso ao conteúdo, não nos preocupamos em carga de trabalho ou sobrecarregar a aplicação, mas sim na melhor forma que poderíamos replicar os serviços que estariam sendo executados na nuvem para a infraestrutura MEC e entre os próprios veículos. Assim, quando replicamos esse ambiente que está inicialmente sendo executado na nuvem, poderíamos migrar a instância por completa, contudo, isso iria demandar muito tempo e ficaríamos dependentes de uma boa conexão de rede. Sendo assim a estratégia criada por nós foi compartilhar apenas o *template* que foi utilizado para criar as instâncias e as aplicações. Assim transferimos um *template*, arquivo no formato *yaml* medido em *KB* que foi utilizado pelo orquestrador NFV que também está integrado a arquitetura MEC para iniciar rapidamente uma instância executando a aplicação na MEC que será utilizada para fazer um balanceamento de carga entre os dois ambientes.

Figura 19 – Cenário de testes entre Nuvem e MEC



Fonte: O Autor (2019)

Embora algumas aplicações possam funcionar muito bem no contexto monolítico que é executado em um ou apenas alguns servidores já conhecidos na rede, nossa arquitetura irá ser integrada completamente em uma arquitetura de aplicação distribuída. Em primeiro lugar, neste cenário, temos muitos componentes e controlá-los manualmente torna-se algo complexo. Além disso, o veículo normalmente nunca saberá em qual nó do cluster os componentes são executados. Sua localização pode não ser estável, na verdade, na grande

maioria das vezes não será, já que o componente que contém a aplicação irá ser movido da infraestrutura de nuvem para a MEC ou para qualquer outra infraestrutura, o que poderia vir a ocorrer devido a vários motivos externos à aplicação. Para fins de testes, apenas assumimos como *default* essa operação de migração. Trabalhos futuros, consideraram limiares e a inteligência para a execução da arquitetura Edge-On de maneira autonômica. Assim, precisamos de uma maneira pela qual o veículo possa localizar a aplicação independente do local onde ela esteja hospedada. O que é mais comumente usado é uma autoridade externa que está ciente da topologia do sistema a qualquer momento. Esta autoridade ou serviço externo conhece todos os nós e seus endereços IPs que pertencem atualmente ao cluster; Essa aplicação conhece todos os serviços que estão em execução e onde estão sendo executados. Esse serviço é chamado de serviço DNS, para fins de teste foi utilizado o DNS do Docker.

Implantamos a aplicação juntamente com os serviços em forma de microsserviços com a ajuda do orquestrador Swarm. Posteriormente para que os veículos acessem esses serviços, foi necessário utilizar um *proxy* reverso. Os *proxies* reversos tradicionais exigem que configuremos cada rota que conectará caminhos e subdomínios a cada microsserviço. Em um ambiente onde podemos adicionar, remover, desligar, atualizar ou dimensionar nossos serviços várias vezes ao dia, a tarefa de manter as rotas manualmente atualizadas torna-se bastante improdutiva ou até mesmo inviáveis em um ambiente tão complexo como o de IoV.

Um dos requisitos para a viabilidade de nossa arquitetura desenvolvida é a capacidade de adaptação dinâmica da mesma. Este requisito é inimigo de ferramentas que necessitam de arquivos com configurações estáticas. Esta necessidade foi o principal motivador da escolha do *Proxy Traefik*, que suporta reconfiguração dinâmica, além de trazer consigo todas as outras características de *proxy* reverso como balanceamento de carga, *cache* e etc. O *Traefik* escuta a API do orquestrador e gera instantaneamente as rotas para que os microsserviços sejam conectados ao mundo externo sem mais intervenções de nossa parte para que fiquem acessíveis aos veículos que estejam fazendo requisições para a aplicação.

5.3.2.1.1 Tempo de Resposta: Nuvem vs MEC

Para esse segundo experimento, a intenção é avaliar e apresentar insumos que comprovem a melhoria da latência da rede fazendo uso da infraestrutura MEC. Esse experimento avaliou o tempo de resposta quando os veículos se comunicam diretamente com infraetruturas externas ao emulador Mininet-Wifi, esse cenário pode ser visualizado na figura 17. Em ambos os casos o Mininet-Wifi por padrão não dá suporte a esses experimentos sendo necessário adicionar uma interface de rede no servidor que executa o emulador, criando conexões com o *switch* virtual que é integrado ao veículo. Então, esse veículo acessaria a internet através dessa nova porta que foi adicionada e consequentemente se conectaria as infraestruturas externas.

Para esse teste, usamos o ICMP para comparar o tempo de resposta do veículo para a infraestrutura de nuvem e para a MEC. A Tabela 6 mostra os resultados dos dois experimentos. É possível verificar a latência entre os dois pontos finais: Veículo-MEC e Veículo-Nuvem.

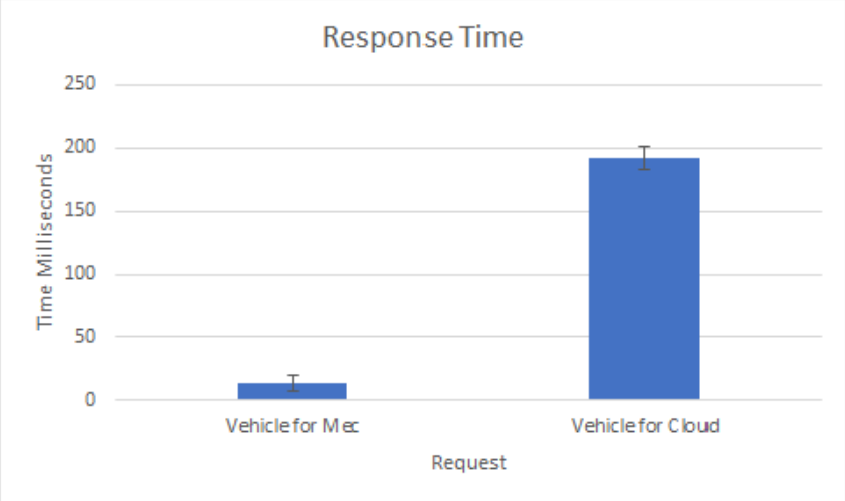
Tabela 6 – Estatística descritiva em (ms) para o tempo de resposta entre Nuvem vs MEC

Cenário	Média	Desvio Padrão	Margem de Erro	Intervalo de Confiança
V2MEC	13,92	23,76	6,01	$7{,}91 \leq 13{,}92 \geq 19{,}94$
V2Cloud	192,15	35,65	9,02	$183{,}12 \leq 192{,}15 \geq 201{,}17$

Fonte: O Autor (2019)

Por sua vez, a Figura 20 demonstra o comportamento temporal discutido na literatura: variações significativas de latência ao acessar a infraestrutura em nuvem, assim como redução de latência e redução da variação de latência no ambiente MEC. Monstrando a necessidade de investigar minuciosamente os mecanismos e estruturas que alavancam a comunicação com infraestrutura na borda da rede.

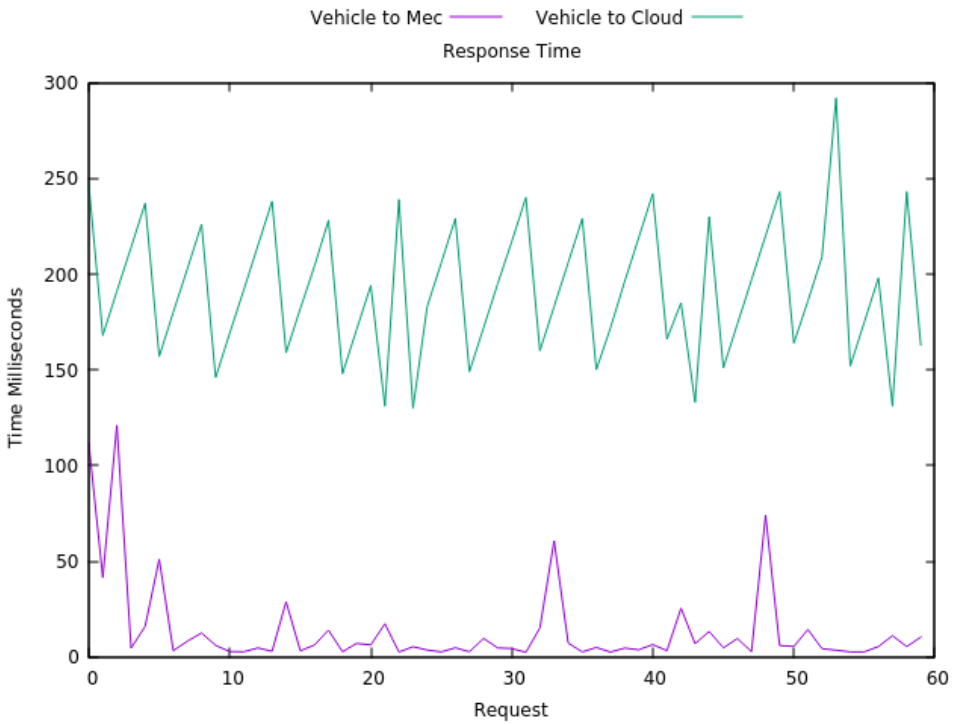
Figura 20 – Gráfico do tempo de resposta entre Nuvem vs MEC



Fonte: O Autor (2019)

As Figuras 20 e 21 e a Tabela 6 demonstram que o V2Mec possui picos de 13ms, o que é uma latência muito baixa para aplicações reais, estando muito próximo dos limites estabelecidos pelo ETSI. A maioria desses picos ficou abaixo dos 10ms estabelecidos pelo ETSI, porém, como podemos ver na Figura 21, no início do estabelecimento da comunicação há picos muito altos a partir de 100 ms, esse tempo maior imagina-se que é devido não ter nenhuma regra criada do controlador ao veículo, então somente depois normalizados, onde esses valores iniciais impactaram diretamente na média final. A latência de acesso à nuvem tem uma média maior que a do MEC, em torno de 200ms, mas também tem uma variação alta, de até 100ms.

Figura 21 – Tempo de Resposta entre Nuvem vs MEC



Fonte: O Autor (2019)

Como esperado, o atraso até a infraestrutura de nuvem foi o mesmo. Entretanto, a requisição até a infraestrutura MEC diminuiu bastante em comparação com o primeiro experimento feito, que pode ser visualizado na Figura 18, quando nesse cenário a requisição passava pela infraestrutura (RSU) e iria até um veículo e retornava com a resposta, os picos da imagem do primeiro cenário acontecem muito devido ambos os veículos estarem em movimento, com velocidades distintas, mudanças de trajetórias e mudanças de RSUs. E nesse cenário com a proposta, a infraestrutura estará fixa próxima à RSU. Percebe-se com a junção desses dois testes que mesmo a requisição feita de um veículo para outro, onde essa requisição não seja feita diretamente, mas necessite de conexão com a RSU pode ser mais lenta do que a comunicação ou busca de conteúdo feita por um veículo em uma infraestrutura MEC na borda da rede.

5.3.2.1.2 Requisição a uma aplicação web: Nuvem vs MEC

Para este experimento, a intenção é avaliar a proposta fazendo uso de todos os serviços descritos nesta subseção, tais como repasse do *template* NFV de uma arquitetura a outra, a utilização do resolvedor de nomes (*Domain Name Server - DNS*), do balanceamento de carga, e de todas as funcionalidades que um *proxy* reverso nos proporciona. Além de que uma vez o *template* sendo compartilhado, todo o processo passa a ser dinâmico, a partir da criação da instância até a criação e configuração de toda aplicação, pois quando o *template* chega até a MEC é utilizada uma imagem pré-configurada à nível de recursos

com o Docker instalado e com todas as configurações necessárias para que essa instância se comunique com a nuvem e assim o *cluster swarm* fique disponível. É importante ressaltar que não foi medido o tempo desde o envio do *template* até a criação de uma nova instância juntamente com a aplicação em execução, pois nosso trabalho focou em avaliar se uma vez o ambiente já replicado entre nuvem e MEC haveria ganho no tempo das respostas das requisições feitas pelo veículo.

Neste cenário, o veículo executa o comando *curl*[9], esse comando é usado em linhas de comando ou *scripts* para transferências de dados, então, o utilizamos para solicitar uma página *web* inicialmente somente para a nuvem e testar o tempo de resposta e posteriormente redirecionamos essas requisições para o *proxy* que ficou responsável por gerir a quem seria designado a função de resolver essa requisição, uma vez que o ambiente estava replicado entre a MEC e a nuvem. Os resultados aparecem na Tabela 7, e também nas Figuras 22 e 23. Os resultados dos testes entre veículos e infraestrutura MEC, ou seja, V2Mec, exibem picos de 47ms, que é uma resposta relativamente baixa para uma aplicação real, e que é importante ressaltar que estamos trabalhando com mobilidade dos veículos, e que diferentemente de um comando ICMP estamos fazendo requisição e tendo como resposta a uma página HTML, por isso esse tempo maior de resposta, tempo esse que ficou próximo aos limites estabelecidos pela ETSI.

Tabela 7 – Estatística descritiva em (ms) para o tempo de resposta HTTP: Nuvem vs MEC

Cenário	Média	Desvio Padrão	Margem de Erro	Intervalo de Confiança
V2MEC	47,28	22,97	5,81	$41{,}47 \leq 47{,}28 \geq 53{,}09$
V2Cloud	404,56	40,34	10,20	$394{,}35 \leq 404{,}56 \geq 414{,}77$

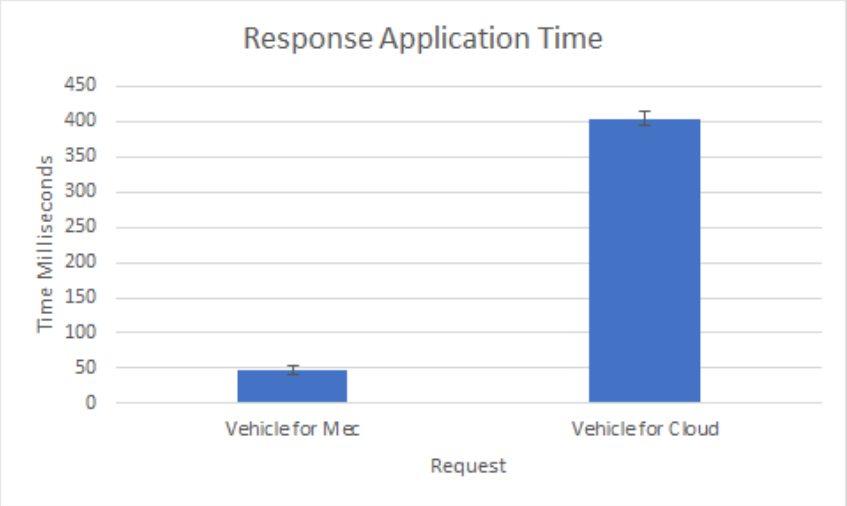
Fonte: O Autor (2019)

Este aumento do tempo de resposta em nossos experimentos estão diretamente relacionados às configurações do cenário, como a criação de tunelamento (*Generic Routing Encapsulation - GRE*) dentro dos veículos e a criação de regras *IPTABLES* e redirecionamento de porta via (*Network Address Translation - NAT*) da infraestrutura de nuvem e da MEC para as aplicações que estão sendo executadas dentro dos containers. Como principal limitação em nossos experimentos, não podemos usar IP público para as máquinas virtuais, exigindo, portanto, alguns redirecionamentos de porta, onde esses redirecionamentos podem ter impactado de alguma forma, já que estamos trabalhando com valores de tempo bem pequenos.

A latência para alcançar a nuvem apresenta uma média maior do que a MEC, em torno de 400ms, e também tem uma alta variação, de até 150ms. Como podemos ver na Figura 23, quando o veículo estava solicitando a página ao *proxy* reverso *traefik*, o balanceador

[9] https://curl.haxx.se/

Figura 22 – Gráfico do tempo de resposta entre Nuvem vs MEC



Fonte: O Autor (2019)

nunca redirecionou a solicitação para a nuvem. Isso ocorre porque o nó do *cluster* que passou a executar na MEC é um nó *worker*/trabalhador o que por padrão fez com que todas as requisições feitas fossem redirecionada para ele. Como os veículos não geraram muitas requisições, essas requisições não precisaram ser balanceadas com uma camada acima da infraestrutura que era onde a mesma aplicação estava em execução em um nó *master* do *cluster swarm*.

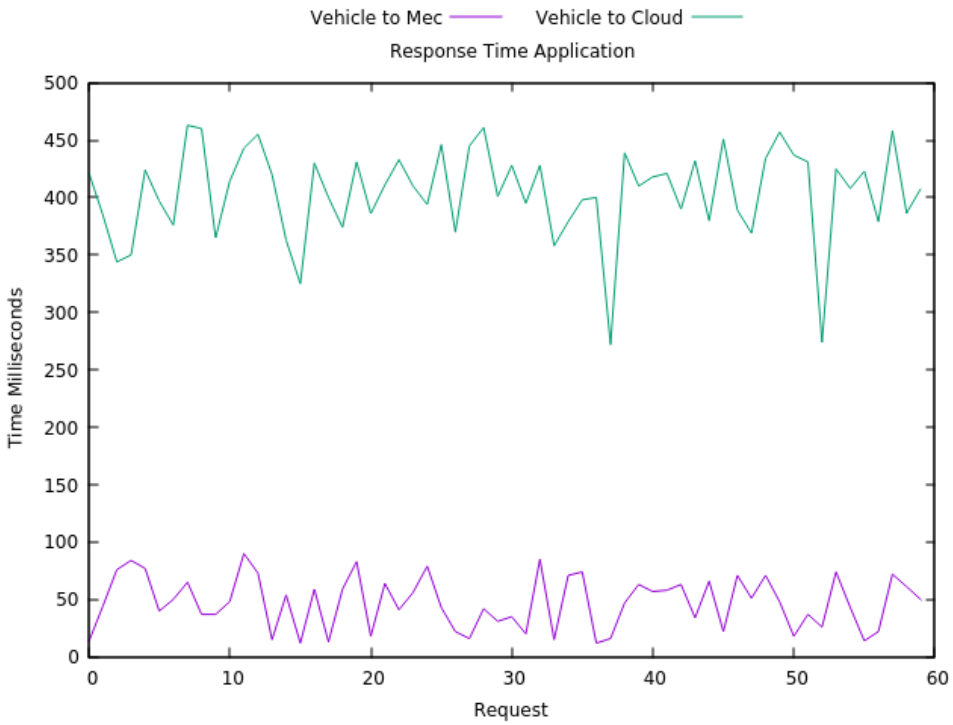
Esse experimento pôde nos mostrar como distribuirmos um *cluster swarm* em uma arquitetura que busca trazer o conteúdo ou processamento para a borda da rede, essa boa prática é deixar os nós gerentes do *cluster swarm* na camada superior, em nosso caso a nuvem e os nós *workers* em uma camada abaixo, em nosso caso na MEC assim todas as requisições terão prioridade em serem respondidas pela MEC, contribuindo assim por uma resposta mais rápida, e caso ocorra uma sobrecarga de requisições, as mesma passarão a ser balanceadas automaticamente com o nó *manager* que está em execução na infraestrutura de nuvem.

5.3.2.2 Transferência de Arquivo

Tendo ainda como ponto de partida os resultados obtidos no primeiro experimento, onde podem ser melhores visualizados na Figura 18, podemos observar que o tempo de comunicação até a nuvem acabou sendo um pouco maior, o que pode ter um grande impacto negativo em um cenário IoV. Nosso objetivo nessa seção é avaliar um ambiente onde além de adicionar uma infraestrutura MEC para fazer parte do processamento, os próprios veículos pudessem distribuir o conteúdo entre eles próprios.

Como podemos ver na Figura 24, no primeiro momento as buscas pelo conteúdo serão feitas a um servidor que está executando somente na infraestrutura de nuvem. Desenvolvemos um agente que monitora quais os arquivos eram mais buscados pelos veículos

Figura 23 – Tempo de requisição de uma página Web: Nuvem vs MEC



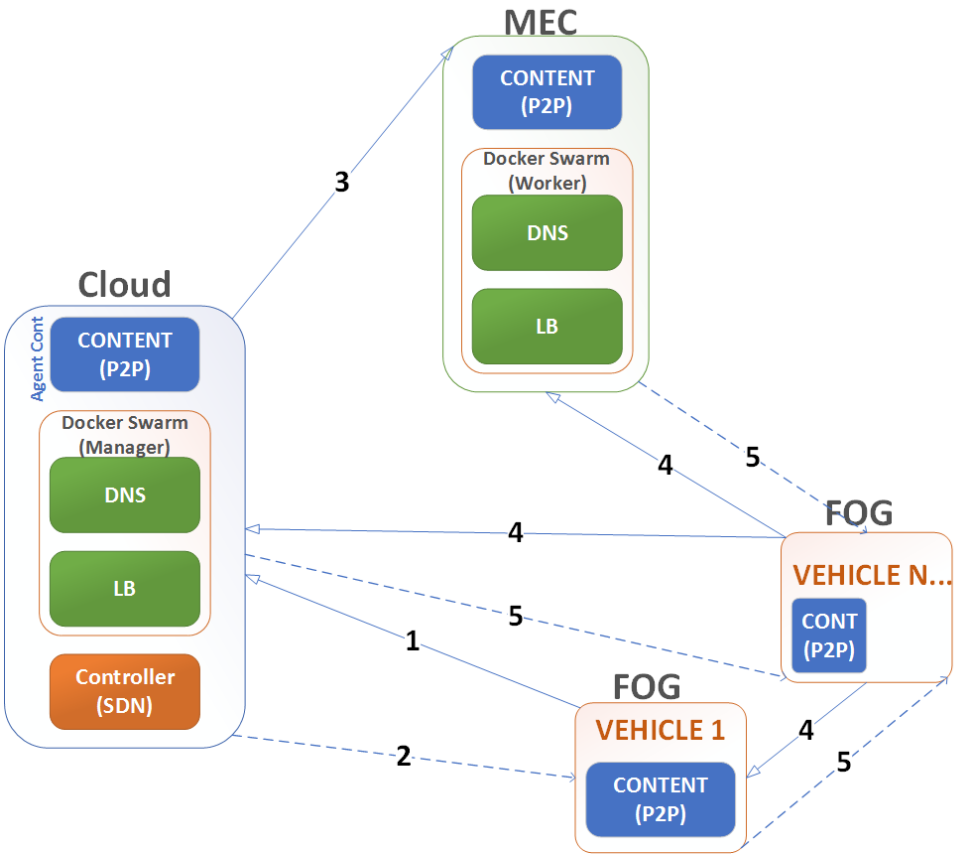
Fonte: O Autor (2019)

e a partir de um número X, esse arquivo poderia ser replicado também para a infraestrutura MEC. Contudo, este modelo do agente ser disparado para que o conteúdo fosse transferido para a MEC não foi utilizado na avaliação pois fizemos uma variação relativamente pequena de veículos fazendo requisições ao mesmo conteúdo, e para que nosso agente fosse disparado seria necessário de um número maior de requisições em busca do mesmo conteúdo, pois se cada arquivo que fosse requisitado fosse enviado a MEC poderia ocorrer uma sobrecarga na infraestrutura sem necessidade, pois o arquivo poderia não ser requisitado mais nenhuma vez por outros veículos. Explicado esse passo, posteriormente todos os veículos que fizerem a busca pelo mesmo conteúdo que o veículo 1, esse conteúdo estará acessível tanto na nuvem quanto nos veículos que já têm aquele mesmo conteúdo.

Para a transferência de arquivos, criamos quatro diferentes cenários variando o número de carros que participavam do compartilhamento do conteúdo usando uma aplicação P2P. O objetivo é usar a comunicação entre os veículos para comparar o modelo tradicional cliente-servidor com o modelo P2P proposto baseado em containers. A mobilidade dos veículos é definida pelo SUMO (BEHRISCH et al., 2011). Neste experimento, os carros também executam suas rotas de acordo com o mapa estabelecido, com base em dados da ilha de manhattan da cidade de *New York* e suas posições são atualizadas no emulador Mininet-WiFi. A simulação está definida para durar 180 segundos, que foi o período máximo necessário para o veículo baixar o arquivo solicitado.

Todas as RSUs, que operam como AP, são conectadas à um único controlador SDN remoto. O tráfego de dados é originado no cliente, também chamado de Car#, que primeiro

Figura 24 – Cenário de teste de Tranferência de arquivo



Fonte: O Autor (2019)

solicita o arquivo para um servidor na nuvem. Em um segundo momento, o arquivo será compartilhado diretamente entre um conjunto determinado de veículos no experimento. A tabela 4 mostra os principais parâmetros usados na configuração dos experimentos de teste.

O experimento foi repetido 60 vezes com diferentes sementes aleatórias para garantir a confiabilidade estatística dos dados. A Tabela 8 exibe as estatísticas descritivas para as experiências de *download* dos arquivos. A latência média quando o *download* do arquivo é feito de veículos para a nuvem é quase duas vezes maior do que quando eles fazem o compartilhamento do conteúdo usando a aplicação P2P. Neste momento os veículos irão ter dois pontos distintos para fazerem o *download* do arquivo em paralelo, a infraestrutura de nuvem e os próprios veículos que já possuam este arquivo.

O desvio padrão mostra a dispersão dos dados em torno da média e, nesse caso, indica um possível *jitter* na transferência de dados ou instabilidade na transferência de arquivos ao longo do tempo gerada pela mobilidade. Novamente, o compartilhamento de conteúdo distribuído seria mais vantajoso porque quanto mais veículos participarem do cenário de compartilhamento de conteúdo, menos jitter ou variações existirão, uma vez que poderão solicitar o arquivo, ou partes do arquivo, ao veículo mais próximo.

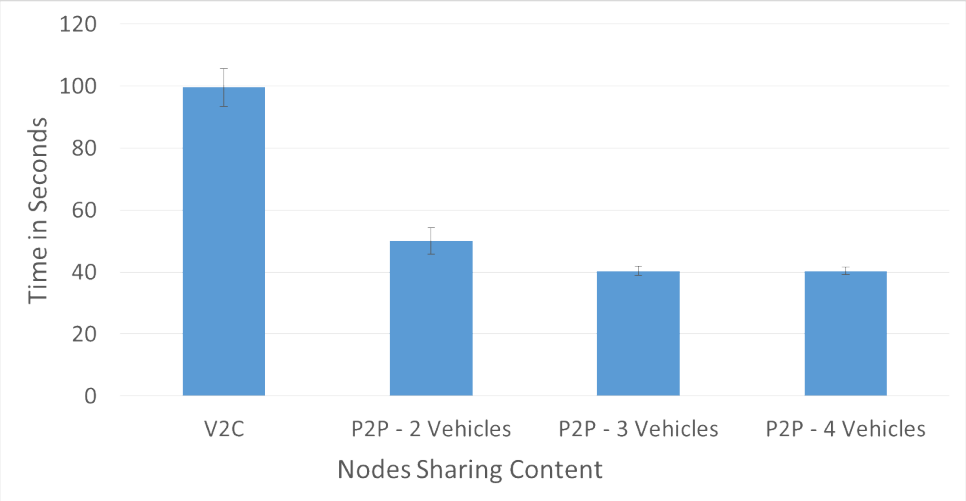
A Figura 25 e a Tabela 8 exibem a duração do download. Podemos observar que a du-

Tabela 8 – Estatística descritiva em (s) para o tempo de resposta de compartilhamento do conteúdo

Cenário	Média	Desvio Padrão	Margem de Erro	Intervalo de Confiança
V2C	99,63	24,29	6,09	$93{,}54 \leq 99{,}63 \geq 105{,}72$
P2P - 2 Veículos	50,05	17,29	4,37	$45{,}67 \leq 50{,}05 \geq 54{,}42$
P2P - 3 Veículos	40,38	5,57	1,41	$38{,}97 \leq 40{,}38 \geq 41{,}79$
P2P - 4 Veículos	40,3	4,84	1,22	$39{,}07 \leq 40{,}3 \geq 41{,}52$

Fonte: O Autor (2019)

Figura 25 – Gráfico de tranferência de arquivo



Fonte: O Autor (2019)

ração do download tende a diminuir no modelo P2P comparado ao download diretamente da nuvem. Também é possível observar que o tempo diminui à medida que aumenta o número de veículos participantes na rede. A duração do download, diretamente da nuvem, é de cerca de 100s. Quando avaliamos os mecanismos de P2P para dois veículos, há uma redução média de até 50% na duração do download. Os resultados também mostram uma tendência de diminuição da latência de acordo com o número de veículos participantes na distribuição P2P. Quando três veículos participam da rede P2P, há uma redução média de 20% no tempo de download comparado à rede de 2 veículos, e 60%, em média, quando comparado à nuvem. Finalmente, na rede P2P com quatro veículos, medimos reduções médias semelhantes às medidas na rede com três veículos, mas detectamos uma redução de 14% na variação, o que também pode indicar uma tendência de queda.

Ao realizar esses testes com esta aplicação de compartilhamento de arquivos entre os próprios veículos na comunicação V2V, comparado ao cenário em que um veículo realiza requisições para uma infraestrutua de nuvem através da comunicação V2I, percebe-se que, de acordo com os resultados apresentados, é correto afirmar que existe um impacto em aplicações veiculares. Ou seja, quando a requisição e o compartilhamento ocorrem

na borda da rede, isso pode impactar na velocidade do fornecimento de um determinado conteúdo para o usuário final. Assim, a arquitetura aqui proposta pode ser usada para beneficiar aplicações sensíveis a atrasos em suas comunicações.

5.4 CONSIDERAÇÕES FINAIS

Este Capítulo apresentou os elementos que compuseram a avaliação da proposta dessa dissertação, incluindo o ambiente de testes, equipamentos de *hardware* e *software* utilizados, as aplicações que foram desenvolvidas em microsserviços com os limiares e métricas definidos para as avaliações e por fim, o resultados dos experimentos. Os resultados demonstraram ganho em nível de tempo de resposta ao usar nossa proposta.

6 CONCLUSÃO

Este capítulo resume os principais pontos discutidos neste trabalho. As considerações finais desta dissertação são apresentadas na Seção 6.1. Na Seção 6.2, são descritas as contribuições e, em seguida, na Seção 6.3, serão discutidos alguns pontos que podem ser explorados em trabalhos futuros.

6.1 CONSIDERAÇÕES FINAIS

Este trabalho apresentou uma plataforma para o gerenciamento de aplicações em redes veiculares definidas por *software*, usando orquestradores NFV e orquestradores de *containers*, com o intuito de gerenciamento flexível e dinâmico na borda da rede e com suporte a um balanceamento de carga entre os dispositivos que participam das requisições. Dessa forma, é possível evitar tanto atrasos quanto gargalos na rede decorrentes de um cenário tradicional no qual as requisições são feitas apenas à nuvem remota. Discutiu-se a necessidade de novas tecnologias para as redes veiculares devido à alta complexidade que o ambiente IoV proporciona. Foram apresentados conceitos como: redes definidas por *software*, virtualização de funções de rede, orquestração, tecnologia de *containers*, gerenciamento em VANETs e controladores SDN. Foram apresentadas boas práticas de replicação de serviço entre nuvem e MEC através da utilização de templates e imagens pré-configuradas, bem como o bom posicionamento de gerentes de *clusters swarm* uma camada acima dos nós *workers*. Foram discutidos orquestradores NFV, o tacker que foi utilizado para implementar toda a arquitetura NFV e orquestrador de *containers* ao qual escolhido foi o Docker *Swarm* para o gerenciamento de nossas aplicações. Em seguida, foram apresentadas as redes veiculares, o procedimento de utilização de *containers* acoplados ao *switch* interno dos veículos que, por sua vez, viabilizou o desenvolvimento de uma aplicação P2P. Por fim, a arquitetura do tacker e todo o processo de sua implementação no ambiente *OpenStack*.

O Edge-On mostra que a sinergia entre SDN, NFV e *containers* provê resultados promissores em relação aos requisitos de aplicações para IoV. Em experimentos utilizando veículos individuais ou um pequeno grupo deles, onde os mesmos podem apresentar condições sem muito tráfego na rede, a latência para a conectividade permaneceram abaixo de 10ms para cenários em que os veículos não dependem da infraestrutura de nuvem para obter seus dados, ou seja, participam diretamente do processamento e da transferência. A implementação e as avaliações também indicam que o SDN pode efetivamente controlar aplicativos P2P, mesmo sob mobilidade. Testes futuros nos permitirão avaliar o comportamento da arquitetura em relação à resiliência, no caso de desconexões, rápida mudança topológica da rede e alocação dinâmica de aplicações distribuídas entre diferentes infra-

estruturas.

6.2 CONTRIBUIÇÕES

Esta seção apresenta as principais contribuições obtidas na proposta dessa dissertação.

- **Criação de uma plataforma em um ambiente real de tráfego de dados e requisições a aplicações reais.**
- **Integração entre os ambientes e sinergia entre as tecnologias utilizadas;**

  Integração entre os ambientes de nuvem, MEC e NFV com *containers* para aplicação P2P em um ambiente IoV onde o SDN pôde controlar o tráfego de todos os dados.
- **Implementação de um modelo de OBU;**

  Foi implementado um modelo de OBU que nos possibilitou desenvolver uma aplicação real associada ao veículo e que servirá para a criação de futuras aplicações para ambientes distribuídos e dispositivos móveis.
- **Proposta de uma estratégia para migração e replicação entre diferentes infraestruturas;**

  Foi proposto uma estratégia para migração e replicação de um ambiente completo para uma outra infraestrutura sem ter que migrar a VM de um local para outro. Para essa tarefa fizemos o uso de um template e imagem personalizados. Essa mesma estratégia pode ser utilizada como forma de *backup* para casos extremos de catástrofes em uma determinada zona onde nossas instâncias estariam em execução. Assim teríamos como executar esses mesmo serviços em um outro local através dos templates e imagens configurados.
- **Estratégia de transparência de acesso as aplicações;**

  Foi implementado uma estratégia de transparência de acesso às aplicações, pois os veículos não sabem se a aplicação que estão acessando está na nuvem ou na MEC, pois o proxy deixa isso transparente. O mesmo ocorre na outra aplicação pois é transparente para os veículos, pois eles não sabem de quais veículos estão acessando ou para quem estão transferindo o conteúdo.

6.3 TRABALHOS FUTUROS

Através das propostas elencadas por esta dissertação e a leitura da literatura sobre soluções em redes VANETs, abaixo sugerimos algumas possíveis extensões à pesquisa desenvolvida neste trabalho.

- Automatizar todo o processo de implantação e manutenção da infraestrutura *OpenStack* para que haja um funcionamento com alta disponibilidade;
- Implantar políticas de geolocalização nas arquiteturas *Edge Computing* através de determinados serviços que nos possibilite saber de onde está vindo o maior número de requisições e assim replicarmos a aplicação para a borda da rede dessa região;
- Criar estratégias para a criação e gerenciamento de aplicações de *streaming* de vídeo para CDN em conjunto com P2P;
- Criar uma entidade registradora para processamento de aplicações distribuídas.
- Propor um modelo mais seguro de autenticação e criptografia para as mensagens trocadas entre os *containers* associados aos veículos.
- Criar uma estratégia de tomada de decisão para saber qual é o melhor local para acontecer o processamento, na arquitetura fog, MEC ou cloud.
- Adicionar controladores SDN na camada MEC.

REFERÊNCIAS

AL-BADARNEH, J.; JARARWEH, Y.; AL-AYYOUB, M.; FONTES, R.; AL-SMADI, M.; ROTHENBERG, C. Cooperative mobile edge computing system for vanet-based software-defined content delivery. *Computers & Electrical Engineering*, Elsevier, v. 71, p. 388–397, 2018.

AL-SULTAN, S.; AL-DOORI, M. M.; AL-BAYATTI, A. H.; ZEDAN, H. A comprehensive survey on vehicular ad hoc network. *Journal of network and computer applications*, Elsevier, v. 37, p. 380–392, 2014.

ANDROUTSELLIS-THEOTOKIS, S.; SPINELLIS, D. A survey of peer-to-peer content distribution technologies. *ACM computing surveys (CSUR)*, ACM, v. 36, n. 4, p. 335–371, 2004.

BABY, D.; SABAREESH, R.; SARAVANAGURU, R.; THANGAVELU, A. Vcr: vehicular cloud for road side scenarios. In: *Advances in computing and information technology*. [S.l.]: Springer, 2013. p. 541–552.

BASAGNI, S.; CONTI, M.; GIORDANO, S.; STOJMENOVIC, I. *Mobile ad hoc networking: Cutting edge directions*. [S.l.]: John Wiley & Sons, 2013. v. 35.

BEHRISCH, M.; BIEKER, L.; ERDMANN, J.; KRAJZEWICZ, D. Sumo–simulation of urban mobility. In: *The Third International Conference on Advances in System Simulation (SIMUL 2011), Barcelona, Spain*. [S.l.: s.n.], 2011. v. 42.

BITTENCOURT, L. F.; DIAZ-MONTES, J.; BUYYA, R.; RANA, O. F.; PARASHAR, M. Mobility-aware application scheduling in fog computing. *IEEE Cloud Computing*, IEEE, v. 4, n. 2, p. 26–35, 2017.

BIZANIS, N.; KUIPERS, F. A. Sdn and virtualization solutions for the internet of things: A survey. *IEEE Access*, IEEE, v. 4, p. 5591–5606, 2016.

BONOMI, F.; MILITO, R.; ZHU, J.; ADDEPALLI, S. Fog computing and its role in the internet of things. In: ACM. *Proceedings of the first edition of the MCC workshop on Mobile cloud computing*. [S.l.], 2012. p. 13–16.

BOUKERCHE, A.; ROBSON, E. Vehicular cloud computing: Architectures, applications, and mobility. *Computer Networks*, Elsevier, v. 135, p. 171–189, 2018.

CHAHAL, M.; HARIT, S.; MISHRA, K. K.; SANGAIAH, A. K.; ZHENG, Z. A survey on software-defined networking in vehicular ad hoc networks: Challenges, applications and use cases. *Sustainable cities and society*, Elsevier, 2017.

CHIOSI, M.; CLARKE, D.; WILLIS, P.; REID, A.; FEGER, J.; BUGENHAGEN, M.; KHAN, W.; FARGANO, M.; CUI, C.; DENG, H. et al. Network functions virtualisation: An introduction, benefits, enablers, challenges and call for action. In: SN. *SDN and OpenFlow World Congress*. [S.l.], 2012. v. 48.

CONSORTIUM, O. et al. Openfog reference architecture for fog computing. *Architecture Working Group*, 2017.

DASTJERDI, A. V.; BUYYA, R. Fog computing: Helping the internet of things realize its potential. *Computer*, IEEE, v. 49, n. 8, p. 112–116, 2016.

DASTJERDI, A. V.; GUPTA, H.; CALHEIROS, R. N.; GHOSH, S. K.; BUYYA, R. Fog computing: Principles, architectures, and applications. In: *Internet of Things*. [S.l.]: Elsevier, 2016. p. 61–75.

DOCKER. *Docker: The Modern Platform for High-Velocity Innovation*. 2019. Disponível em: <https://www.docker.com/why-docker>.

DOCKER. *Enterprise Container Platform for High-Velocity Innovation*. 2019. Disponível em: <https://www.docker.com/>.

DOCKER. *Networking Overview*. 2019. Disponível em: <https://docs.docker.com/network/>.

DOCKER. *A standardized unit of software*. 2019. Disponível em: <https://www.docker.com/resources/what-container>.

DOCKER. *Swarm mode overview*. 2019. Disponível em: <https://docs.docker.com/engine/swarm/>.

DOCKERHUB. *Build and Ship any Application Anywhere*. 2019. Disponível em: <https://hub.docker.com/>.

ETSI-GR-MEC-017. *Mobile Edge Computing (MEC); Deployment of Mobile Edge Computing in an NFV environment*. 2018. Disponível em: <https://www.etsi.org/deliver/etsi_gr/mec/001_099/017/01.01.01_60/gr_mec017v010101p.pdf>.

ETSI-GS-MEC-003. *Multi-access Edge Computing (MEC); Framework and Reference Architecture*. 2019. Disponível em: <https://www.etsi.org/deliver/etsi_gs/MEC/001_099/003/02.01.01_60/gs_MEC003v020101p.pdf>.

ETSI-GS-NFV. *Network Functions Virtualisation (NFV);Architectural Framework*. 2013. Disponível em: <https://www.etsi.org/deliver/etsi_gs/nfv/001_099/002/01.01.01_60/gs_nfv002v010101p.pdf>.

FONTES, R. dos R.; ROTHENBERG, C. E. Mininet-wifi: Emulaç ao de redes sem fio definidas por software com suporte a mobilidade. 2017.

FONTES, R. R.; AFZAL, S.; BRITO, S. H.; SANTOS, M. A.; ROTHENBERG, C. E. Mininet-wifi: Emulating software-defined wireless networks. In: IEEE. *2015 11th International Conference on Network and Service Management (CNSM)*. [S.l.], 2015. p. 384–389.

GIUST, F.; SCIANCALEPORE, V.; SABELLA, D.; FILIPPOU, M. C.; MANGIANTE, S.; FEATHERSTONE, W.; MUNARETTO, D. Multi-access edge computing: The driver behind the wheel of 5g-connected cars. *IEEE Communications Standards Magazine*, IEEE, v. 2, n. 3, p. 66–73, 2018.

GUEDES, D.; VIEIRA, L. F. M.; VIEIRA, M.; RODRIGUES, H.; NUNES, R. V. Redes definidas por software: uma abordagem sistêmica para o desenvolvimento de pesquisas em redes de computadores. *Minicursos do Simpósio Brasileiro de Redes de Computadores-SBRC*, v. 30, n. 4, p. 160–210, 2012.

HE, Q.; ZHANG, C.; MA, X.; LIU, J. Fog-based transcoding for crowdsourced video livecast. *IEEE Communications Magazine*, IEEE, v. 55, n. 4, p. 28–33, 2017.

HERRERA, J. G.; BOTERO, J. F. Resource allocation in nfv: A comprehensive survey. *IEEE Transactions on Network and Service Management*, IEEE, v. 13, n. 3, p. 518–532, 2016.

HSIEH, H.-C.; LEE, C.-S.; CHEN, J.-L. Mobile edge computing platform with container-based virtualization technology for iot applications. *Wireless Personal Communications*, Springer, v. 102, n. 1, p. 527–542, 2018.

HU, Y. C.; PATEL, M.; SABELLA, D.; SPRECHER, N.; YOUNG, V. Mobile edge computing—a key technology towards 5g. *ETSI white paper*, v. 11, n. 11, p. 1–16, 2015.

KAUSHIK, S. S. Review of different approaches for privacy scheme in vanets. *International Journal Of Advances In Engineering & Technology*, IAET Publishing Company, v. 5, n. 2, p. 356, 2013.

KIM, W.-S.; CHUNG, S.-H. User-participatory fog computing architecture and its management schemes for improving feasibility. *IEEE Access*, IEEE, v. 6, p. 20262–20278, 2018.

KREUTZ, D.; RAMOS, F.; VERISSIMO, P.; ROTHENBERG, C. E.; AZODOLMOLKY, S.; UHLIG, S. Software-defined networking: A comprehensive survey. *arXiv preprint arXiv:1406.0440*, 2014.

LANTZ, B.; HELLER, B.; MCKEOWN, N. A network in a laptop: rapid prototyping for software-defined networks. In: ACM. *Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks*. [S.l.], 2010. p. 19.

LARA, A.; KOLASANI, A.; RAMAMURTHY, B. Network innovation using openflow: A survey. *IEEE communications surveys & tutorials*, IEEE, v. 16, n. 1, p. 493–512, 2013.

LE, M.; SONG, Z.; KWON, Y.-W.; TILEVICH, E. Reliable and efficient mobile edge computing in highly dynamic and volatile environments. In: IEEE. *Fog and Mobile Edge Computing (FMEC), 2017 Second International Conference on*. [S.l.], 2017. p. 113–120.

LI, J.; SHEN, X.; CHEN, L.; VAN, D. P.; OU, J.; WOSINSKA, L.; CHEN, J. Service migration in fog computing enabled cellular networks to support real-time vehicular communications. *IEEE Access*, IEEE, v. 7, p. 13704–13714, 2019.

LUA, E. K.; CROWCROFT, J.; PIAS, M.; SHARMA, R.; LIM, S. et al. A survey and comparison of peer-to-peer overlay network schemes. *IEEE Communications Surveys and tutorials*, v. 7, n. 1-4, p. 72–93, 2005.

MAHMUD, R.; KOTAGIRI, R.; BUYYA, R. Fog computing: A taxonomy, survey and future directions. In: *Internet of everything*. [S.l.]: Springer, 2018. p. 103–130.

MAI, A.; SCHLESINGER, D. Connected vehicles: Service providers at a crossroads. *Cisco Internet Business Solutions Group, April*, 2011.

MARKETSNDMARKETS. *Connected Car Market Size, Share and Market Forecast to 2017 | MarketsndMarkets*. 2019. Disponível em: <https://www.marketsandmarkets.com/Market-Reports/connected-car-market-102580117.html>.

MCKEOWN, N.; ANDERSON, T.; BALAKRISHNAN, H.; PARULKAR, G.; PETERSON, L.; REXFORD, J.; SHENKER, S.; TURNER, J. Openflow: enabling innovation in campus networks. *ACM SIGCOMM Computer Communication Review*, ACM, v. 38, n. 2, p. 69–74, 2008.

MEJRI, M. N.; BEN-OTHMAN, J.; HAMDI, M. Survey on vanet security challenges and possible cryptographic solutions. *Vehicular Communications*, Elsevier, v. 1, n. 2, p. 53–66, 2014.

MELL, P.; GRANCE, T. et al. The nist definition of cloud computing. Computer Security Division, Information Technology Laboratory, National . . . , 2011.

MÜLLER, S.; ROERMUND, T. van; STEIGEMANN, M. Communication beyond vehicles—road to automated driving. In: *Advanced Microsystems for Automotive Applications 2016*. [S.l.]: Springer, 2016. p. 55–64.

NIKOLOUDAKIS, Y.; PANAGIOTAKIS, S.; MARKAKIS, E.; PALLIS, E.; MASTORAKIS, G.; MAVROMOUSTAKIS, C. X.; DOBRE, C. A fog-based emergency system for smart enhanced living environments. *IEEE Cloud Computing*, IEEE, n. 6, p. 54–62, 2016.

NING, Z.; HUANG, J.; WANG, X. Vehicular fog computing: Enabling real-time traffic management for smart cities. *IEEE Wireless Communications*, IEEE, v. 26, n. 1, p. 87–93, 2019.

NOBRE, J. C.; SOUZA, A. M. de; ROSARIO, D.; BOTH, C.; VILLAS, L. A.; CERQUEIRA, E.; BRAUN, T.; GERLA, M. Vehicular software-defined networking and fog computing: Integration and design principles. *Ad Hoc Networks*, Elsevier, v. 82, p. 172–181, 2019.

OLARIU, S.; WEIGLE, M. C. *Vehicular networks: from theory to practice*. [S.l.]: Chapman and Hall/CRC, 2009.

ONF. *Open Networking Foundation*. 2019. Disponível em: <https://www.opennetworking.org/>.

ONF. *Open Networking Foundation: OpenFlow Switch Specification v1.5.1.* 2019. Disponível em: <https://www.opennetworking.org/wp-content/uploads/2014/10/openflow-switch-v1.5.1.pdf>.

ONF. *Open Networking Foundation: Software Defined Networking (SDN).* 2019. Disponível em: <https://www.opennetworking.org/sdn-definition/>.

OPENSTACK. *Documentação oficial do OpenStack*. 2019. Disponível em: <https://docs.openstack.org/stein/>.

OPENSTACK. *Orquestração NFV - OpenStack*. 2019. Disponível em: <https://wiki.openstack.org/wiki/Tacker>.

OUYANG, T.; ZHOU, Z.; CHEN, X. Follow me at the edge: Mobility-aware dynamic service placement for mobile edge computing. *IEEE Journal on Selected Areas in Communications*, IEEE, 2018.

PARAMESWARAN, M.; SUSARLA, A.; WHINSTON, A. B. P2p networking: an information sharing alternative. *Computer*, 2001.

REZNIK, A.; MURILLO, L. M. C.; FANG, Y.; FEATHERSTONE, W.; FILIPPOU, M.; FONTES, F.; GIUST, F.; HUANG, Q.; LI, A.; TURYAGYENDA, C. et al. Cloud ran and mec: a perfect pairing. *ETSI, White Paper*, n. 23, 2018.

SABELLA, D.; SUKHOMLINOV, V.; TRANG, L.; KEKKI, S.; PAGLIERANI, P.; ROSSBACH, R. et al. Developing software for multi-access edge computing. *ETSI White Paper*, v. 20, 2019.

SANTORO, D.; ZOZIN, D.; PIZZOLLI, D.; PELLEGRINI, F. D.; CRETTI, S. Foggy: a platform for workload orchestration in a fog computing environment. In: IEEE. *2017 IEEE International Conference on Cloud Computing Technology and Science (CloudCom)*. [S.l.], 2017. p. 231–234.

SATOH, I. A framework for data processing at the edges of networks. In: SPRINGER. *International Conference on Database and Expert Systems Applications*. [S.l.], 2013. p. 304–318.

SHIN, M.-K.; NAM, K.-H.; KIM, H.-J. Software-defined networking (sdn): A reference architecture and open apis. In: IEEE. *2012 International Conference on ICT Convergence (ICTC)*. [S.l.], 2012. p. 360–361.

SOUA, A.; TOHME, S. Multi-level sdn with vehicles as fog computing infrastructures: A new integrated architecture for 5g-vanets. In: IEEE. *2018 21st Conference on Innovation in Clouds, Internet and Networks and Workshops (ICIN)*. [S.l.], 2018. p. 1–8.

STOJMENOVIC, I.; WEN, S. The fog computing paradigm: Scenarios and security issues. In: IEEE. *2014 Federated Conference on Computer Science and Information Systems*. [S.l.], 2014. p. 1–8.

SUN, X.; ANSARI, N. Edgeiot: Mobile edge computing for the internet of things. *IEEE Communications Magazine*, IEEE, v. 54, n. 12, p. 22–29, 2016.

VAQUERO, L. M.; RODERO-MERINO, L. Finding your way in the fog: Towards a comprehensive definition of fog computing. *ACM SIGCOMM Computer Communication Review*, ACM, v. 44, n. 5, p. 27–32, 2014.

VARGHESE, B.; WANG, N.; NIKOLOPOULOS, D. S.; BUYYA, R. Feasibility of fog computing. *arXiv preprint arXiv:1701.05451*, 2017.

VILALTA, R.; LÓPEZ, V.; GIORGETTI, A.; PENG, S.; ORSINI, V.; VELASCO, L.; SERRAL-GRACIA, R.; MORRIS, D.; FINA, S. D.; CUGINI, F. et al. Telcofog: A unified flexible fog and cloud computing architecture for 5g networks. *IEEE Communications Magazine*, IEEE, v. 55, n. 8, p. 36–43, 2017.

YANG, F.; LI, J.; LEI, T.; WANG, S. Architecture and key technologies for internet of vehicles: a survey. *Journal of Communications and Information Networks*, Springer, v. 2, n. 2, p. 1–17, 2017.

YI, S.; LI, C.; LI, Q. A survey of fog computing: concepts, applications and issues. In: ACM. *Proceedings of the 2015 workshop on mobile big data*. [S.l.], 2015. p. 37–42.

ZHANG, J. A survey on trust management for vanets. In: IEEE. *2011 IEEE International Conference on Advanced Information Networking and Applications*. [S.l.], 2011. p. 105–112.

APÊNDICE A – FERRAMENTAS E AMBIENTES DE AVALIAÇÃO

A.0.1 Mininet Wifi

Mininet-Wifi (FONTES et al., 2015) é um emulador baseado no projeto mininet[1]. O Mininet é descrito como um *software* de rápida prototipagem para um ambiente SDN, seu principal objetivo é emular uma rede completa de *hosts*, *links* e *switches* em uma única máquina(LANTZ; HELLER; MCKEOWN, 2010). O Mininet-WiFi é uma adição de classes sobre o mininet, suportando todos os recursos normais de emulação SDN e virtualização de estações *Wi-Fi* e pontos de acesso usando *drivers wireless* padrão do Linux. A partir dessas adições, ele suporta modelos de mobilidade e propagação.

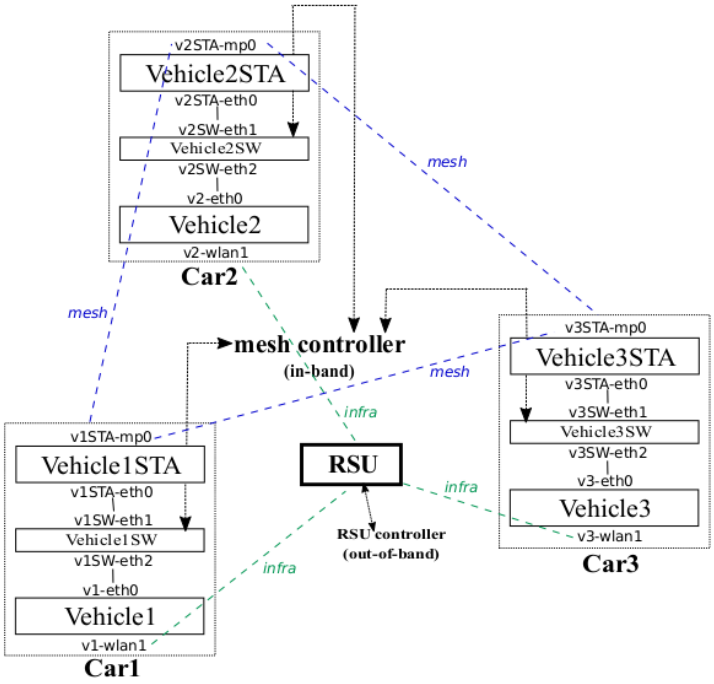
A virtualização *Wi-Fi* é possível usando *drivers* Linux, ou seja, o mac802.11_hsim e *namespaces*. Os *drivers* permitem que a simulação dos padrões de rádio 802.11 IEEE e *namespaces* sejam usados para a criação da rede virtual, fornecendo um conjunto de interfaces, endereços IP e tabelas de roteamento. O Mininet-WiFi é atualizado regularmente, está escrito em Python e usa *software* de código aberto[2].

O emulador possui uma a implementação de uma arquitetura *node car*, permitindo cenários de VANETs realísticos. As comunicações entre os carros e as RSUs são estabelecidas no modo sem fio no modelo V2I. Já as comunicações V2V entre os carros, são estabelecidas no modo sem fio usando outras interfaces sem fio. Além disso, a arquitetura ainda permite o uso da programação SDN, usando todas as interfaces do veículo ao mesmo tempo, sendo possível escolher qualquer controlador *OpenFlow*, seja centralizado ou distribuído (FONTES; ROTHENBERG, 2017). A Figura 26 apresenta a arquitetura interna de um veículo do Mininet-WiFi.

Para permitir que controladores *OpenFlow* controle nós simples, como um carro, foi necessário criar uma arquitetura com um *switch*. Assim, todo pacote que vem do veículo# precisa passar pelo veículo#SW se o destino for um nó da rede de malha sem fio V2V. A comunicação entre carros e RSUs ou seja, comunicações V2I, por sua vez, é feita diretamente entre o veículo# e a RSU. Através dessa configuração inicial já existente na arquitetura interna de cada veículo do Mininet-Wifi com a utilização de um *switch* virtual, foi possível fazermos a implementação da OBU para que fosse possível executarmos nossas aplicações em *containers* acoplada ao *switch* interno de cada veículo, tal implementação irá ser descrita posteriormente.

[1] http://mininet.org/
[2] https://github.com/intrig-unicamp/mininet-WiFi

Figura 26 – Arquitetura Node Car



Fonte:(FONTES; ROTHENBERG, 2017)

A.0.2 Simulador de Mobilidade - SUMO

Ferramentas de simulação de tráfego emulam o fluxo de veículos em redes de transporte. Tradicionalmente, os simuladores de tráfego podem ser classificados como microscópicos, macroscópicos e mesoscópicos, dependendo da granularidade: os simuladores microscópicos modelam o tráfego em grande escala, tratando cada entidade (por exemplo, carros, trens) individualmente; ferramentas macroscópicas são usadas para modelar o tráfego em grande escala, tratando o tráfego como um fluido; Simulação mesoscópica combina as propriedades de ambos os modelos de simulação microscópicos e macroscópicos, descrevendo a realidade em termos de tráfego agregado de características de veículos, evitando as questões de tempo e complexidade de implementação de cenários microscópicos(OLARIU; WEIGLE, 2009). No entanto, os cenários de VANETs requerem modelos precisos de comunicação entre nós e sua posição exata, o que torna as soluções macroscópicas e mesoscópicas não tão confiáveis, uma vez que oferecem menos detalhes quando comparadas a simulações microscópicas.

Utilizando a microssimulação é possível criar modelos virtuais que representem a mobilidade individual dos veículos em uma rede de tráfego e construir cenários para avaliar o comportamento dos veículos em um determinado local quando são realizadas alterações na infraestrutura rodoviária.

O SUMO é escrito em C++, e é uma plataforma de simulação de tráfego de características microscópicas, e multimodal, onde a representação dos veículos adotam medidas de

espaço-contínuo e tempo-discreto, destinado ao planejamento de otimização de projetos rodoviários(BEHRISCH et al., 2011). Foi desenvolvido em sua maior parte pelo Instituto de Sistemas de Transportes situado no Centro Aeroespacial da Alemanha. O início de seu desenvolvimento deu-se em 2001 e tão logo, em 2002, foi disponibilizada uma versão opensource à comunidade com a licença *(GNU Public Licence - GPL)*. A ferramenta foi desenvolvida com dois propósitos: ser aberta a comunidade para que fosse acessível a estudantes e pesquisadores, e também ser um simulador que recebesse suporte de outras instituições.

Ao longo dos anos, o SUMO[3] se transformou em uma suíte de aplicações completa para modelagem de sistemas rodoviários, incluindo a capacidade de se comunicar com outras aplicações como *OpenStreetMaps*[4], *PTV Vissim*[5], entre outros, e ainda conta com uma interface programável, nomeada de *(Traffic Communication Interface - TraCI)*. A característica microscópica deste simulador permite com que os veículos sejam modelados individualmente dado uma rede rodoviária. Cada veículo possui sua própria rota e movem-se individualmente pelas vias determinadas. Apesar de utilizar elementos determinísticos, é possível empregar certa aleatoriedade na simulação.

A.0.3 Controlador SDN -*Ryu*

O controlador *Ryu* é um componente criado para o contexto das redes definidas por *software*. Todo seu código está disponível gratuitamente com a licença de Apache 2.0 e está completamente escrito na linguagem de programação *Python*, facilitando, assim, a criação de novas aplicações de controle.

O *Ryu* fornece componentes de *software* com APIs muito bem definidas, o que facilita aos desenvolvedores a criação de novas formas de gerenciamento de redes e aplicações de controle. *Ryu* dá suporte a vários protocolos para o gerenciamento dos equipamentos de rede como *Netconf*, *OF-config*, *OpenFlow*. Particularmente, para este último dá suporte às versões 1.0 até a 1.5, característica essencial para sua utilização na arquitetura desenvolvida neste trabalho.

Devido a sua compatibilidade com os protocolos *OpenFlow*, seu desenvolvimento ativo, documentação atualizada e de fácil acesso, ele foi escolhido para ser utilizado nesse trabalho.

A.0.4 *OpenStack*

O projeto *OpenStack* é uma colaboração global de desenvolvedores e tecnólogos de Computação em Nuvem produzindo uma plataforma de Computação em Nuvem de código aberto, única e padronizada para nuvens públicas e privadas. O projeto visa oferecer so-

[3] (http://sumo.sourceforge.net/)
[4] https://www.openstreetmap.org
[5] http://vision-traffic.ptvgroup.com/nl/products/ptv-vissim/

luções para todos os tipos de nuvens por ser simples de implementar, altamente escalável, e rico em recursos. A tecnologia consiste em uma série de projetos inter-relacionados responsáveis pelo desenvolvimento de componentes que, juntos constituem uma solução de infraestrutura em Nuvem. O *OpenStack* pode ser instalado em qualquer versão GNU Linux a partir de seu código fonte ou do repositório oficial da distribuição Linux (OPENSTACK, 2019a).

A arquitetura funcional é construída com um nó de rede comumente chamado de *controller* para o controle e gerenciamento dos serviços e um ou mais nós de computação comumente chamado de *compute node* para armazenamento e provisionamento das máquinas virtuais. Dentre os principais serviços e projetos tidos como base para o bom funcionamento inicial do *OpenStack* estão o ***Nova*** para gerenciar o ciclo de vida das VMs, ***Keystone*** que fornece identidade e autenticação para todos os serviços, ***Glance*** para fornecer o repositório de imagens de computação. Todas as instâncias de computação são iniciadas a partir de imagens de determinados sistemas operacionais. ***Neutron*** que é responsável pelo provisionamento das redes virtuais ou físicas que as instâncias são conectadas durante sua inicialização e o ***Horizon***, uma interface gráfica para ter acesso e controle à todos os serviços, dinamizando o gerenciamento dos recursos da nuvem (OPENSTACK, 2019a).

Para a implementação da arquitetura proposta, foi utilizado esse gerenciador de nuvem. Ele controla, gerencia e automatiza amplos serviços de computação, armazenamento e recursos de rede e faz o monitoramento dos recursos computacionais disponíveis. O gerenciador disponibiliza aos usuários um ponto de controle central, por meio dos quais novos recursos são adicionados, máquinas virtuais são criadas, redes virtuais são configuradas e novas imagens são disponibilizadas manualmente ou por meio de templates pré-configurados. Além de dar suporte à infraestrutura de *container* computacional e funções de rede virtuais.

A.0.4.1 *Tacker* e Serviços NFV

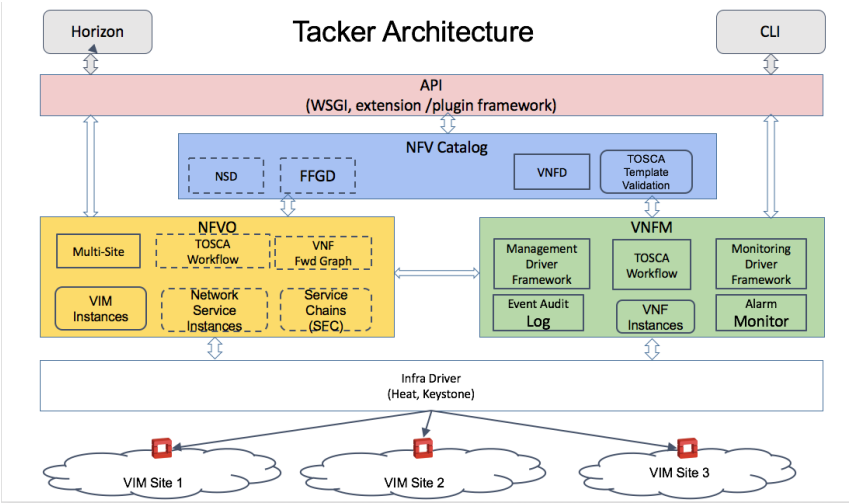
O modelo NFV permite que funções de rede sejam migradas para a nuvem pela criação de VNFs para substituir equipamentos físicos. Essa estratégia favorece o desenvolvimento ágil de novos serviços, pois agrega flexibilidade e promove a redução de custos operacionais (ETSI-GS-NFV, 2013). Nesse contexto, a plataforma de computação em nuvem *OpenStack* pode ser empregada para gerenciar a infraestrutura virtualizada e um de seus módulos, o *Tacker*, habilita a orquestração e gerenciamento de VNFs.

Existem diferentes tipos de orquestradores NFV para gerenciar as VNFs implantadas em uma ou em um conjunto de infraestrutura, tais como *Tacker* [6], *Cloudify* [7], *Open*

[6] https://wiki.openstack.org/wiki/Tacker
[7] https://cloudify.co/

Source Mano [8], *Open Baton* [9], ONAP [10] e assim por diante. O orquestrador escolhido para esse trabalho foi o *Tacker* que é um projeto oficial do *OpenStack* que executa um (*Network Functions Virtualization Manager - VNFM*) e um NFVO para implantar e operar serviços de rede e funções de rede virtual VNFs em uma plataforma de infraestrutura NFV como o *OpenStack*. Ele é um projeto baseado na arquitetura ETSI MANO e fornece uma pilha funcional para orquestrar os serviços de rede de ponta a ponta usando VNFs (OPENSTACK, 2019b). A arquitetura de alto nível do *tacker* é representada na Figura 27. Os componentes são descritos conforme a definição na documentação oficial do projeto.

Figura 27 – Arquitetura Tacker Alto Nível.



Fonte: (OPENSTACK, 2019b)

- NFV *Catalog* - Responsável por descrever as funções e serviços de rede, e dos gráficos de encaminhamento das VNFs.
- VNFM - VNF Manager provê o ciclo básico de vida das VNFs como criação, atualização e exclusão, monitoramento do comportamento das VNFs, escalonamento baseado em políticas que facilita a configuração inicial das VNFs.
- NFVO - O orquestrador é responsável por prover o *template* com os serviços de rede fim a fim, políticas de instanciação das VNFs, conectar VNFs utilizando (*Service Function Chaining - SFC*), verificação e alocação de recursos do VIM, capacidade de orquestrar VNFs em vários VIMs. Funciona como um orquestrador central para unir as funções de redes instanciadas em infraestruturas distintas e coordenar os serviços em ambientes NFV.

A exploração das funcionalidades do ambiente *OpenStack* juntamente com o módulo *Tacker* viabilizam a composição de serviços virtualizados de rede. Essas funcionalidades

[8] https://osm.etsi.org/
[9] https://openbaton.github.io/
[10] https://www.onap.org/

são: (i) gerência do ciclo de vida de VNFs; (ii) monitoramento de VNFs; (iii) escalabilidade e (iv) encadeamento de VNFs (OPENSTACK, 2019a).

A.0.4.2 BTSync

BTSync[11] é uma ferramenta que utiliza o protocolo do *Bittorrent* [12] para o sincronismo de dados, através dele é possível sincronizar pastas e arquivos entre diferentes plataformas (desde que tenha o *BTSync* instalado), de forma bastante rápida. O *BitTorrent* é um protocolo de transferência de arquivos que, juntamente, com alguns outros *websites* torrentes como Piratebay.org, kickasstorrent.cr e servidores *trackers*, provê atualmente, um dos maiores sistemas de compartilhamento de arquivos. No *BitTorrent* os arquivos são divididos em pedaços menores (*chunks*) que podem ser encontrados através de chaves (*hash*) únicas e o *peer* passa a disponibilizar cada arquivo para que outros usuários possam realizar o *download* do mesmo, esses usuários são classificados como (*Leecher*). Quando outros *peers* obtêm os arquivos, esses passam a também disponibilizá-los formando assim um *cluster (Swarm)*. Os servidores *tracker* tem a função de manter uma lista atualizada de quais *peers* possuem cada *chunk*.

A.0.4.3 Traefik

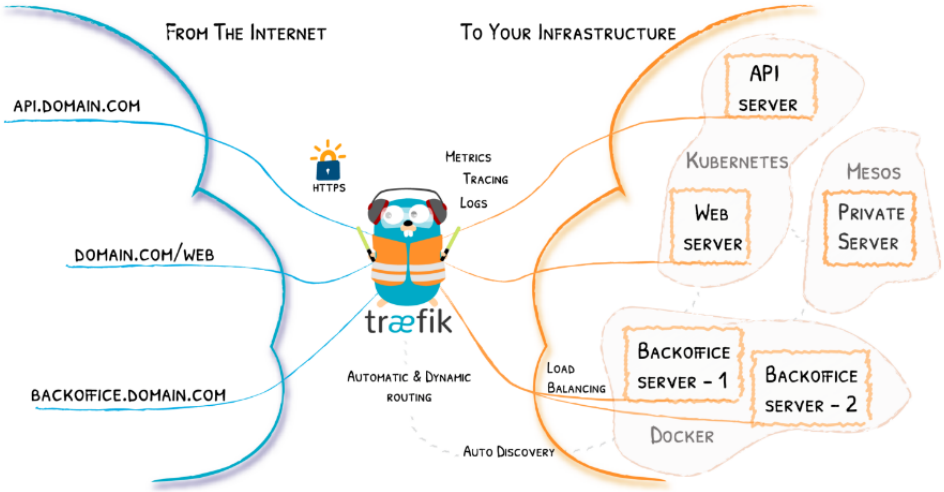
O *Traefik*[13] é um moderno *proxy* reverso HTTP e balanceador de carga que facilita a implantação de aplicações em microsserviços. O *Traefik* integra-se com os orquestradores e componentes de infraestrutura existentes tais como: Docker[14], *Swarm*[15], Kubernetes[16], *Marathon*[17], *Consul*[18], *Rancher*[19], Amazon ECS[20], ...) e configura-se automática e dinamicamente a qualquer um dos orquestradores citados.

Devido a um maior amadurecimento na utilização de plataformas de orquestração de *containers* e as novas tecnologias de *SaaS* e suas evoluções, o orquestrador utilizado foi o Docker *Swarm*, devido a evolução natural das tecnologias e os ganhos de escalabilidade e disponibilidade, devido ao conceito de *cluster* computacional implementado por essa plataforma. Também neste ciclo, foi adicionado o *Traefik* como ferramenta fundamental para fazer a descoberta de serviços, e responsável por fazer o roteamento interno entre os mesmos.

[11] https://www.resilio.com/
[12] https://www.bittorrent.com/lang/pt/
[13] https://docs.traefik.io/
[14] https://www.docker.com/
[15] https://docs.docker.com/engine/swarm/
[16] https://kubernetes.io/pt/
[17] https://mesosphere.github.io/marathon/
[18] https://www.consul.io/
[19] https://rancher.com/
[20] https://aws.amazon.com/pt/ecs/

Traefik é implementado em *Golang*[21] que é facilmente configurável e customizável, outro benefício deste *proxy* em relação a outras soluções como o *Nginx*[22], é a sua integração com os controladores de *containers*. O *Traefik* foi desenvolvido para atender a demanda de requisições *Web*, então ele pode ser utilizado para fazer o roteamento das conexões de um site ou API para o *container* ou serviço que foi criado para atender a determinada demanda. Um dos requisitos para a viabilidade da nossa arquitetura desenvolvida é a capacidade de adaptação dinâmica da mesma. Este requisito é inimigo de plataformas que fazem uso de arquivos de configuração estáticas, esta necessidade foi o principal motivador da escolha do *Proxy Traefik*, que suporta reconfiguração dinâmica. Uma vez configurado o *Traefik* fica ativo e escutando em uma determinada porta a espera de requisições, para que possa repassar automaticamente essas requisições aos *containers* específicos. A imagem 28 é um exemplo clássico, e explica bem esse comportamento:

Figura 28 – Modelo de Alto nível do Funcionamento do Traefik.



Fonte: Traefik.io (2019)

Como podemos ver, o *traefik* fica "escutando" as ações que ocorrem no orquestrador, independente de qual tenha sido escolhido, e baseado nessas ações, se reconfigura para garantir o acesso ao serviço que está sendo executado no *container* criado. Dessa forma, basta redirecionarmos as requisições *Web* (http ou https) para o servidor onde o *traefik* está em execução.

[21] https://golang.org/
[22] https://www.nginx.com/

APÊNDICE B – CÓDIGOS DAS APLICAÇÕES QUE FORAM EXECUTADAS EM CONTAINERS DOCKER

Listing B.1 – Código para a criação e atribuição do hash para a aplicação P2P

```
1
  /*
3 #!/bin/sh
  #
5 # Codigo para a criacao e atribuicao do hash para a aplicacao P2P
  #
7 # Edigleison Barbosa: <fesb@cin.ufpe.br> ou <barbosaedigleison@gmail.com>
  #
9 */

11 //# criando o diretorio de configuracao

13 mkdir -p /etc/rslsync

15 //# Atribuindo o nome do dispositivo

17 if [ -z $RSLSYNC_NAME ]; then
     RSLSYNC_NAME=`hostname`
19 fi

21 //# Atribuindo o caminho do diretorio a ser criado

23 if [ -z $RSLSYNC_PATH ]; then
     RSLSYNC_PATH="/data"
25 fi

27 //# Criacao da base de configuracao

29 cat <<EOT > /etc/rslsync/rslsync.conf
   {
31   "storage_path" : "/root",
     "listening_port" : 33333,
33   "use_upnp" : false,
     "vendor" : "docker",
35   "max_file_size_for_versioning" : $RSLSYNC_SIZE,
     "sync_trash_ttl" : $RSLSYNC_TRASH_TIME,
37   "device_name" : "$RSLSYNC_NAME",
     "pid_file" : "/var/run/rslsync.pid",
39 EOT

41   //# Criando a chave secreta
   if [ -z $RSLSYNC_SECRET ]; then
43     RSLSYNC_SECRET=`/usr/local/bin/rslsync --generate-secret`
       echo "add -e RSLSYNC_SECRET=$RSLSYNC_SECRET para sicronizar outros nos"
45   fi

47   cat <<EOT >> /etc/rslsync/rslsync.conf
     "shared_folders" :
49   [
```

```
    {
51    "secret" : "$RSLSYNC_SECRET",
      "dir" : "$RSLSYNC_PATH",
53    "use_relay_server" : true,
      "use_tracker" : true,
55    "search_lan" : true,
      "use_sync_trash" : $RSLSYNC_TRASH,
57    "overwrite_changes" : false,
      "known_hosts" :
59    [
      ]
61  }
  ]
63 }
EOT
```

Listing B.2 – Código do Docker file para a aplicação P2P

```
2 /*
  #
4 # Codigo do Docker file para a aplicacao P2P
  #
6 # Edigleison Barbosa: <fesb@cin.ufpe.br> ou <barbosaedigleison@gmail.com>
  #
8 */

10 FROM nimmis/alpine-glibc

12 MAINTAINER Edigleison Barbosa <fesb@cin.ufpe.br>

14 ENV RSLSYNC_SIZE=1000 \
       RSLSYNC_TRASH_TIME=30 \
16     RSLSYNC_TRASH=true
   COPY root/. /
18
   RUN apk update && apk upgrade && \
20     apk add zip curl && \
       cd /root && \
22     curl https://download-cdn.resilio.com/stable/linux-x64/resilio-sync_x64.tar.gz |
           tar xfz - && \
       mv rslsync /usr/local/bin && \
24     rm -rf /var/cache/apk/

26 VOLUME /data

28 EXPOSE 33333
   EXPOSE 8888
```

Listing B.3 – Código para a utilização do Traefik como proxy reverso, domínio DNS e balanceador de carga

```
1
  /*
3 #
  # Codigo Docker para a utilizacao do Traefik como proxy reverso, dominio DNS e #
      balanceador de carga
```

```
5  #
   # Edigleison Barbosa: <fesb@cin.ufpe.br> ou <barbosaedigleison@gmail.com>
7  #
   */
9
   version: "3.3"
11
   services:
13   traefik:
       image: traefik
15     command: --web --docker --docker.swarmmode --docker.watch --docker.domain=edi.
           ufpe --logLevel=DEBUG
       ports:
17       - "80:80"
         - "8080:8080"
19       - "443:443"
       volumes:
21       - /var/run/docker.sock:/var/run/docker.sock
         - /dev/null:/traefik.toml
23     labels:
         - "traefik.enable=false"
25     networks:
         - edi-net
27     deploy:
         replicas: 3
29       placement:
           constraints: [node.role==manager]
31       restart_policy:
           condition: on-failure
33
     web:
35     image: emilevauge/whoami
       networks:
37       - edi-net
       deploy:
39       replicas: 3
         labels:
41         - "traefik.port=80"
           - "traefik.docker.network=edi-net"
43         - "traefik.frontend.rule=Host:whoami.edi.ufpe"
           - "traefik.backend.loadbalancer.sticky=true"
45
     visualizer:
47     image: dockersamples/visualizer:stable
       ports:
49       - "8081:8080"
       stop_grace_period: 1m30s
51     volumes:
         - "/var/run/docker.sock:/var/run/docker.sock"
53     deploy:
         placement:
55         constraints: [node.role==manager]

57 networks:
     edi-net:
59     driver: overlay
```