GARNET: An Edge Virtualized Everything Function Management Architecture

Hugo Gustavo Valin Oliveira da Cunha

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

Uberlândia
2021

Hugo Gustavo Valin Oliveira da Cunha

GARNET: An Edge Virtualized Everything Function Management Architecture

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

Área de concentração: Ciência da Computação

Orientador: Flávio de Oliveira Silva

Uberlândia
2021

Dados Internacionais de Catalogação na Publicação (CIP)
Sistema de Bibliotecas da UFU, MG, Brasil.

C972g 2021

Cunha, Hugo Gustavo Valin Oliveira da, 1997-

Garnet [recurso eletrônico] : an edge virtualized everything function management architecture / Hugo Gustavo Valin Oliveira da Cunha. - 2021.

Orientador: Flávio de Oliveira Silva.

Dissertação (mestrado) - Universidade Federal de Uberlândia. Programa de Pós-Graduação em Ciência da Computação.

Modo de acesso: Internet.

Disponível em: http://doi.org/10.14393/ufu.di.2021.5578

Inclui bibliografia.

Inclui ilustrações.

1. Computação. I. Silva, Flávio de Oliveira, 1970-, (Orient.). II. Universidade Federal de Uberlândia. Programa de Pós-Graduação em Ciência da Computação. III. Título.

CDU: 681.3

Glória Aparecida
Bibliotecária - CRB-6/2047

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

ATA DE DEFESA - PÓS-GRADUAÇÃO

Programa de Pós-Graduação em:	Ciência da Computação				
Defesa de:	Mestrado Acadêmico, 20/2021, PPGCO				
Data:	11 de agosto de 2021	Hora de início:	08:05	Hora de encerramento:	10:22
Matrícula do Discente:	11922CCP005				
Nome do Discente	Hugo Gustavo Valin Oliveira da Cunha				
Título do Trabalho:	GARNET: An Edge Virtualized Everything Function Management Architecture				
Área de concentração:	Ciência da Computação				
Linha de pesquisa:	Sistemas de Computação				
Projeto de Pesquisa de vinculação:	-				

Reuniu-se, por videoconferência, a Banca Examinadora, designada pelo Colegiado do Programa de Pós-graduação em Ciência da Computação, assim composta: Professores Doutores: Pedro Frosi Rosa - FACOM/UFU; Lisandro Zambenedetti Granville - INF/UFRGS e Flávio de Oliveira Silva - FACOM/UFU, orientador do candidato.

Os examinadores participaram desde as seguintes localidades: Lisandro Zambenedetti Granville - Porto Alegre/RS; Pedro Frosi Rosa e Flávio de Oliveira Silva - Uberlândia/MG. O discente participou da cidade de Uberlândia/MG.

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

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

Aprovado.

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

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

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

Documento assinado eletronicamente por **Pedro Frosi Rosa**, **Professor(a) do Magistério Superior**, em 01/09/2021, às 14:07, conforme horário oficial de Brasília, com fundamento no art. 6º, § 1º, do Decreto nº 8.539, de 8 de outubro de 2015.

Documento assinado eletronicamente por **Flávio de Oliveira Silva**, **Professor(a) do Magistério Superior**, em 02/09/2021, às 09:25, conforme horário oficial de Brasília, com fundamento no art. 6º, § 1º, do Decreto nº 8.539, de 8 de outubro de 2015.

Documento assinado eletronicamente por **Lisandro Zambenedetti Granville**, **Usuário Externo**, em 03/09/2021, às 08:47, conforme horário oficial de Brasília, com fundamento no art. 6º, § 1º, do Decreto nº 8.539, de 8 de outubro de 2015.

Dedico esse trabalho ao Supremo Criador, o Rei da Eternidade, Jeová Deus. (Isaías 40:26; 42:8)

Agradecimentos

Agradeço primeiramente a Jeová Deus, por estar ao meu lado, ter me apoiado e consolado, e em especial, nos momentos mais difíceis da minha vida. Se consegui chegar até aqui, e ser o que sou, nunca foi por força própria, mas pela força ativa de Jeová Deus, seu espírito santo.

Agradeço a minha mãe, Cleusdete Maria de Oliveira, e ao meu pai, Jair Pinto da Cunha Júnior, que sempre me amaram, me apoiaram, e se esforçaram tanto para me dar uma vida digna e a oportunidade de estudar que eles não puderam ter. Espero que possa retribuir-lhes por todo carinho e amor que eles têm por mim. Espero muito um dia, poder fazer muito mais por eles do que eles fizeram por mim. Vocês são as pessoas mais importantes da minha vida!

Ao Prof. Dr. Flávio de Oliveira Silva, pela orientação na condução deste trabalho, pela confiança, pelos conselhos em momentos que me sentia perdido, pela motivação quando queria desistir, pela compreensão quando queria fazer diferente e principalmente, pela parceria em todos os momentos dessa jornada.

A todos meus colegas da UFU, em especial ao Rodrigo Moreira e Diego Nunes Molinos que por entre conversas do corredor, puderam me acalmar e me ajudar a entender quem eu sou, onde me encontrava, e por onde poderia caminhar.

A BRF S.A, a todos seus funcionários, direção e administração cujo propósito é o de uma vida melhor para a sociedade, baseado em segurança, excelência e integridade, qualidades estas que me motiva a olhar para o futuro e ser alguém melhor todos os dias.

Aos professores do Programa de Pós-graduação em Ciência da Computação da Universidade Federal de Uberlândia (UFU) por partilharem experiências e conhecimentos.

A Universidade Federal de Uberlândia, seu corpo docente, direção e administração que proporcionaram um novo caminho, baseado em confiança, mérito e ética, qualidades estas presentes na instituição.

"Eu, Jeová, sou o seu Deus, Aquele que ensina o que é melhor para você, Aquele que o guia no caminho em que deve andar" - Isaías 48:17

Resumo

A *Internet of Things* representa um ambiente abrangente que conecta um grande número de objetos físicos heterogêneos, como eletrodomésticos, instalações, animais, veículos, fazendas, fábricas, etc., à Internet, a fim de aumentar a eficiência de aplicativos, como logística, manufatura, agricultura, computação urbana, automação residencial, vida assistida e vários aplicativos de computação em tempo real.

Embora *Cloud Computing* tenha sido uma tecnologia de capacitação chave para a Internet das Coisas, um pequeno aumento na porcentagem de objetos ciberfísicos ou conectados representa uma mudança dramática no espaço de recursos de computação e um potencial tsunami de hiperconectividade, onde a infraestrutura de hoje vai lutar para acomodar os níveis históricos na qualidade de serviço.

O *Multi-Access Edge Computing* preenche a lacuna entre a nuvem e os dispositivos de *Internet of Things*, permitindo computação, armazenamento, rede e gerenciamento de dados em nós de rede próximos aos dispositivos finais.

Network Functions Virtualization faz a transição das funções de serviço de rede de *hardware* para *software*. Juntos, *Multi-Access Edge Computing* e *Network Functions Virtualization* fornecem uma solução eficaz para a computação moderna, garantindo velocidade e confiabilidade para os usuários finais. Um benefício de usar ambas tecnologias é a combinação de escalabilidade e baixa latência. *Multi-Access Edge Computing* desempenha um papel significativo no alívio e solução das demandas de rede, como *Internet of Things*.

Portanto, o objetivo principal deste trabalho é propor e avaliar uma arquitetura compatível com o a pilha de software *ETSI Management and Orchestration* para gerenciamento de funções virtualizadas em *Multi-Access Edge Computing* implantado em dispositivos de baixo custo e servidores bare metal de forma integrada, que ofereça um monitoramento unificado nativo do hardware até os serviços.

Palavras-chave: Virtualização de Máquina. Virtualização de Funções de Rede. Internet das Coisas. Computação de Borda. Monitoramento de Sistemas Distribuídos. .

Abstract

The Internet of Things represents a comprehensive environment that connects a large number of heterogeneous physical objects, such as appliances, facilities, animals, vehicles, farms, factories, etc., to the Internet to increase the efficiency of applications such as logistics, manufacturing, agriculture, urban computing, home automation, assisted living and various real-time computing applications.

While Cloud Computing has been a key enabling technology for the Internet of Things, a small increase in the percentage of cyber-physical or connected objects represents a dramatic shift in the computing resource space and a potential tsunami of hyper-connectivity. Today's infrastructure will struggle to accommodate historic levels of service quality.

Multi-Access Edge Computing bridges the gap between cloud and Internet of Things devices, enabling computing, storage, networking and data management on network nodes close to the end devices.

Network Function Virtualization transitions network service functions from hardware to software. Together, Multi-Access Edge Computing and Network Functions Virtualization provide an effective solution for modern computing, ensuring speed and reliability for end-users. One benefit of using both technologies is the combination of scalability and low latency. Multi-Access Edge Computing plays a significant role in alleviating and solving network demands such as the Internet of Things.

Therefore, the main objective of this work is to propose and evaluate an architecture compatible with the ETSI Management and Orchestration software stack for managing virtualized functions in Multi-Access Edge Computing deployed in low-cost devices and bare metal servers in an integrated way, that offers unified native monitoring from hardware to services.

Keywords: Machine Virtualization. Network Function Virtualization. Internet of Things. Edge Computing. Distributed Systems Monitoring.

List of Figures

List of Tables

Acronyms list

API Application Programming Interface

ASIC Application Specific Integrated Circuits

AR Augmented Reality

CAPEX Capital Expenditure

CIoT Cloud-centric Internet of Things

CFS Completely Fair Scheduler

CLI Command-line Interface

CNCF Cloud Native Computing Foundation

CoAP Constrained Application Protocol

CoRE Constrained RESTful Environments

EM Element Management

EMS Element Management System

FDU Fog Deployment Unit

FIM Fog Infrastructure Manager

FOrcE Fog Orchestration Engine

FSM Finite State Machine

GPL General Public License

GNF Glasgow Network Functions

HCI Human-computer interaction

ICT Information and Communication Technology

IETF Internet Engineering Task Force

IoT Internet of Things

ISG ETSI Industry Specification Group of European Telecommunications Standards Institute

JIT Just-In-Time

M2M Machine to Machine

MAC Mandatory Access Control

MANO Management and Orchestration

MEC Multi-Access Edge Computing

MOM Message Oriented Middleware

MR Mixed Reality

MQTT Message Queuing Telemetry Transport

NFVI Network Functions Virtualization Infrastructure

NFVO NFV Orchestrator

OPEX Operational Expenditure

OSS/BSS Operations Support System and Business Support System

QoS Quality of Service

RPi Raspberry Pi

RTT Round-Trip Time

SDN Software Defined Networking

TCG Tiny Code Generator

URI Universal Resource Identifier

VIM Virtualized Infrastructure Manager

vIoT Virtualized Internet of Things

VNF Virtualized Network Function

VNFM Virtualized Network Function Manager

VM Virtual Machine

VMM Virtual Machine Manager

VR Virtual Reality

VxF Virtualized Everything Functions

Contents

Chapter 1

Introduction

The Internet of Things (GUBBI et al., 2013) represents a comprehensive environment that interconnects a large number of heterogeneous physical objects or things such as appliances, facilities, animals, vehicles, farms, factories, etc., in order to enhance the efficiency of the applications such as logistics, manufacturing, agriculture, urban computing, home automation, environment assisted living, and various real-time ubiquitous computing applications.

Commonly, an IoT system follows the architecture of the Cloud-centric Internet in which the physical objects are represented in the form of Web resources that are managed by the servers in the global Internet (CHANG; SRIRAMA; BUYYA, 2016a). Fundamentally, in order to interconnect the physical entities to the Internet, the system will use various front-end devices such as wired or wireless sensors, actuators, and readers to interact with them. Further, the front-end devices have the Internet connectivity via the mediate gateway nodes such as Internet modems, routers, switches, cellular base stations, and so on. In general, the common IoT system involves three major technologies: embedded systems, middleware, and cloud services, where the embedded systems provide intelligence to the front-end devices, middleware interconnects the heterogeneous embedded systems of front-end devices to the cloud and finally, the cloud provides comprehensive storage, processing, and management mechanisms.

Although the Cloud-centric model is a common approach to implement IoT systems, it is facing the growing challenges in IoT. Specifically, Cloud-centric faces challenges like bandwidth, latency, uninterrupted, resource-constraint, and security (CHIANG; ZHANG, 2016).

In the last decade, several approaches have tried to extend the centralized cloud computing to a more geo-distributed manner in which the computational, networking, and storage resources can be distributed to the locations that are much closer to the data sources or end-user applications. For example, the geo-distributed cloud-computing model (SAJJAD et al., 2016) tends to partition the portions of processes to the data centers near the edge network. Further, the mobile cloud computing model introduced the

physical proximity-based cloud computing resources provisioned by the local wireless Internet access point providers. Moreover, academic research projects (LOKE et al., 2015) have experimented with the feasibility of the mobile ad hoc network (MANET)-based cloud using the advanced RISC machine (ARM)-powered devices. Among the various approaches, the industry-led Edge Computing architecture, which was first introduced by Cisco research (BONOMI et al., 2012), has gained the most attention.

For example, a edge-enabled IoT system can distribute the simple data classification tasks to the IoT devices and assign the more complicated context reasoning tasks at the gateway devices. Further, for the analytics tasks that involve terabytes of data, which requires higher processing power, the system can further move the processes to the resources at the core network such as the data centers of wide area network (WAN) service providers or it can utilize the cloud. Certainly, the decision of where the system should assign the tasks among the resources across different tiers depends on efficiency and adaptability. For example, smart systems may need to assign certain decision-making tasks to the edge devices in order to provide timely notification about the situation, such as the patient's condition in the smart healthcare, the security state of the smart home, the traffic condition of the smart city, the water supply condition of smart farming, or the production line operation condition of a smart factory.

The industry has seen edge as the main trend for the practical IoT systems, like energy/utilities, transportation, healthcare, industry, and agriculture, and the leading OpenFog consortium has established collaboration with major industrial standard parties such as European Telecommunications Standards Institute and IEEE Standard for Edge Computing.

1.1 Motivation

While cloud computing has been a key enabling technology for the IoT, a small increase in the percentage of connected or cyber-physical objects represents dramatic change in the feature space of computing and a potential tsunami of computation and hyper-connectivity, which today's infrastructure will struggle to accommodate at historic levels of quality of service. Large-scale distributed control systems, geo-distributed applications, time-dependent mobile applications, and applications that require very low and predictable latency or interoperability between service providers are just some of the IoT application categories that existing cloud infrastructures are not well-equipped to manage at a hyperscale (BONOMI et al., 2014). Traditional cloud computing architectures were simply not designed with an IoT, characterized by extreme geographic distribution, heterogeneity and dynamism, in mind. As such, a novel approach is required to meet the requirements of IoT including transversal requirements (scalability, interoperability, flexibility, reliability, efficiency, availability, and security) as well as cloud-to-thing com-

putation, storage and communication needs (BOTTA et al., 2016).

In response to the need for a new intermediary layer along the cloud-to-thing continuum, Edge Computing, especially Multi-Access Edge Computing, has emerged as a computing paradigm situated between the cloud and smart end-devices providing data management and/or communications services to facilitate the execution of relevant IoT applications. The ambition for Multi-Access Edge Computing is greater support for interoperability between service providers, real-time processing and analytics, mobility, geographic distribution, and different device or MEC node form factors, and as a result the achievement of quality of service expectations. Despite these advantages, Multi-Access Edge Computing adds a layer of complexity that operators across the cloud-to-thing continuum need to account for, not least resource orchestration and management (ÖSTBERG et al., 2017).

Current research has primarily focused on decentralizing resources away from centralized cloud data centers to the edge of the network and making use of them for improving application performance. Typically, edge resources are configured in an ad hoc manner and an application or a collection of applications may privately make use of them. These resources are not publicly available, for example, like cloud resources. Additionally, edge resources are not evenly distributed but are sporadic in their geographic distribution. However, ad hoc, private, and sporadic edge deployments are less useful in transforming the global Internet (CHANG; SRIRAMA; BUYYA, 2016a). The benefits of using the edge should be equally accessible to both the developing and developed world for ensuring computational fairness and for connecting billions of devices to the Internet. However, there is minimal discourse on how edge deployments can be brought to bear in a global context – federating them across multiple geographic regions to create a global edge-based fabric that decentralizes data center computation.

Industrial marketing research forecasts that the market value of edge hardware components will reach $7,659 million by the year 2022, which indicates that more Edge-ready equipment such as routers, switches, IP gateway, or hubs will be available in the market (SHI et al., 2016). The edge experience faces challenges in defining standardization for software and hardware. First, edge-based equipment raises a question for the vendors as to what edge platform and related software packages should be included in their products. Second, edge-based software raises a question for the vendors regarding compatibility. Specifically, users may have devices in heterogeneous specification and processing units (e.g. x86, ARM etc.) in which the vendor may need to provide a version for each type of hardware. Moreover, developing and maintaining such an edge-based software can be extremely costly unless a corresponding common specification or standard for hardware exists.

Monitoring of edge resources will be a key to provide the edge advantages. For example, performance metrics will need to be monitored for implementing auto-scaling methods

to balance workloads on the edge. Existing monitoring systems for distributed systems either do not scale or are resource consuming (ABDERRAHIM et al., 2017). These are not suitable for large-scale resource-constrained edge deployments. Current mechanisms for auto scaling resources are limited to single-edge nodes and employ lightweight monitoring. However, scaling these mechanisms is challenging.

However, designing a proper monitoring system is a challenging task because Edge infrastructure differ significantly from traditional Cloud infrastructure (ABDERRAHIM et al., 2017). Three intrinsic characteristics differ in their development: First, while a traditional Cloud Computing basically depends on high performance servers and networks placed in very few sites, Edge Computing, especially Multi-Access Edge Computing infrastructure, is deployed in a significant number of sites and the distance that separates its resources can reach hundreds of kilometers affecting latency as well as bandwidth. Second, the infrastructure in Multi-Access Edge Computing is made up of several heterogeneous resources with different characteristics in terms of capacity, reliability and usability. Third, infrastructure resources based on Edge Computing, especially Multi-Access Edge Computing, can be highly dynamic and can permanently enter and exit the network according to service usage, failures, policies and maintenance operations.

1.2 Objectives and Research Challenges

Therefore, this work's main goal is to propose and evaluate an architecture compatible with ETSI Management and Orchestration software stack for managing virtualized functions in the Multi-Access Edge Computing deployed on low-cost devices and bare metal servers seamlessly, that offers a native unified monitoring approach from the hardware until the services.

In order to achieve such higher goals, some specific goals should be considered, namely:

- ❏ Investigate and review the state of the art on Network Functions Virtualization, machine virtualization concerning to management in Multi-Access Edge Computing;
- ❏ Investigate the concept of monitoring highly distributed systems, reviewing its general objectives, as well as obstacles present for systems in Multi-Access Edge Computing;
- ❏ Design the architecture using open-source frameworks and deploy it on top low-cost devices and bare-metal servers;
- ❏ Using an experimental approach, evaluate the architecture to highlight its features and capabilities.

1.3 Hypothesis

Once the previously presented objectives are established, the hypothesis is supported, it is possible to propose and evaluate an architecture compatible with ETSI Management and Orchestration software stack for managing Virtualized Everything Functions in the Multi-Access Edge Computing deployed on low-cost devices and bare metal servers seamlessly, that offers a native unified monitoring approach from the hardware until the services.

1.4 Contributions

The main contributions of this work in relation to the state of the art are: (1) an architecture that allows the continuous use of VxFs in devices present at the edge of the network; (2) an infrastructure for continuous monitoring of computing resources, in physical and virtualized environments, as well as the monitoring of specific applications that are at the edge of the network; (3) an experimental study of the feasibility of VxFs, using different virtualization platforms, on Multi-Access Edge Computing devices such as a Raspberry Pi 4; (4) an experimental study of the native unified monitoring and management of VxFs deployed in different infrastructures present at the edge of the network; (5) an experimental study of the complete monitoring capability of an environment capable of supporting VxFs, correlated metrics at the physical infrastructure level, virtualized environment and at the application level;

1.5 Dissertation Organization

This dissertation is organized into five chapters. Chapter 2 details and discusses the theoretical and practical aspects that clarify the various topics covered in the work objectives and their validity. Computing systems technologies are detailed, such as Machine Virtualization, Network Functions Virtualization, Internet of Things, Edge Computing and Monitoring Distributed Systems concepts.

Chapter 3 describes the dissertation proposal, as well as implementation details, and organizational structure of components and their respective functions.

Chapter 4 presents three experimental evaluations, highlighting each of the objectives, in addition to presenting an in-depth analysis of the contributions of this work in relation to previous proposals.

Chapter 5 discusses the conclusion reached in the construction and experimentation of the proposed solution. In addition, this chapter points out some projects and initiatives of productive or test scope that can be achieved by the proposed architecture.

Chapter 2

Background and Related Work

The IT infrastructure has been virtualized for years (MAVRIDIS; KARATZA, 2017). The network is the next step, as operators want to keep up with technological changes. Applications are demanding a lot of bandwidth, flexibility and speed of networks (ALAM et al., 2020; ZHOU et al., 2018). However, the expansion of networks to support peak traffic loads is beyond the reach of most operators, in addition to the high cost of maintenance and extensibility (MORLEY; WIDDICKS; HAZAS, 2018). At the same time, it is unrealistic to purchase specific hardware for a single highly volatile application, in the hope that it will work for 10 years or more, thus ensuring a long-term investment (PATTARANANTAKUL et al., 2016). Therefore, greater agility and control of the network and its essential functions are necessary. This virtualization involves programming using software control (KARAKUS; DURRESI, 2017). Following the recommendations of the European Telecommunications Standards Institute (ETSI) and the NFV Industry Standards Group (ISG) (GIUST; COSTA-PEREZ; REZNIK, 2017).

NFV offers service providers and operators the opportunity to reduce their capital costs Capital Expenditure (CAPEX), Operational Expenditure (OPEX) and network infrastructure costs, while speeding up the configuration and deployment of new network services (DEMIRCI; SAGIROGLU; DEMIRCI, 2021). This new, more flexible software-based network service environment allows service providers and operators to quickly activate new network services as needed for specific situations and customers, shortening the process from weeks or months to days or even minutes (POURGHASEMIAN et al., 2021). This business agility creates a significant competitive advantage, because it allows network operators to search for new markets and opportunities that were not economically viable, while using traditional network hardware and software (LI et al., 2021b).

The remainder of this chapter is structured as follows. The section 2.1 introduces the benefits of NFV and the market drivers that are enabling it to adapt. This chapter also focuses on building the basic knowledge of NFV by introducing the architectural structure and its components. The 2.2 section focuses on the key technology that makes NFV possible - virtualization. The purpose of this section is to gain a good understanding

of virtualization technologies and how they relate to NFV. The section 2.3 provides an overview of the Internet of Things paradigm, its concepts, principles and potential benefits. specifically, the section focuses on the main technologies of the Internet of Things and its emerging development protocols. Furthermore, with the rapid development of the Internet of Things, new problems were presented, proposing a new computing paradigm, Edge Computing. The section 2.4 focuses on the monitoring service as a key element for any management system responsible for operating a distributed infrastructure. And finally, the section 2.5 concisely presents each of the NFV-related research in Edge Computing, as well as monitoring tools, and their applications in IoT environments.

2.1 Network Functions Virtualization

NFV is based on this concept of server virtualization. Such virtualization goes beyond servers, expanding the scope to include network devices. NFV also allows for the creation of an ecosystem capable of managing, provisioning, monitoring and deploying virtualized network entities (RAY; KUMAR, 2021).

NFV can also be defined as a method and technology that allows you to replace physical network devices that perform specific network functions with one or more software programs that perform the same functions by running on generic computer hardware (SANCHEZ-AGUERO et al., 2021). An example is to replace a physical firewall device with a software-based virtual machine. This virtual machine provides firewall functions, runs the same operating system and has the same configurations, using non-dedicated, shared and generic hardware (MONIR; PAN, 2020). With NFV, network functions can be deployed on any generic hardware that offers virtualization for basic data processing, storage and transmission capabilities (BRUSCHI et al., 2017).

Network virtualization opens up new possibilities in how networks can be deployed and managed. Flexibility, agility, savings in operating costs and investment capital, combined with NFV scalability, opens the opportunity for new network architectures, technological innovations and vertical business (BARAKABITZE et al., 2020).

2.1.1 NFV Architecture

NFV allows software developed by third parties to run on generic shared hardware, creating multiple points of contact for management (VILALTA et al., 2019). In the world of NFV, the virtual implementation of network functions is known as the Virtualized Network Function (VNF). A VNF is designed to perform a certain network function, such as a router, a switch, a firewall, a network load balancer, etc (QI; SHEN; WANG, 2019). However, function virtualization is not intended exclusively for network functions, in fact, any software capable of being virtualized can be used. For example, we can highlight a machine learning application, a content management system, a system for

detecting images or even a messaging application, in short, any specific purpose software can be used. To this end Silva et al. (2019) expands the concept of VNF to include all types of functions by proposing the concept of Virtualized Everything Functions (VxF). In this way, the present work will use this concept, making no distinction between the type of function and its applicability.

Different providers can offer different VxFs and service providers can choose a combination of functions that best suit your needs (QI; SHEN; WANG, 2019). This freedom of choice creates the need for standardization in the way of managing them in the virtual environment. This standard structure must guarantee that the implanted VxFs are not obliged to specific hardware and be specially adapted for any environment capable of operating them (LAN et al., 2017). Consequently, suppliers must be offered a reference architecture that they can follow for consistency and uniformity in any VxF deployment and development methodologies.

Industry Specification Group of European Telecommunications Standards Institute (ISG ETSI) has developed more than 100 different specifications and reports for the virtualization of network functions (DAHMEN-LHUISSIER, 2021), with a focus on the management and orchestration of virtualized resources. From an architectural point of view, the NFV specifications describe and specify the requirements for virtualization, the functional components and their interfaces, as well as the protocols and APIs for those interfaces. Another set of specifications proposed by the ISG NFV is the structure and format of the artifact deployment and packaging models used by the NFV management and orchestration structure (GIUST; COSTA-PEREZ; REZNIK, 2017).

Based on these specifications, a high-level architectural structure was established, defining distinct areas of focus, as shown in Figure 1.

This architectural structure forms the basis for standardization and development work and is commonly referred to as the ETSI NFV framework (BENEDICTIS; LIOY, 2017). The ETSI ISG categorized these functions into three high-level blocks, namely: infrastructure block, virtualized function block and management block. In the definition of ETSI, the formal names of these blocks are defined as:

- **Network Functions Virtualization Infrastructure (NFVI)**: This block represents the hardware to host the virtual machines, the software to make virtualization possible and the virtualized resources (TUSA et al., 2018).

- **Virtualized Everything Functions**: The VxF block uses the virtual machines offered by NFVI and builds on them by adding specialized software for building virtualized functions (SILVA et al., 2019).

- **Management and Orchestration (MANO)**: MANO is defined as a block that interacts with the NFVI and VxF blocks. The framework delegates to the MANO

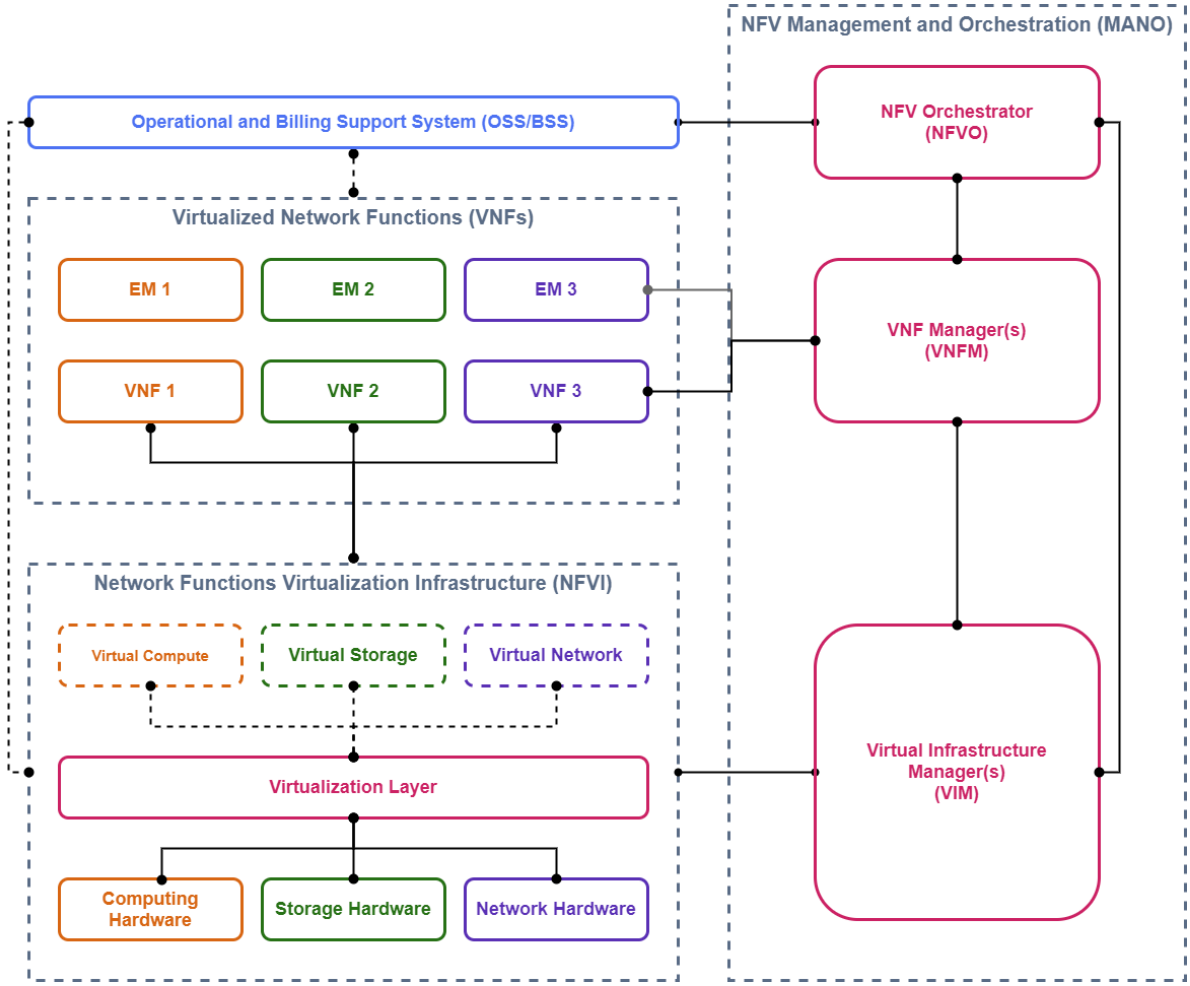

Figure 1 – ETSI NFV framework.

Adapted from: Mechtri et al. (2017)

layer the management of all resources of the infrastructure layer; in addition, this layer manages resources and allocates VxFs (GARCÍA-ROIS et al., 2021).

To implement the network service, VxFs can be deployed as stand-alone entities or as a combination of several VxFs (QI; SHEN; WANG, 2019). Since there is no need for dedicated or custom hardware designed to run these VxFs, a general purpose device with generic hardware features such as processor, storage, memory and network interfaces can be used to run these VxFs (LAN et al., 2017). Virtualization technologies can be used to share hardware between multiple VxFs (AMRI; MEDDEB, 2021). These technologies can be hypervisor based or container based (BHARDWAJ; KRISHNA, 2021).

As MANO is intended to have a full view of the entities and is responsible for managing them, it is fully aware of the use, operational status and usage statistics of each block in the architecture (GARCÍA-ROIS et al., 2021). This makes MANO the most suitable interface for operating systems and the central point for collecting network data.

The MANO management component is actually defined as a combination of three functional components: the Virtualized Infrastructure Manager (VIM), the Virtualized Network Function Manager (VNFM) and the NFV Orchestrator (NFVO). For a better understanding, these functional components are grouped into layers, where each layer deals with a particular aspect of the NFV implementation (MIJUMBI et al., 2016).

VxFs require availability of virtual hardware, which simulates by software resources running on physical hardware. In the ETSI NFV framework, this is possible thanks to the infrastructure component (NFVI). This infrastructure component comprises physical hardware resources, the virtualization layer and virtual resources (TUSA et al., 2018). This functional component can be extended and scaled from a single physical host to multiple interconnected local devices.

The virtualization layer is the functional block that is part of NFVI. It interacts directly with the host machine's hardware devices, offering a simple and abstract view of computing, storage and networking the VxFs. In short, it is the virtualization layer that decouples VxFs from the underlying hardware of the host machine, providing them with a computational resource management interface (COMPASTIé et al., 2020).

The Virtualized Infrastructure Manager is part of MANO and has the responsibility to manage the computing, storage and network hardware, which is under the virtualization layer, and which will therefore be virtualized (MIJUMBI et al., 2016). Since VIM directly manages hardware resources, it has a complete list of these resources and their operational states, as well as the ability to monitor the performance of all underlying hardware. VIM also manages the virtualization layer and controls the influence of the virtualization layer on the hardware (CALLEGATI et al., 2017). VIM is, therefore, responsible for controlling NFVI resources and works with other functional management blocks to determine the requirements necessary for the implementation of VxFs.

The VxF layer is where virtualized functions are deployed. This layer is composed of the VxFs block and the functional block that manages it, called VNF-Manager . The VxFs block is defined as a combination of VxF and Element Management (EM) blocks.

Element Management is another functional block defined in the ETSI structure and is intended to assist in the implementation of the management functions of one or more VxF. The scope of EM management is analogous to the traditional Element Management System (EMS), which serves as an interaction layer between the network management system and devices that perform network functions (GEDIA; PERIGO, 2018). Element Management interacts with VxFs using proprietary methods while employing open standards to communicate with VNFM.

When switching from physical to virtual devices, network operators may not want to renew the management tools and applications that are already in place and Operations Support System and Business Support System (OSS/BSS) (PEREIRA; KARIA, 2018). The framework does not require a change in these tools as part of the transformation

to NFV. In fact, it allows you to continue to manage the operational and commercial aspects of the network, even if they are replaced by VxFs. One way that providers can follow is to improve and develop tools and systems capable of using the functional blocks of architecture management and using its benefits such as elasticity, agility, etc. The solution that the ETSI framework offers is to use the functional block, NFV Orchestrator. It extends the current OSS/BSS and manages the operational and deployment aspects of NFVI and VxF (KUKLINSKI; TOMASZEWSKI, 2018).

NFVO has a critical role in the ETSI NFV architecture, observing the implementation of end-to-end services, analyzing the general panorama and communicating the information necessary for the functioning of VIM and VNFM. NFVO also works with VIM(s) and has a complete view of the resources they are under their control (MONTERO et al., 2020). As previously indicated, there may be several NFVIs and their respective VIMs, each of which has a view of the resources under its supervision.

2.1.2 Message Flow in NFV architecture

For a better explanation, the following Figure 2 shows how the main components of the ESTI architecture collectively interact for the implementation of a simple service.

The following steps depict this process:

1. The full view of the end-of-end topology is visible to the NFVO.
2. The NFVO instantiates the required VxFs and communicate this to the VNFM.
3. VNFM determines the number of VMs needed as well as the resources that each of these will need and reverts back to NFVO with this requirement to be able to fulfill the VNF creation.
4. Because NFVO has information about the hardware resources, it validates if there are enough resources available for the VMs to be created. The NFVO now needs to initiate a request to have these VMs created.
5. NFVO sends request to VIM to create the VMs and allocate the necessary resources to those VMs.
6. VIM asks the virtualization layer to create these VMs.
7. Once the VMs are successfully created, VIM acknowledges this back to NFVO.
8. NFVO notifies VNFM that the VMs it needs are available to bring up the VxFs.
9. The block VxFs now configures the VxFs with any specific parameters.
10. Upon successful configuration of the VxFs, VNFM communicates to NFVO that the VxFs are ready, configured, and available to use.

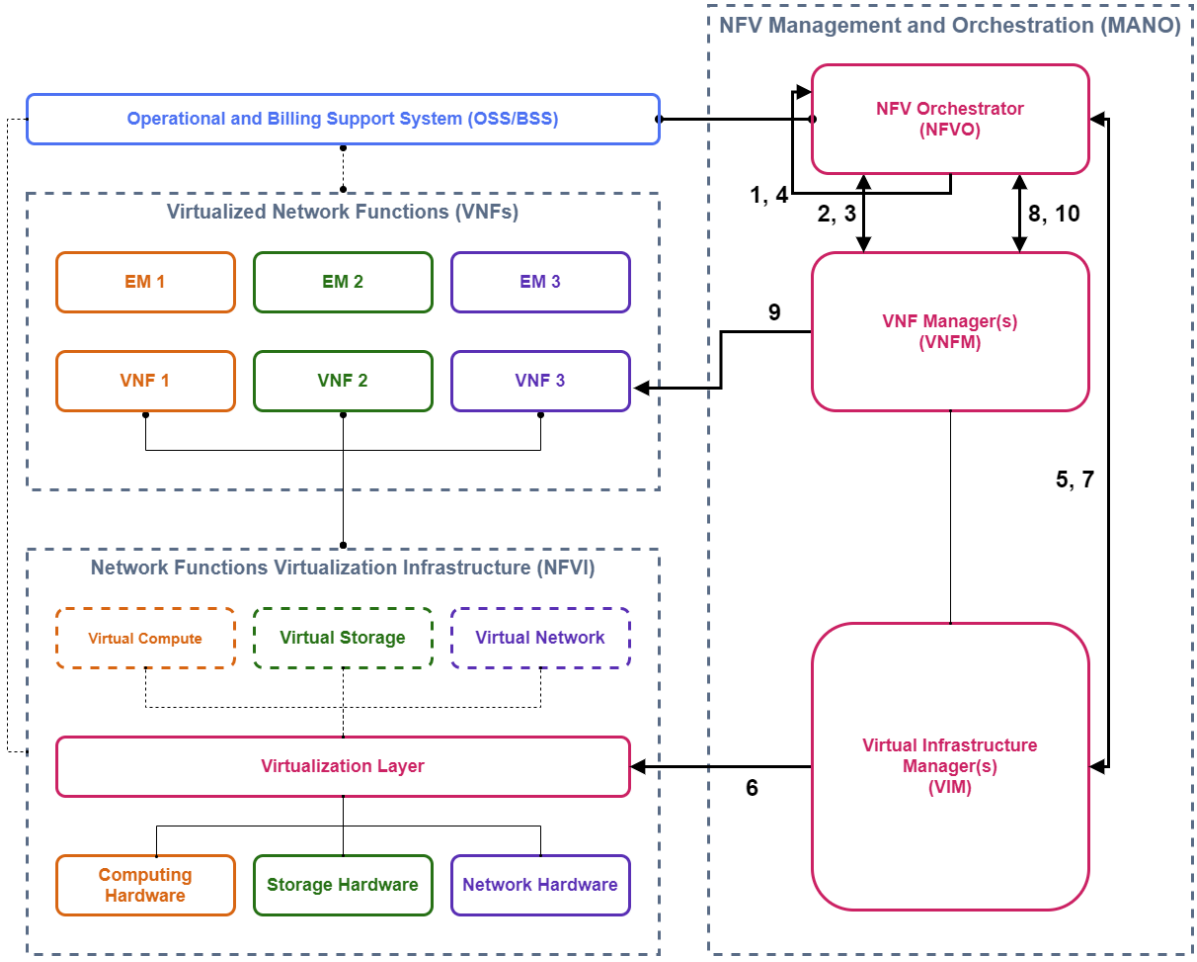

Figure 2 – Message Flow in the ETSI NFV architecture.

Adapted from: Chayapathi (2017)

2.1.3 Benefits of NFV

Network Functions Virtualization offers a framework to completely transform the way networks are designed, deployed, managed and operated, while offering many layers of improvement and efficiency in all of them (CHAYAPATHI, 2017). These are some of the benefits that NFV offers:

1. **Faster service life cycle**: New network services or features can now be deployed more quickly, in an on-demand and on-need basis, providing benefits for end users as well as the network providers.

2. **Hardware flexibility**: Because NFV uses generic hardware, network operators have the freedom to choose and build the hardware in the most efficient way to suit their needs and requirements.

3. **Scalability and elasticity**: NFV allows capability changes by offering a means to

expand and shrink the resources used by the VxFs. For instance, if any of the VxFs requires additional CPU, storage, or bandwidth, it can be requested from the VIM and allocated to the VxF from the hardware pool.

4. **Rapid development and vendor independence**: New solutions and features can be put into production rapidly, without waiting for the existing deployed vendor to develop and support them.

5. **Validation of new solutions**: With NFV, building and managing test-setups can be dynamic and thus scaled and changed to meet the test and validation scenarios.

6. **Leveraging existing tools**: The NFV can use the same infrastructure already acquired, reusing and taking advantage of the deployment and management tools.

7. **Barrier of entry**: NFV opens the door for new suppliers and providers to enter the market, bringing innovations and offering implementations of network functions with lower prices and better performance.

2.1.4 Challenges of NFV

Virtual machines or containers need to communicate with each other, either within the same host or with physical or virtual devices outside of hosts. In any case, they need network interface cards for this communication. One is through virtualized interfaces and the other is using dedicated physical interfaces to a virtual machine. Physical interfaces, of course, are only useful for communicating with devices outside the host. However, as the number of interfaces required for virtual machines is often much greater than the number of physical interfaces available in the infrastructure, interface-level virtualization is not avoidable. The NFVI layer takes care of this, along with other virtualization functions it performs, including virtual network interface cards (vNICs) that are made available to the virtual machines. For virtual machines, these vNICs are treated as real physical interfaces and use the interfaces to send and receive packets outside the virtual machine. However, to make this virtual interface scalable, able to offer some switching capabilities and perform packet switching without significant performance degradation, a number of researches and different approaches have been used to offer this service (LIU et al., 2017).

Virtualization adds another layer of overhead to computing environments by imposing an achievable throughput barrier. VxFs that are often in the data path need to be highly optimized for the performance of your packages. Furthermore, the efficiency of CPU, storage and memory resource utilization methods plays an important role in the overall performance of the virtual environment. Thus, several performance techniques have been researched in order to maximize the transfer rate by decreasing the latency of VxFs deployed along the network (LUIZELLI; RAZ; SA'AR, 2018).

2.1.5 Eclipse fog05

Eclipse fog05 is an open source project, acting as a Virtualized Infrastructure Manager capable of provisioning and managing compute, storage, communication and I/O resources available anywhere on the network (VALANTASIS et al., 2021). Eclipse fog05 addresses highly heterogeneous systems, even those with extremely limited resource nodes.

Eclipse fog05 allows the end-to-end management of compute, storage, networking and I/O fabric in the Edge and Fog environment. Instead of relying on a centralized architecture (like cloud-management-systems) it is based on a decentralized architecture. Here are some features of Eclipse fog05:

Eclipse fog05 allows users to manage and deploy different types of applications, packaged as containers, VMs, binaries and so on. This possibility is achieved thanks to Eclipse fog05 plugin architecture. Here are some features of Eclipse fog05:

- ❑ **Unified abstractions**: Eclipse fog05 provides an unified API for the management of the virtualization infrastructure.
- ❑ **Operating system plugins**: Eclipse fog05 can run on different operating systems, it just need the right OS Plugin.
- ❑ **Networking plugins**: Eclipse fog05 can manage networking fabrics for which a Networking plugin is present.
- ❑ **Heterogeneous applications**: Support of heterogeneous runtimes, hypervisors and networking. Deploy heterogeneous applications composed by VMs, containers, ROS2, native applications.
- ❑ **Runtime plugins**: Eclipse fog05 can manage and open-ended set of hypervisors and container technologies for which a Runtime Plugin was implemented.
- ❑ **Single descriptor**: Eclipse fog05 allows you to define your application in a single descriptor. No matters if it is composed by heterogeneous components.
- ❑ **Lightweight**: Eclipse fog05 is designed to be deployed from big servers to micro-controllers.
- ❑ **Decentralized state**: Because Eclipse fog05 uses Zenoh for location-transparency state access and management. It can be deployed on resource constrained devices and leverage other nodes for state management.
- ❑ **Modular**: Eclipse fog05 is built with a plugin architecture and his components can be deployed separately.

Eclipse fog05 is divided in two modules: Fog Infrastructure Manager (FIM) and Fog Orchestration Engine (FOrcE). Both modules provides different abstractions an APIs.

2.1.5.1 Fog Infrastructure Manager

This module virtualizes the hardware infrastructure, such as computing, communication, storage and I/O resources. It provides the main resources for managing the infrastructure and the application deployed on it. The FIM is designed to be a distributed system, with no controller nodes. The state of the system is managed in a local and transparent manner, taking advantage of Zenoh for distribution and sharing of data across the system.

In Eclipse fog05 a resource is any kind of asset that can be assigned either in a shared or exclusive manner. The different kinds of resources supported by fog05 are computational resources, such as, CPU, FPGA, GPU, storage resources, such as, RAM, block storage, object storage, networking resources, such as, 802.11 devices, tunnels and I/O resources, such as, COM, CAN, GPIO and I2C.

An Eclipse fog05 node represents a locus of resources. Examples of fog05 nodes are: server, an Raspberry Pi, an embedded board. In order to expose its resources in the system each node has either a running fog05 agent or a device plugin.

In Eclipse fog05, a network represents a collection of deployable components. These networks are usually virtualized and span different nodes, even from end to end, and can also be connected to physical networks.

An Eclipse fog05 router is a networking element able to interconnect two or more networks. It can also connect a network to the physical networking infrastructure.

A port in Eclipse fog05 is used to represent a network connection between a deployable unit and a network. Ports are also present in router to connect the router to the different networks, a port can have a floating IP assigned to it.

In Eclipse fog05 a Fog Deployment Unit (FDU), is an indivisible unit of deployment, such as, a binary executable, a Unikernel, a container or a virtual machine. An FDU requires a certain kind of resources as a precondition to its execution. The life-cycle of an FDU is defined by a Finite State Machine (FSM) [1].

2.1.5.2 Components

Eclipse fog05 is composed of the following components. The agent is the component responsible for managing a node and disseminating the node's resources and functionality to the entire system. It also provides an entry point for managing and monitoring the node, as well as managing the FDUs running on the node.

FIM is designed to take advantage of plug-ins for most of their functionality, in Eclipse fog05 different types of plug-ins can be used. The operating system plug-in is the one used to abstract the underlying operating system and provide primitives for the agent and

[1] A state machine is a model of behavior. It consists of a finite number of states. Based on the current state and a given input, the machine performs state transitions and produces or not outputs.

other plug-ins. Operating system plug-ins need to implement an interface to be recognized by fog05.

The Network Manager plug-in is used to create all network-related components, such as routers and ports, and communication networks themselves, providing an implementation for each of these components. This plug-in also provides some functionality for the agent and other plug-ins. Network Manager plug-ins need to implement an interface to be recognized by fog05.

The Runtime Plugin implements the FDU lifecycle management, strictly following the state machine defined for the FDU and providing unique identifiers for managing and monitoring the status of the running FDUs. Runtime plugin need to implement an interface to be recognized by fog05. In Eclipse fog05, the following Runtime plugins are implemented: KVM Virtual Machine, Linux Containers (LXD), native applications (Linux and Windows).

The Device Plugin is the one that abstracts the previous three, and is designed to allow the deployment and discovery of microcontrollers and IoT cards that do not have an operating system. The device plugin must provide a subset of the functionality provided by the operating system, network manager and runtime plugin. It will also be connected and managed by an agent running on a different node that looks like any other node in the system.

2.1.5.3 Deployments

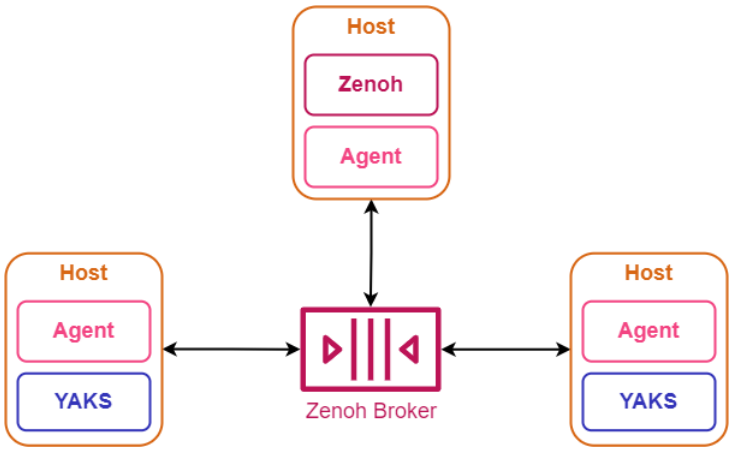

Figure 3 – Eclipse fog05 deployed as a distributed system.

Adapted from: Corsaro e Baldoni (2018)

All components of Eclipse fog05 are designed to be deployed in a distributed manner as shown in Figure 3. In particular, the minimum set to run a fog05 node is to have an agent and a set of plug-ins running on it, while in the case of micro-controllers the

minimum set consists only of the device plug-in. Other components, such as Zenoh [2] routers and YAKS [3], can be deployed on a fog05 node or on different machines. This combine deployment makes it possible to reduce the footprint and optimize the use of infrastructure software resources.

2.1.5.4 Fog Orchestration Engine

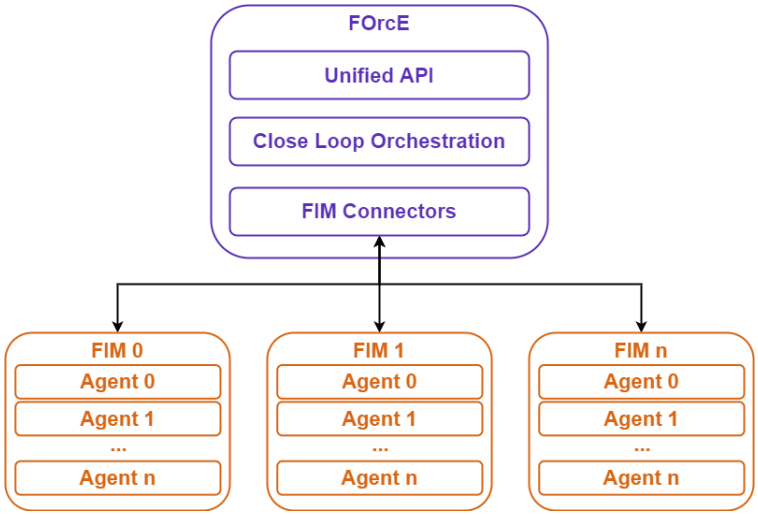

Figure 4 – Fog Orchestration Engine Architecture.

Adapted from: Corsaro e Baldoni (2018)

The Fog Orchestration Engine is responsible for end-to-end orchestration and management of applications and services in the Fog infrastructure. It enforces the restrictions defined by the application / service and provides a component allocation algorithm. FOrcE was designed to work in a decentralized way, taking advantage of Zenoh for distribution and storage of the state of the system. The FOrcE architecture is illustrated in Figure 4:

The FOrcE architecture consists of: A unified API capable of covering the components of Eclipse fog05. An Orchestration Engine that monitors, schedules and provides a replication-based auto-correction mechanism for deployed applications. A set of connectors to communicate with different FIM deployments. Users interact with FOrcE through fosctl, a command line utility that uses the FOrcE REST interface for interaction.

[2] Eclipse Zenoh is a traditional publish-subscribe platform with geo-distributed stores, queries and calculations that unify data in motion, data in use and data at rest.

[3] YAKS (Yet Another Key-value Store) is a framework that provides a unified view similar to a key-value store of distributed and heterogeneous data stores.

2.2 Virtualization Concepts

Virtualization technologies form the primary building block for the NFV infrastructure. Therefore, knowledge of virtualization is important in order to achieve an in-depth understanding of NFV deployment and implementation (BEHRAVESH; CORONADO; RIGGIO, 2019).

Virtualization is the technology used to run multiple operating systems or applications on top of a single physical infrastructure by providing each of them an abstract view of the hardware (BEHRAVESH; CORONADO; RIGGIO, 2019). It enables these applications or OSs to run in isolation while sharing the same hardware resources.

Software on network devices has historically been proprietary and customized. These devices use dedicated CPUs for processing and routing network traffic and specialized memory for address lookup (LAN et al., 2017). These packet processing CPUs are highly customized to implement network functions, such as routing, classification, queuing and access control, and are often implemented as Application Specific Integrated Circuits (ASIC) on traditional network devices.

Network function virtualization extends the idea of server virtualization to network devices designed to perform specific functions on the network. It is the success of server virtualization that has attracted network operators to look at NFV and, consequently, has led equipment suppliers and manufacturers to break away from a single, personalized device, allowing their network systems to run in a virtualized environment (HERRERA; BOTERO, 2016).

2.2.1 Virtualization techniques

Running multiple operating systems in parallel (each representing an application or set of applications that would run on an independent system) required collaboration between those operating systems or the introduction of a mediator (virtualization layer) to allow hardware sharing (BEHRAVESH; CORONADO; RIGGIO, 2019). Otherwise, when these operating systems or applications wish to access the hardware simultaneously, it could cause disorder and malfunction for each party (TIBURSKI et al., 2021). Virtualization provided a solution to make this possible by separating the hardware and the operating system and adding the virtualization layer as an interface between them.

There are four privilege levels defined for interacting with the hardware. The lowest privilege levels have a higher priority and run closer to the hardware. Generally speaking, applications run at privilege level 3, while device drivers are run using privilege levels 1 and 2 (CHIRAMMAL; MUKHEDKAR; VETTATHU, 2016). The operating system must be at privilege level 0 so that it can interact directly with the hardware, as shown in Figure 5.

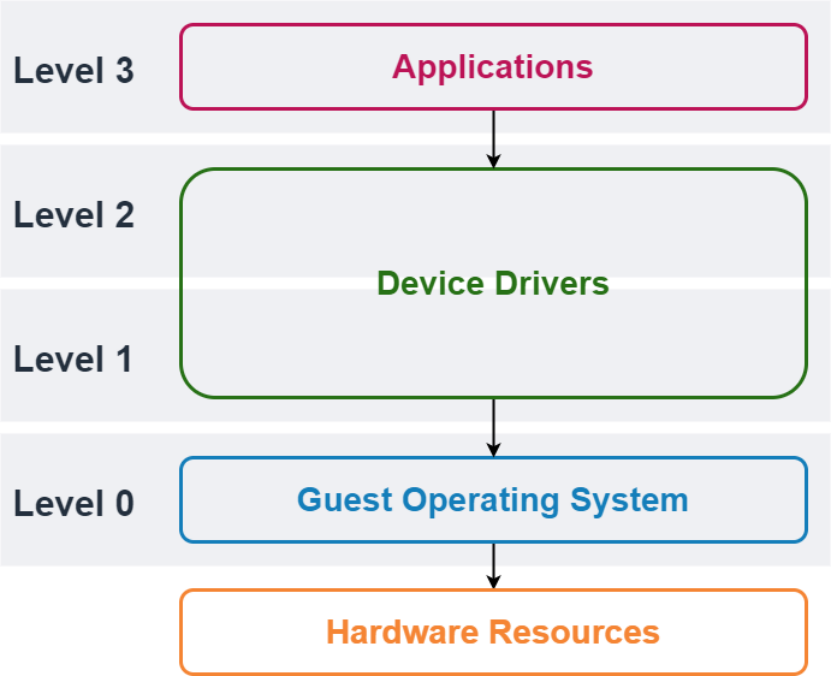

Figure 5 – Privilege levels for interaction to the hardware.

Adapted from: Chirammal, Mukhedkar e Vettathu (2016)

2.2.1.1 Full virtualization

In the full virtualization technique, the operating system is moved to the upper layer and the virtualization layer now occupies Layer 0, interacting with the hardware as shown in Figure 6. Therefore, the operating system instructions for the hardware are now translated by the virtualization layer and, in turn, sent to the hardware. Full virtualization, therefore, separates the hardware and the operating system (BABU et al., 2014). Some examples of this virtualization technique are VMware's ESXi, Linux KVM/Qemu, etc.

2.2.1.2 Paravirtualization

Full virtualization brings with it a cost associated with the translation of data and commands from the operating system to the virtualization layer and then to the hardware. For some applications, this delay can be critical, resulting in inefficiency. To resolve this situation, the operating system may be allowed to interact directly with the hardware for some functional calls, while continuing to work through the virtualization layer for other needs. This virtualization technique is called paravirtualization (KIYANOVSKI; TSAFRIR, 2017). In this technique, the virtualization layer works along the guest operating systems at the same privilege level (layer 0), as shown in Figure 7. An example of this technique for virtualization is Xenserver.

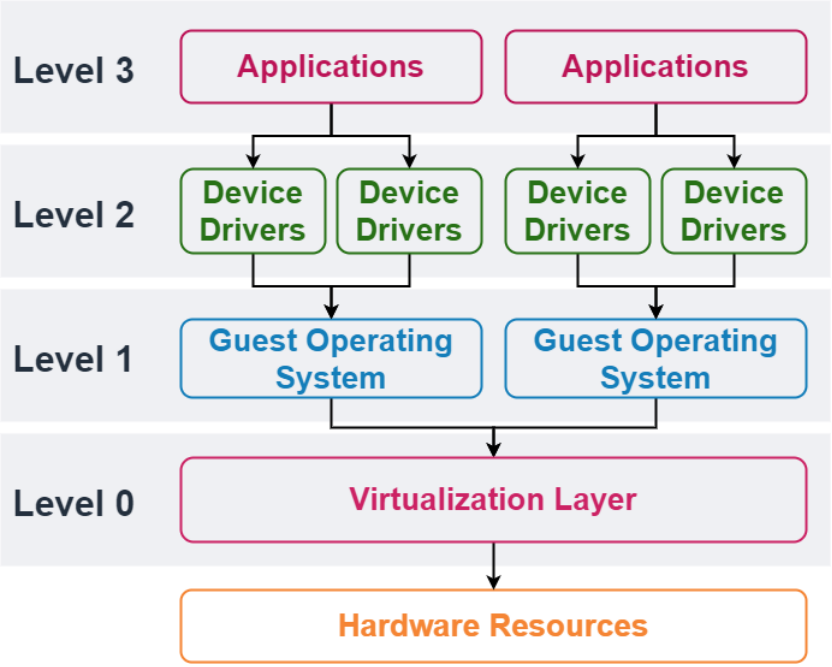

Figure 6 – Full virtualization.

Adapted from: Chirammal, Mukhedkar e Vettathu (2016)

2.2.1.3 Hardware-Assisted virtualization

Hardware Assisted Virtualization is a special type of full virtualization, in which the native hardware provides support for enabling virtualization. Modern day processors support native virtualization, allowing the instructions from multiple VMs to be executed directly on the shared CPU (ASVIJA; ESWARI; BIJOY, 2019). The OS and the virtualization layer can take advantage of hardware-assisted function calls to provide virtualization, as shown in Figure 8. Many operating systems and virtualization vendors are using this feature to provide improved performance and processing.

2.2.1.4 OS-Level virtualization

In OS-level virtualization, the operating system is modified to support these applications in its own workspace. This virtualization approach relies on OS facilities that partition the physical machine resources, creating multiple isolated user-space instances on top of a single host kernel (BESERRA et al., 2017). It ensures the separation between the use of the application's resources, making it look like they are still running on independent standalone servers. Figure 9 shows a representation of virtualization at the operating system level. Examples of this virtualization technique are Linux and Docker containers.

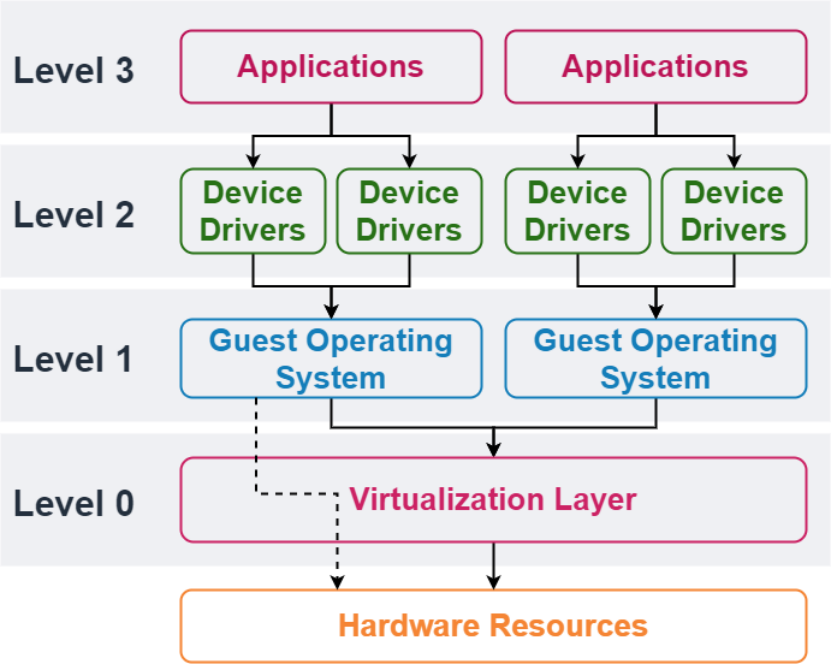

Figure 7 – Paravirtualization.

Adapted from: Chirammal, Mukhedkar e Vettathu (2016)

2.2.2 Virtual machine

Virtualization provides an isolated virtual hardware environment for an operating system or application to run upon. The resulting environment is usually referred to as a Virtual Machine (VM) (ZHANG et al., 2018). Virtual machines have three main components: host operating system, hypervisor or virtual machine manager and guest operating system.

The host perating system (host OS) is the operating system that is running directly on the hardware. The host OS must have support for virtualization and the right set of applications installed. The host OS has the visibility of the full set of hardware resources that are available to be allocated to the virtual machines.

Originally, the term Virtual Machine Manager (VMM) was used to specify the management platform for VMs, but in the early 1970s the term hypervisor was coined by IBM engineers, since this software was run on top of the operating system that he was called a supervisor (KHOSHKHOLGHI et al., 2017). A hypervisor allows the creation of the virtual machine, allocating resources for that virtual machine, modifying the parameters of the virtual machine and excluding the virtual machine (Li et al., 2017).

The operating system running on this virtual server is called a guest operating system. This operating system can be of any type, but it must be compatible with the hardware features offered by the hypervisor. The guest operating system does not need any modifications to run on the virtual machine and views the virtual machine as if it were running

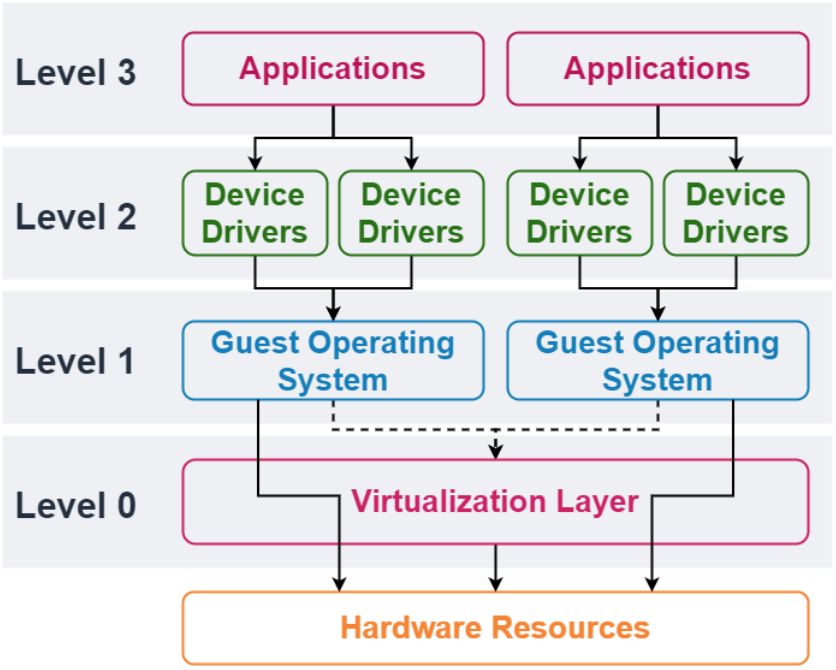

Figure 8 – Hardware-assisted virtualization.

Adapted from: Chirammal, Mukhedkar e Vettathu (2016)

on a real hardware system. Therefore, it has no visibility into the true hardware resources or knowledge of any other virtual machine that may be present by another instance of the hypervisor. This creates the impression that the guest operating system is communicating directly with the hardware entities.

2.2.3 Kernel-based Virtual Machine

KVM represents the latest generation of open source virtualization (BABU et al., 2014). The aim of the project is to create a modern hypervisor that builds on the experience of previous generations of technologies and takes advantage of the modern hardware available today (VT-x, AMD-V and so on).

KVM simply turns the Linux kernel into a hypervisor when you install the KVM kernel module. However, since the standard Linux kernel is the hypervisor, it benefits from changes made to the standard kernel (memory support, scheduler and so on). For input and output emulations, the KVM uses a user area software, QEMU; this is a user program that does hardware emulation.

QEMU emulates the processor and a long list of peripheral devices, such as disk, network, VGA, PCI, USB, serial / parallel ports, and so on, to build a complete piece of virtual hardware on which the guest operating system can be installed (DJORDJEVIC et al., 2021). This emulation is provided by KVM.

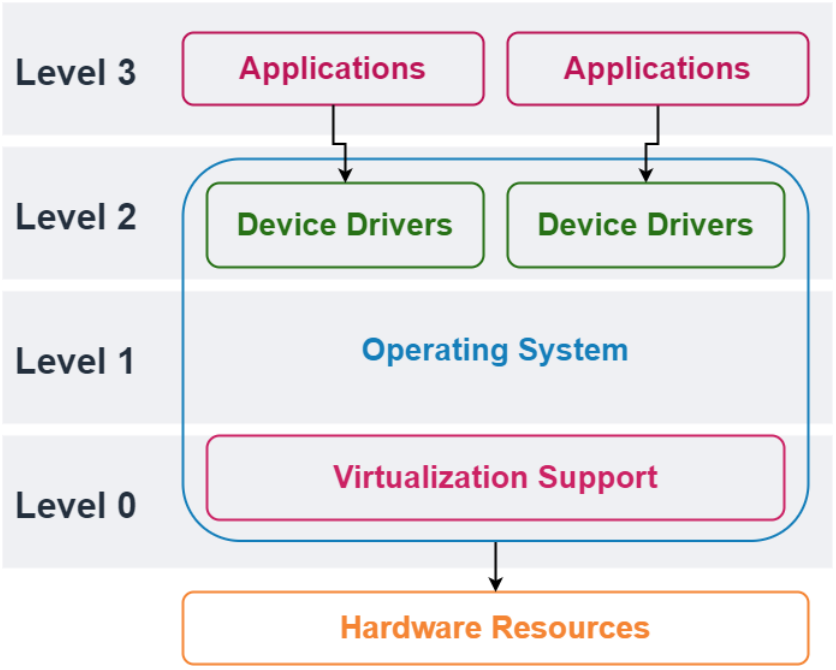

Figure 9 – OS-Level virtualization.

Adapted from: Chirammal, Mukhedkar e Vettathu (2016)

2.2.3.1 The internal workings of libvirt, QEMU, and KVM

The interaction of libvirt, QEMU and KVM are the most important pieces in the Linux virtualization puzzle, as each has a specific role to play.

When working with KVM, the main Application Programming Interface (API) is called libvirt. But libvirt has other features - it is also a daemon and a management tool for different hypervisors, such as Xen, LXC, VirtualBox and Mircrosoft Hyper-V (DAKIC et al., 2020). One of the most common tools used to interface with libvirt is called virt-manager, a graphical Gnome-based [4] utility that you can use to manage various aspects of your local and remote hypervisors. Libvirt's Command-line Interface (CLI) is called virsh.

The purpose of the libvirt library is to provide a common and stable layer for managing virtual machines running on a hypervisor. In short, as a management layer, it is responsible for providing the API that performs management tasks, such as virtual machine provisioning, creation, modification, monitoring, control, migration and so on. On Linux, the libvirt process is daemonized and is called libvirtd (YUN et al., 2020). As with any other daemon process, libvirtd provides services to its customers through the process of requests and responses.

[4] GNOME is a free and open-source desktop environment for Unix-like operating systems. GNOME 3 is the default desktop environment on many major Linux distributions including Fedora, Debian, Ubuntu, CentOS and others.

2.2.3.2 QEMU

QEMU is a free piece of software and mainly licensed under GNU's General Public License (GPL) (DJORDJEVIC et al., 2021). QEMU is a generic and open source machine emulator and virtualizer. When used as a machine emulator, QEMU can run OSes and programs made for one machine (such as an ARM board) on a different machine (such x86_64). QEMU can act as an emulator or virtualizer.

When QEMU operates as an emulator, it is capable of running operating systems and programs made for a different type of machine than the original. It is possible only using binary translation methods. In this mode, QEMU emulates CPUs using dynamic binary translation techniques and provides a set of device models (FEDERICO, 2018). Thus, it is able to run different unmodified guest operating systems with different architectures. Binary translation is required here because the guest code to run on the host CPU. The binary translator that does this job is known as the Tiny Code Generator (TCG); is a Just-In-Time (JIT) compiler (LI et al., 2021a). It transforms binary code written for a given processor into another form of binary code.

When QEMU operates as a virtualizer, QEMU executes guest code directly on the host's CPU, achieving native performance. For example, when working with Xen / KVM hypervisors, QEMU can operate in this mode. If KVM is the underlying hypervisor, QEMU can virtualize embedded guests, such as Power PC, S390, x86 and so on (CHIRAMMAL; MUKHEDKAR; VETTATHU, 2016). In short, QEMU is able to function without KVM using the previously mentioned binary translation method. This execution will be slower compared to KVM-enabled hardware-accelerated virtualization (MATHEW; JOSE, 2017). In any case, whether as a virtualizer or an emulator, QEMU doesn't just emulate the processor; it also emulates different peripherals, such as disks, networks, VGA, PCI, serial and parallel ports, USB and so on (DJORDJEVIC et al., 2021).

2.2.4 Containerization

The virtual machines described in the previous section provide a very isolated and independent environment. But there is still a performance cost and resources associated with them due to the overhead required to emulate virtual hardware (BABU et al., 2014). In some cases, however, the isolation level of the virtual machine is not really necessary and a compromise with a lighter level of isolation may be acceptable. This can be achieved using a simpler virtualization technique called container-based virtualization (BESERRA et al., 2017).

This type of virtualization uses an operating system-level virtualization technique and provides a closed, restricted and segregated environment for applications within the operating system. This environment is called a container and applications can run in-

dependently within it (LI et al., 2017). Container-based virtualization has its roots in the evolution of UNIX/Linux kernels. The capability to implement this virtualization is a direct result of the efforts made to provide kernel level support for isolation between applications (BESERRA et al., 2017).

2.2.5 Linux Containers

Nowadays, the implementation of applications inside some type of Linux container is a widely adopted practice, mainly due to the evolution of the tools and the ease of use that it presents. Although Linux containers, or virtualization at the operating system level, in one form or another, have been around for more than a decade (AULIYA; NURDINSYAH; WULANDARI, 2019), it took some time for the technology to mature and enter mainstream. One reason for this is the fact that hypervisor-based technologies, such as KVM and Xen, have been able to solve most limitations of the Linux kernel during this period and the overhead it presented were not considered a problem (RAHO et al., 2015). However, with the advent of kernel namespaces and control groups (cgroups), the notion of light virtualization has become possible through the use of containers.

2.2.5.1 The OS kernel and its early limitations

The current state of Linux containers is a direct result of the problems that early OS designers were trying to solve - memory management, I/O and scheduling processes in the most efficient way (KOVACS, 2017).

In the past, only a single process could be scheduled to work, wasting CPU cycles if blocked in an I/O operation. The solution to this problem is to develop better CPU schedulers, so that more work can be allocated fairly for maximum CPU usage. Modern schedulers, such as Completely Fair Scheduler (CFS) on Linux are able to allocate a reasonable amount of time for each process, and be able to give higher or lower priority to a process and its subprocesses, increase the level of granularity or control that can be achieved.

Likewise, before the advent of virtual memory, several processes would allocate memory from a shared pool of physical memory. Virtual memory provided some form of memory isolation per process, in the sense that the processes would have their own address space, and extend the available memory through a context switch, but there was still no good way to limit the amount of memory that each process and its children could use.

To further complicate matters, running different workloads on the same physical server generally resulted in a negative impact on all services running. A memory leak or pause in the kernel can cause an application to shut down the entire operating system. For example, a web server that is mainly limited by memory and a database service that has

a high demand for I/O running together has become problematic. In an effort to avoid such scenarios, system administrators would separate the various applications between a pool of servers, leaving some machines underutilized, especially at certain times of the day, when there was not much work to be done. This is a similar problem, as a single running process blocked by an I/O operation is a waste of CPU and memory resources.

The solution to these problems is to use hypervisor-based virtualization, containers, or a combination of both (WATADA et al., 2019). The hypervisor is the main element of the operating system responsible for managing the life cycle of virtual machines, and has existed since the first mainframes in the late 1960s. Most modern virtualization implementations, such as Xen and KVM, can trace their origins back to that time (CHIRAMMAL; MUKHEDKAR; VETTATHU, 2016). The main reason for the widespread adoption of these virtualization technologies around 2005 was the need to better control and use the growing clusters of computing resources. The security inherited from having an extra layer between the virtual machine and the host operating system was a good starting point for those concerned with security, although, as with any other newly adopted technology, there were security incidents (WATADA et al., 2019).

Hypervisors provide the following benefits, in the context of the problems described above:

- ❏ Ability to run different operating systems on the same physical server.
- ❏ Granular control over resource allocation.
- ❏ Separate network stack and ability to control traffic per virtual machine.
- ❏ Simplifying data center management.
- ❏ Better use of available server resources.

Probably the main reason against using any type of virtualization technology today is the inherited overhead of using multiple kernels on the same operating system (Li et al., 2017). It would be much better, in terms of complexity, if the host operating system could provide this level of isolation, without the need for hardware extensions on the CPU, or the use of emulation software such as QEMU, or even kernel modules, such as KVM (CHIRAMMAL; MUKHEDKAR; VETTATHU, 2016).

Over the past decade, several improvements to the Linux kernel have been made to allow similar functionality, but with less overhead - most notably the kernel namespaces and cgroups. One of the first notable technologies to leverage these changes was the LXC (Li et al., 2017). The main benefits of using LXC include: Less overhead and complexity than running a hypervisor, less startup time and native kernel support. It is important to mention that containers are not inherently as secure as having a hypervisor between the virtual machine and the host operating system (WATADA et al., 2019). However, in

recent years, great progress has been made to narrow this gap using Mandatory Access Control (MAC) technologies.

2.2.6 Comparison between Virtual Machines and Containers

One of the most notable differences in containerization is that it does not create a new (virtual) machine within the host operating system, emulating the presence of real hardware. Instead, containers provide a means of separating the use of system resources and defining restrictions on their use. In terms of the virtual environment, the virtual machine and containers are often used interchangeably, but it is worth mentioning some of the important differences between the two methodologies. Table 2 summarizes these main differences:

Table 1 – Comparison between Virtual Machines and Containers.

Virtual Machines	**Containers**
Allocate chunks of hardware resources	Restrict usage of hardware resources
Needs an operating system inside	Run standalone application or using a different operating system
Provides high level of isolation and security of kernel	Doesn't provide pure isolation, and affect entire host.
Usually slower since there is middle layer	Native performance since there is middle layer.

2.2.7 Virtualization and Network Function Virtualization

NFV is the result of adapting server virtualization to the network. The virtualization capability is what constitutes the core of the NFV architecture (BEHRAVESH; CORONADO; RIGGIO, 2019). Looking back on the ESTI architecture, the NFVI block has the virtualization layer, which is the implementation of VxFs as virtual machines, as a container (CUNHA; MOREIRA; SILVA, 2021).

The evolution in server virtualization has a direct influence on the way VxF is deployed and implemented. Containers were explored as an option to optimize the efficiency of deploying VxFs, since the reload and exclusion times of the containers are much shorter (CUNHA; MOREIRA; SILVA, 2021). In the long run, this option may justify a tendency to use containers in NFV instead of a virtual machine.

2.3 IoT and Edge Computing

2.3.1 Internet of Things

The Internet of Things (IoT) represents a comprehensive environment that connects a large number of heterogeneous physical objects, such as appliances, facilities, animals,

vehicles, farms, factories, etc., to the Internet, in order to increase the efficiency of applications, such as logistics, manufacturing, agriculture, urban computing, home automation, assisted living and various real-time computing applications (BUYYA; SRIRAMA, 2019).

Typically, an IoT system follows a cloud-centric architecture, called Cloud-centric Internet of Things (CIoT). For such architecture, physical objects are presented in the form of resources managed by central servers (CHANG; SRIRAMA; BUYYA, 2016b). In addition, devices at the edge of the network have connectivity to the Internet through intermediate gateway-type nodes, such as modems, routers, switches, cellular base stations, and so on (WU et al., 2021). In general, IoT systems are composed of three major technologies: embedded systems, middleware and cloud services. Embedded systems provide intelligence for physical devices at the edge, middleware interconnects the various heterogeneous embedded systems to cloud servers and, finally, cloud systems offer comprehensive storage, processing and networking mechanisms (FARAHZADI et al., 2018).

Although the CIoT model is a common approach to implementing IoT systems, such an architecture is facing the growing challenges of IoT. Specifically, CIoT faces challenges such as bandwidth, uninterrupted, latency, resource constraints and security.

- **Bandwith**: The ever-increasing, high-frequency data rates at which they are produced by objects in the IoT will exceed the availability of bandwidth (JIA et al., 2019). For example, a connected car can generate tens of megabytes of data per second for its route information, speeds, car operating condition, driver condition, surrounding environment, weather, etc. In addition, an autonomous vehicle can generate gigabytes of data per second, due to the need for real-time video streaming. Therefore, relying entirely on the distant cloud to manage things becomes impractical.

- **Uninterrupted**: The long distance between the cloud and the IoT device it can generate problems due to unstable and intermittent network connectivity (WU et al., 2021). For example, a CIoT-based vehicle will not be able to function properly due to disconnections at the intermediate node between the vehicle and the distant cloud.

- **Latency**: The cloud faces the challenges of meeting the end-to-end latency control requirement. Specifically, industrial smart grid systems, autonomous vehicle networks, virtual and augmented reality applications, healthcare and elderly care applications cannot afford the causes of network latency between physical devices and the cloud (FERRARI et al., 2017).

- **Resource-constrained**: Commonly, many IoT devices have limited resources, preventing them from performing complex computational tasks and, in addition

to a high energy expenditure for the continuous transmission of their data to the cloud (GAWALI; DESHMUKH, 2019).

- ❑ **Security**: A large number of IoT devices may not have sufficient resources to protect themselves from cyber-attacks. Specifically, external devices, which rely on the distant cloud to keep them up to date with security software, can be targeted by attackers (NAIK; MARAL, 2017). In addition, the attacker can also damage or control the IoT device by sending fake data to the cloud.

Despite the problems mentioned above, the Internet of Things, sometimes called the Internet of Objects, will change everything - including ourselves. To prove it, consider the impact that the Internet has already had on education, communication, business, science, government and humanity (KROTOV, 2017). Clearly, the Internet is one of the most important and powerful creations in the entire history of mankind. IoT projects are already underway that promise to close the gap between the poor and the rich, improve the distribution of the world's resources to those who need them most and help us understand our planet so that we can be more proactive and less reactive (BUYYA; SRIRAMA, 2019).

2.3.2 Edge Computing

To solve the aforementioned problems, there is an essential need for a new computing paradigm that occurs closer to connected devices. Fog computing was proposed by industries and academia in order to address the aforementioned issues and to satisfy the need for the computing paradigm closest to the connected devices (PAN et al., 2018). Edge Computing bridges the gap between the cloud and IoT devices, enabling computing, storage, networking and data management at network nodes nearby IoT devices. Note that the Edge Computing is not located on the IoT devices, but as close as one hop to them (DOLUI; DATTA, 2017). It is worth noting that the edge can be more than one hop away from IoT devices in the local IoT network.

Open Edge Computing defines edge computing as computing done at the edge of the network through small data centers close to users. The original vision of cutting-edge computing is to provide computing and storage resources close to the user in open standards and in an ubiquitous manner (AI; PENG; ZHANG, 2018). Cutting-edge computing is a crucial computing paradigm in today's IoT device landscape; it integrates IoT devices with the cloud, intelligently filtering, pre-processing and aggregating IoT data through cloud services deployed close to IoT devices (WU et al., 2021).

Some issues that cutting edge computing is well equipped to address are privacy, latency and connectivity. Due to its proximity to users, the latency in cutting-edge computing is usually less than that of cloud computing, if there is sufficient local computing capacity; latency in high-end computing can be slower than the cloud if the local computing unit is not powerful enough (CHEN et al., 2018). Service availability is also greater

in high-end computing because connected devices do not have to wait for a highly centralized platform to provide a service, nor are connected devices limited by the limited resources of traditional mobile computing.

Although fog computing and edge computing move computing and storage to the edge of the network and closer to the end nodes, these paradigms are not identical. In fact, the OpenFog Consortium claims that edge computing is often called fog computing; The OpenFog Consortium distinguishes that fog computing is hierarchical and provides computing, networking, storage, control and acceleration anywhere, from IoT devices to the cloud; while cutting-edge computing tends to be limited to cutting-edge computing (CHIANG et al., 2017). In addition, Chiang et al. (2017) that "fog includes cloud, core, edge, customers and things" and "fog seeks to provide ongoing support of cloud computing services for things, rather than treat the edges of the network as isolated computing platforms" and "fog provides for a horizontal platform that will support common fog computing functions for various sectors and application domains, including, but not limited to, traditional telecommunications services."

2.3.3 Multi-Access Edge Computing

Multi-Access Edge Computing (MEC) was previously referred to as “Mobile Edge Computing”, but the paradigm has been expanded to include a broader range of applications beyond mobile device-specific tasks (GIUST et al., 2018). Examples of multi-access edge computing applications include video analytics, connected vehicles, health monitoring, and augmented reality.

Both edge computing and MEC computing services operate from the edge of the Internet and can function with little to no Internet connectivity. MEC, however, establishes connectivity through a WAN, WiFi, and cellular connections, whereas edge computing generally can establish any form of connectivity (e.g., LAN, WiFi, cellular). MEC is expected to benefit significantly from the up-and-coming 5G platform (PHAM et al., 2020). Likewise, 5G is seen as an enabler of MEC as it allows for lower latency and higher bandwidth among mobile devices, and it supports a wide range of mobile devices with finer granularity.

MEC allows edge computing to be accessible to a wide range of mobile devices with reduced latency and more efficient mobile core networks. MEC also allows for mission-critical delay-sensitive applications over the mobile network (PHAM et al., 2020). It has also incorporated the use of Software Defined Networking (SDN) and Network Function Virtualization capabilities, in addition to 5G technologies. SDN allows for virtual networking deviceso be easily managed through software APIs (KADIYALA; COBB, 2017), and NFV allows for reduced deployment times for networking services through virtualized infrastructure. Moreover, through SDN and NFV, network engineers and possibly enterprise application developers can develop their own orchestrator, whose goal is to

coordinate the resource provisioning across multiple layers (MIRKHANZADEH et al., 2018).

2.3.4 Raspberry Pi

The central idea behind the Raspberry Pi (RPi) project was the development of a small and accessible computing platform that could be used to stimulate children's interest in basic Information and Communication Technology (ICT) education (ZHONG; LIANG, 2016). The rapid evolution of Single Board Computers coupled with low cost made it possible to supply the Raspberry Pi in an affordable way in early 2012 (UPTON; HALFACREE, 2012). Given the proliferation of smartphones, the idea of having a computer at hand capable of executing billions of instructions per second and easy to adopt is nothing short of incredible.

Raspberry Pi boards alone are too complex to be used by the general public; it is the ability of the cards to run embedded Linux, in particular, that makes the resulting platform accessible, adaptable and powerful. Together, Linux and embedded systems enable ease of development for devices that can face future challenges in Multi-Access Edge Computing like smart buildings, the Internet of Things, robotics, smart energy, smart cities, Human-computer interaction (HCI), cyber-physical systems, 3D printing, advanced vehicle systems and many, many other applications (PEREIRA et al., 2018).

The integration of high-level Linux software and low-level electronics represents a paradigm shift in the development of embedded systems. It is revolutionary that you can build a low-level electronic circuit and then install a Linux web server, using just a few short commands, so that the circuit can be controlled over the Internet (RUPANI et al., 2017). Some general characteristics of RPi devices include the following:

- ❏ They are low cost, available for as little as $5–$35.
- ❏ They are powerful computing devices. For example, the Raspberry Pi 4 contains a 1.5 GHz Quad core Cortex-A72 (ARM v8) processor that can run two monitors at once and boths in 4K.
- ❏ They are available in a range of models that are suitable for different applications.
- ❏ They support many standard interfaces for electronic devices.
- ❏ The fanless, energy-efficient Raspberry Pi runs silently and uses far less power than other computers.
- ❏ Raspberry Pi 4 comes with Gigabit Ethernet, along with on-board wireless networking and Bluetooth.

- ❏ Raspberry Pi 4 has upgraded USB capacity: along with two USB 2 ports you'll find two USB 3 ports, which can transfer data up to ten times faster.
- ❏ Different variants of the Raspberry Pi 4 available, depending on how much RAM you need — 2GB, 4GB, or 8GB.

The RPi platform can run the Linux operating system, which means that you can use many open source software libraries and applications directly with it. The availability of the open source software driver also allows you to connect devices such as USB cameras, keyboards and Wi-Fi adapters with your design, without having to provide proprietary alternatives (NALEPA; GWIAZDA; BANAS, 2018). Therefore, you have access to comprehensive code libraries that have been built by a vast community of free software (HALFACREE; UPTON, 2012); however, it is important to remember that the code usually comes with no warranty whatsoever. If there are problems, you will have to rely on the good nature of the community to resolve them. Of course, you can also solve the problems yourself and make the solutions publicly available.

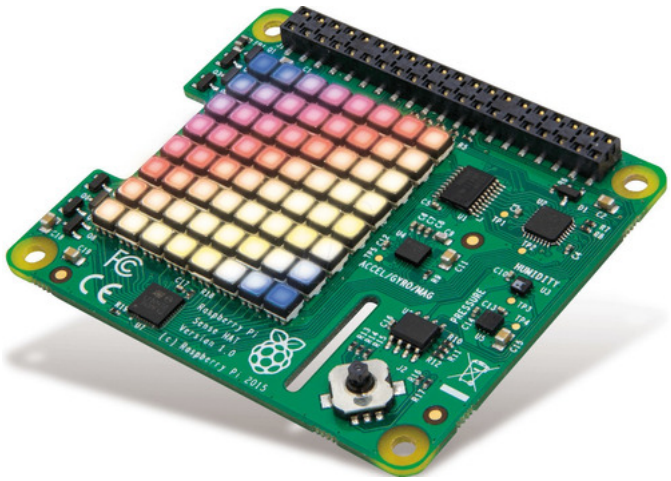

Figure 10 – Hardware Attached on Top of Raspberry Pi.

Source: Manjula, Sreenivasulu e Hussain (2016)

An impressive feature of recent RPi models is that its functionality can be extended with daughter cards, called acHAT, which connect to the GPIO header, as shown in Figure 10. You can design your own HATs and attach them securely to the RPi using this header. In addition, many HATs are available for purchase and can be used to expand the functionality of your Raspberry Pi platform.

2.3.5 Protocols

In networking, a protocol is a set of rules for formatting and processing data. Network protocols are like a common language for computers. The computers within a network

may use vastly different software and hardware (HUANG et al., 2019); however, the use of protocols enables them to communicate with each other regardless.

The HTTP protocol has provided significant services and abilities for the internet for many years, yet was designed and architected for general purpose computing in client/server models (MANOHAR; ASIR, 2018). IoT devices can be very constrained, remote, and bandwidth limited. Therefore, more efficient, secure, and scalable protocols are necessary to manage a plethora of devices in various network topologies such as mesh networks (CHEN; KUNZ, 2016).

In transporting data to the internet, designs are relegated to the TCP/IP foundation layers. TCP and UDP protocols are the obvious and only choice in data communication with TCP being significantly more complex in its implementation than UDP (LAI et al., 2019). UDP, however, does not have the stability and reliability of TCP, forcing some designs to compensate by adding resiliency in the application layers above UDP.

The protocols listed in this section are implementations ofMessage Oriented Middleware (MOM). The basic idea of a MOM is that communication between two devices takes place using distributed message queues (YONGGUO et al., 2019). A MOM delivers messages from one user space application to another. Some devices produce data to be added to a queue, while others consume data stored in a queue. Some implementations require a broker or intermediary to be the central service. In this case, producers and consumers have a publishing and signing relationship with the broker. MQTT and recent implementations of CoAP are MOM implementations (NAIK, 2017); others include CORBA and Java messaging services. An implementation of MOM using queues can use them for design resilience. Data can persist in the queues, even if the server fails (GRUENER; KOZIOLEK; RüCKERT, 2021).

The alternative to implementing MOM is RESTful. In a RESTful model, a server has the state of a resource, but the state is not transferred in a message from the client to the server (RATHOD, 2017). RESTful designs use HTTP methods such as GET, PUT, POST and DELETE to make a resource available through the Universal Resource Identifier (URI). No broker or broker is required in this architecture. Because they are based on the HTTP stack, they enjoy most of the services offered, such as HTTPS security (DURUMERIC et al., 2017). RESTful projects are typical of client-server architectures. Customers initiate access to resources through synchronous request-response patterns.

In addition, clients are responsible for errors, even if the server fails. The Figure 11 below illustrates a MOM versus a RESTful service. On the left is a messaging service using an intermediate server and event publishers and subscribers. Here, many customers can be publisher and subscribers and information may or may not be stored in queues for resilience. On the right is a RESTful design where the architecture is built over HTTP and uses HTTP paradigms for communication from the client to the server.

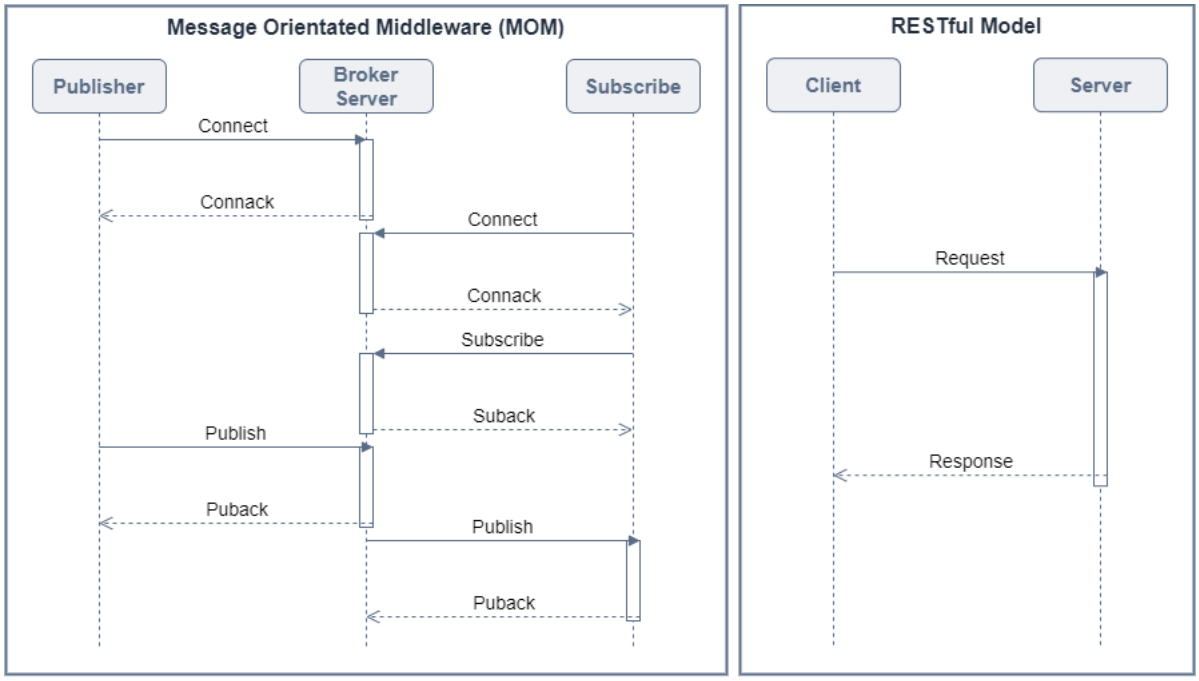

Figure 11 – An example comparing a MOM to a RESTful implementation.

Adapted from: Lea (2018)

2.3.5.1 Message Queue Telemetry Transport

IBM Websphere Message Queue technology was first conceived to solve secure communication problems in distributed, non-competing systems (DAVIES; BROADHURST et al., 2005). A derivative of WebSphere Message Queue was created by Andy Stanford-Clark and Arlen Nipper to address specific restrictions on connecting remote pipelines and pipelines via a satellite connection (GUPTA; QUAMARA, 2020). This protocol became known as Message Queuing Telemetry Transport (MQTT). The objectives of this protocol are:

- ❏ It must be simple to implement
- ❏ Provide a form of quality of service
- ❏ Be lightweight and bandwidth efficient
- ❏ Be independent of the data
- ❏ Troubleshoot security issues

Therefore, MQTT stands for MQ Telemetry Transport. It is an extremely simple and lightweight publishing/signing message protocol, designed for restricted devices and low-bandwidth, high-latency or unreliable networks (YASSEIN et al., 2017). The design

principles are to minimize network bandwidth and device resource requirements while trying to ensure reliability and some degree of delivery guarantee. These principles also end up making the ideal protocol for the emerging Machine to Machine (M2M) or Internet of Things world of connected devices, and for mobile applications where features like bandwidth and power are valuable (WANG; FAPOJUWO, 2017).

Although client-server architectures have been the foundation of data center services for years, publication-subscription models represent a useful alternative to using IoT (WIRAWAN et al., 2018). Publish-subscribe, is a way to separate a customer who is transmitting a message from another customer who is receiving a message. Unlike a traditional client-server model, clients are not aware of any physical identifiers, such as an IP address or port. MQTT is a publish-subscribe architecture, but it is not a message queue. Message queues store messages in nature, while MQTT does not. In MQTT, if no one is subscribing (or answering) a topic, it will simply be ignored and lost. Message queues also maintain a client-server topology in which a consumer is paired with a producer (SONI; MAKWANA, 2017).

A customer that transmits a message is called a publisher; a customer who receives a message is called a subscribe. At the center is an MQTT broker that is responsible for connecting clients and filtering data. These filters provide subject filtering, that is, customers subscribe to topics and certain topic branches and do not receive more data than they want. Each published message must contain a topic and the broker is responsible for relaying that message to subscribed customers or ignoring it (SONI; MAKWANA, 2017).

MQTT successfully separates publishers from subscribes. As the broker is the regulator between publisher and subscribes, there is no need to directly identify each one based on physical aspects (such as an IP address). This is very useful in IoT deployments, as the physical identity may be unknown or ubiquitous. MQTT and other Publish-Subscribe models are also time-invariant. This means that a message published by a customer can be read and responded to at any time by a subscriber (SONI; MAKWANA, 2017). A subscriber may be in a very low state of limited power / bandwidth and respond minutes or hours later to a message. Because of the lack of physical relationships and time, Publish-Subscribe models adapt very well to extreme capacity.

Cloud-managed MQTT brokers can generally ingest millions of messages per hour and support tens of thousands of publishers. MQTT is independent of the data format. Any type of data can reside in the payload, so publishers and subscribers must understand and agree with the data format. The maximum allowed packet size in MQTT is 256 MB, which allows for an extremely large payload (STANDARD, 2014). The payload field is optional. It is advisable to check with the specifications of MQTT brokers to ensure that the load sizes are compatible.s

2.3.5.2 Constrained Application Protocol

The Constrained Application Protocol (CoAP) was first draft by Internet Engineering Task Force (IETF) Constrained RESTful Environments (CoRE) in June 2014 but had worked for several years on its creation. It is specifically intended as a communication protocol for constrained devices (IGLESIAS-URKIA et al., 2018). The protocol is unique as it is first tailored for Machine to Machine communication between edge nodes. It also supports mapping to HTTP through the use of proxies. This HTTP mapping is the on-board facility to get data across the internet.

CoAP is excellent for providing a similar and easy structure of addressing resources on the web with reduced resources and bandwidth. A study by Colitti, Steenhaut e Caro (2011) demonstrated the efficiency of CoAP over standard HTTP. CoAP provides similar functionality with significantly less overhead and power requirements.

CoAP is based on the concept of imitating HTTP protocol that uses high computational resources for the execution of an equivalent but lightweight protocol for IoT environments (IGLESIAS-URKIA et al., 2018). It is not an HTTP replacement, as it lacks resources; HTTP requires more powerful, service-oriented systems. The CoAP characters can be summarized as follows:

- ❏ Similar to HTTP.
- ❏ Connection-less protocols.
- ❏ Asynchronous message exchanges.
- ❏ Design requirements and lightweight features and low platform overhead.
- ❏ Support for URI and content types.
- ❏ Built in UDP versus TCP / UDP for a normal HTTP session.
- ❏ A stateless HTTP mapping that allows proxies to be connected in HTTP sessions.

CoAP shares context, syntax, and usage similarly to HTTP. Addressing in CoAP is also styled like HTTP. An address extends to the URI structure (IGLESIAS-URKIA et al., 2018). As in HTTP URIs, the user must know the address beforehand to gain access to a resource. At the top-most level, CoAP uses requests such as GET, PUT, POST, and DELETE, as in HTTP.

A CoAP system has seven main actors:

- ❏ **Endpoints**: These are the sources and destinations of a CoAP message.
- ❏ **Proxies**: A CoAP endpoint that is tasked by CoAP clients to perform requests on its behalf. Reducing network load, access sleeping nodes, and providing a layer of

security are some of the roles of a proxy. Alternatively, a proxy can map from one CoAP request to another CoAP request or even translate to a different protocol (cross-proxying).

- ❏ **Client**: The originator of a request. The destination endpoint of a response.
- ❏ **Server**: The destination endpoint of a request. The originator of a response.
- ❏ **Intermediary**: A client acting as both a server and client towards an origin server. A proxy is an intermediary.
- ❏ **Origin servers**: The server on which a given resource resides.
- ❏ **Observers**: An observer client can register itself using a modified GET message. The observer is then connected to a resource and if the state of that resource changes, the server will send a notification back to the observer. Observers are unique in CoAP and allow a device to watch for changes to a particular resource. In essence, this is similar to the MQTT publish-subscribe model where a node will subscribe to an event.

MQTT and CoAP are by far the predominant IoT internet protocols in the industry with support from nearly every cloud provider. MQTT provide a scalable and efficient publish-subscribe model of data communication while CoAP provides all the relevant features of an HTTP RESTful model without the overhead (LARMO; RATILAINEN; SAARINEN, 2019). An architect must consider the overhead, power, bandwidth, and resources necessary to support a given protocol and have enough forward vision to ensure the solution scales.

2.4 Monitoring Distributed Systems

Understanding the state of a system is essential to ensure reliability and stability. Information about system health and performance not only helps entire teams react to problems, but also provides security when making changes to their services (TURNBULL, 2018). One of the best ways to gain knowledge about the system is through robust monitoring that gathers metrics, data visualization, as well as alerts to operators when something seems to be wrong (SUKHIJA; BAUTISTA, 2019). The four golden signs of monitoring are (BEYER et al., 2016):

- ❏ **Latency**: The time it takes to fulfill a request. It is important to distinguish between successful request latency and failed request latency.
- ❏ **Traffic**: A measure of how much demand is being placed on your system, measured on a specific metric of the high-level system. For a web service, this measurement is

usually HTTP requests per second, for an audio streaming system, this measurement can focus on the network's Input/Output rate or simultaneous sessions.

- **Errors**: The rate of requests that fail, either explicitly (for example, HTTP 500s), implicitly (for example, a successful HTTP 200 response, but associated with the wrong content) or per policy (for example, "If you have committed to a second response times, any request over a second is an error").

- **Saturation**: How "full" the system is. A measure of your system fraction, emphasizing the resources that are more restricted e.g. the memory state. In complex systems, saturation can be supplemented with higher-level load measurement: your service can handle twice as much traffic, handle just 10% more traffic, or handle even less traffic than it currently receives.

All of these monitoring requirements on top of one another can result in a very complex monitoring system - your system can end up with many levels of complexity (KORABLEVA; KALIMULLINA; KURBANOVA, 2017). So, design your monitoring system with simplicity. In modern production systems, monitoring systems track a constantly evolving system with changes in software architecture, load characteristics and performance targets (KURTANOVIC; MAALEJ, 2017). An alert that is currently exceptionally rare and difficult to automate can become frequent, perhaps even deserving a script to resolve it (REZENDE et al., 2019). A healthy monitoring and alert pipeline is simple and easy to reason. It focuses mainly on the symptoms oriented to causes to assist in debugging problems.

2.4.1 Prometheus

Prometheus is an open source monitoring system based on metrics. It has a simple but powerful data model and a query language that allows you to analyze the performance of your applications and infrastructure (BRAZIL, 2018). Prometheus was written primarily in Go and licensed under the Apache 2.0 license. It is estimated that in 2018, tens of thousands of organizations were using Prometheus in production. In 2016 the Prometheus project became the second member of the Cloud Native Computing Foundation (CNCF) (TURNBULL, 2018).

A simple text format facilitates the exposure of metrics to Prometheus. Other monitoring systems, both open source and commercial, have added support for this format (BRAZIL, 2018). This allows all of these monitoring systems to focus more on key features, instead of each having to spend time duplicating efforts to support each piece of software that a user wants to monitor.

The data model identifies each time series not only with a name, but also with an unordered set of key-value pairs called labels. The PromQL query language allows aggre-

gation into any of these labels, so you can analyze them not only by process, but also by data center and service or any other labels you have defined (SABHARWAL; PANDEY, 2020). They can be represented graphically in panel systems, such as Grafana.

Alerts can be defined using exactly the same PromQL query language that is used for the generation of dashboards (SABHARWAL; PANDEY, 2020). If you make a dashboard, you can alert about it (RAHMAN; AMNUR; RAHMAYUNI, 2020). Labels make it easy to maintain alerts, as you can create a single alert covering all possible label values. In some other monitoring systems, you would have to create an alert per machine / application individually.

For all these features and benefits, Prometheus is efficient and simple to execute. A single Prometheus server can ingest millions of samples per second (BRAZIL, 2018). It is a single binary statically linked with a configuration file. All components of Prometheus can run in containers and avoid doing anything complicated that gets in the way of configuration management tools. It was designed to be integrated with the infrastructure you already have and on which it was built, not to be a management platform in itself (TURNBULL, 2018). As a metrics-based monitoring system, Prometheus is designed to track overall system health, behavior, and performance rather than individual events.

2.4.1.1 Prometheus Architecture

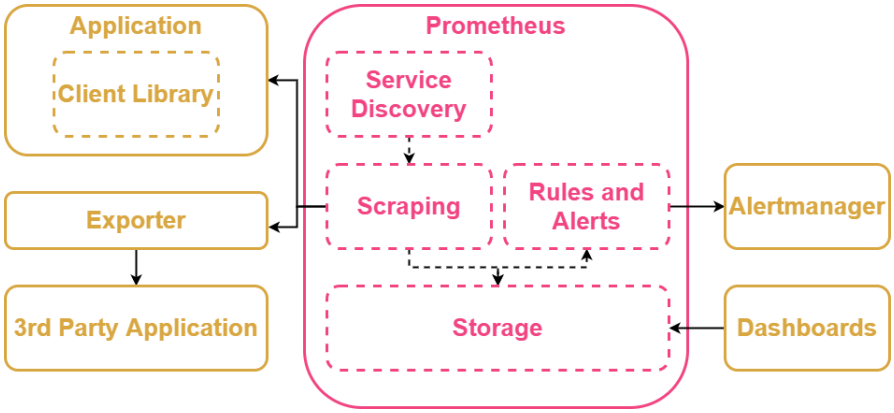

Figure 12 – The Prometheus architecture.

Adapted from: Brazil (2018)

Figure 12 shows the overall architecture of Prometheus. Prometheus discovers targets to be found in service discovery (BRAZIL, 2018). The collected data is stored and you can use it on dashboards using PromQL or send alerts to the alert manager, which will convert them into pages, emails and other notifications (RAHMAN; AMNUR; RAHMAYUNI, 2020).

Typically, with just two or three lines of code, you can define a metric and add the desired instrumentation embedded in the code you control. This is known as direct instrumentation. Client libraries are available for all major languages and runtimes. The Prometheus project provides official client libraries in Go, Python, Java and Ruby. There are also a variety of third-party client libraries, such as .Net, Node.js, Haskell, Erlang and Rust (CHAKRABORTY; KUNDAN, 2021).

Client libraries take care of all the essential details, such as thread-safety, accounting and the metric display format in response to HTTP requests. Since metrics-based monitoring does not track individual events, the client library memory usage does not increase the more events you have. Instead, memory is related to the number of metrics you have (BRAZIL, 2018).

Some metrics are typically provided out-of-the-box by client libraries, such as CPU usage and garbage collection statistics, depending on the library and runtime environment (TURNBULL, 2018). Client libraries are not restricted to generating metrics in Prometheus text format. Prometheus is an open ecosystem, and the same APIs used to power the generation of metrics can be used to produce metrics capable of powering other instrumentation systems.

An exporter is a piece of software that you deploy alongside the application from which you want to obtain metrics (TURNBULL, 2018). It receives requests from Prometheus, gathers the necessary data from the application, transforms them into the correct format and finally returns them in a response to Prometheus. An exporter can be thought of as a small one-to-one proxy, converting data between an application's metrics interface and Prometheus' display format.

Prometheus needs to know where all its instrumented applications and exporters are running. This is so that Prometheus knows what to monitor and is able to see if something he is to monitor is not responding. With dynamic environments, you cannot simply provide a list of applications and exporters only once, with the possibility of never being out of date. To this end, service discovery has the fundamental role of updating available services to be monitored (CHAKRABORTY; KUNDAN, 2021).

The service discovery provides us with a list of targets to be monitored. To search for metrics, Prometheus sends an HTTP request called a scrape. The response to the scrape is analyzed and inserted into storage. Several useful metrics are also added, such as whether the scrape was successful and how long it took. Scrapes happen regularly; normally you would set it up to happen every 10 to 60 seconds for each target.

Prometheus stores data locally in a customized database. The storage system can handle the intake of millions of samples per second, making it possible to monitor thousands of machines with a single Prometheus server (ARIZA-PORRAS; KUZNETSOV; LEGGER, 2021). The compression algorithm used can reach 1.3 bytes per sample in real-world data. In addition, other databases can be used as storage, such as VictoriaMetrics.

Prometheus has several HTTP APIs that allow you to request raw data and evaluate PromQL queries. They can be used to produce graphics and panels. In addition, Prometheus provides the expression browser, providing APIs for ad hoc queries and data exploration (SABHARWAL; PANDEY, 2020).

The recording rules allow PromQL expressions to be evaluated on a regular basis based on the results ingested in the storage engine. Alert rules are another form of registration rules. They also evaluate PromQL expressions regularly and any results from those expressions can become an alert. Alerts are sent to Alertmanager (SABHARWAL; PANDEY, 2020).

Prometheus does not offer a clustering solution for storing data on multiple machines, but there are remote read and write APIs that allow other systems to connect and assume this function (CHAKRABORTY; KUNDAN, 2021). This allows PromQL queries to be executed transparently against local and remote data.

2.4.2 OpenMetrics

Metrics are a specific kind of telemetry data. They represent a snapshot of the current state for a set of data. They are distinct from logs or events, which focus on records or information about individual events.

OpenMetrics is primarily a standard format, independent of any specific transport for metrics exposure (EVANS, 2018). This format is expected to be consumed regularly and to be significant in successive exhibitions. Prometheus is based on the consumption of metrics exposed in OpenMetrics format.

Developers must expose metrics in the OpenMetrics text format in response to a simple HTTP GET request to a documented URL for a given process or device. Developers may also expose OpenMetrics formatted metrics in other ways, such as by regularly pushing metric sets to an operator-configured endpoint over HTTP.

This standard expresses all system states as numerical values; counts, current values, enumerations, and boolean states being common examples. Contrary to metrics, singular events occur at a specific time.

OpenMetrics defines 4 different types of metrics that work that cover most situations, namely:

- **Counter**: it is a cumulative metric that represents a single monotonically increasing counter whose value can only increase or be reset to zero on restart. For example, you can use a counter to represent the number of requests served, tasks completed, or errors.

- **Gauge**: it is a metric that represents a single numerical value that can arbitrarily go up and down. Gauges are typically used for measured values like temperatures or

current memory usage, but also "counts" that can go up and down, like the number of concurrent requests.

- **Histogram**: it is samples observations (usually things like request duration or response sizes) and counts them in configurable buckets. It also provides a sum of all observed values.

- **Summary**: it is samples observations (usually things like request duration and response sizes). While it also provides a total count of observations and a sum of all observed values, it calculates configurable quantiles over a sliding time window.

2.5 Related Work

There are several lines of research in machine virtualization, and some are applied to the Edge Computing environment. Table 2 highlights a comparison between the different solutions found in the state of the art. For each proposal, some aspects were analyzed. First, the virtualization platform, be it LXD, Docker, KVM, Xen and others. Second, the type of infrastructure used to make virtual machines or containers run is on a Single Board Computer or on a Commodity Server. Third, the way used to collect metrics on the infrastructure has been verified. Fourthly and lastly, the metrics collected from the infrastructure until service itself.

Ramalho e Neto (2016) conduct and evaluate the performance of the container-based approach compared to hypervisor-based virtualization when running on devices typically used at the network edge. The study was performed by executing several synthetic benchmarks providing an insight into the performance overhead introduced by Docker containers (lightweight-virtualization) and KVM VMs (hypervisor-virtualization) running at network edge devices.

Ahmad, Alowibdi e Ilyas (2017) describes the Virtualized Internet of Things (vIoT). It's a test enviroment that they argue in favor of infrastructure as an IoT service as a possible model for implementing future IoT services. The vIoT testbed was built from open source, more precisely from OpenStack, LXD, and Raspberry Pi.

Cziva e Pezaros (2017) identifies virtualization opportunities at the edge of the network by presenting Glasgow Network Functions (GNF), a container-based NFV platform that performs and orchestrates VxFs. Finally, there is an economy of such an approach in using the central network, providing less latency to requests in networks.

Li et al. (2017) investigates the performance overhead of a Docker container compared to a Virtual Machine. It is shown that the performance of virtualization can vary from resource to resource. Furthermore, it is shown that the hypervisor-based solution does not come with higher performance overhead in all cases, such as Quality of Service (QoS), in terms of transaction and storage speed.

Riggio et al. (2018) discusses the fundamental challenges of deploying NFV in dispersed environments and then introduces the LightMANO framework, a multi-access network operating system that converges Software Defined Network and NFV into a single lightweight platform for managing and orchestrating network services in an infrastructure distributed NFV.

Rigazzi et al. (2019) proposes a new solution to effectively deploy a 360 video streaming service using the edge and fog computing platform developed by 5G-CORAL. We consider a popular view-port adaptation technique based on adaptive block encoding streaming, where the 360 video is partitioned into small blocks, which are independently encoded and transmitted according to the viewing orientation, and then sewn to recompose the frame 360. To compensate for the extra computational complexity needed to continuously adapt the quality of the video stream, we spread the computing tasks are divided into three different levels, namely, mist, edge and cloud, according to their respective computational resources and requirements of latency. The numerical results show that such an approach can alleviate the cloud delivery service by reducing GPU load, energy consumption and memory usage.

Richards, Moreira e Silva (2020) proposes management and orchestration of Virtual Network Functions in Edge Computing using standard solutions such as Open Source MANO and OpenStack. To demonstrate its viability, an experimental evaluation using a Raspberry Pi 3 infrastructure is demonstrated. The assessment showed the feasibility of using low-cost devices, such as Raspberry Pi 3, with standard management solutions used in the central cloud.

Tambe, Mandge e Antony Franklin (2020) studies the impact of virtualization technologies on the implementation of the MEC framework and its components. The impact of NFV virtualization technologies on MEC is also studied, such as integration time, instantiation time, MEC service and application response times. The experiments and their analysis show that containers perform better than VM for instantiating / terminating MEC components in NFV framework.

Mena et al. (2020) suggests a solution in which a multi-layer orchestration layer with an intelligence of coordination between intra-orchestrators the activation of NFV, MEC and Cloud Native orchestrators within the slice management life-cycle. This involves homogenizing the descriptors and models used by different types of orchestrators, as well as considering the workflows related to slicing that can be resources if applied at scale.

Table 2 – State of the art summary.

	Virtualization	**Infrastructure**	**Monitoring**	**Metrics**
Ramalho e Neto (2016)	Docker and KVM	Single Board Computer (AArch64)	Benchmarks: NBench, SysBench, Bonnie++, LINPACK, STREAM	Schedule thread, floating-point operations, I/O operantions, memory bandwidth.
Ahmad, Alowibdi e Ilyas (2017)	LXD	Single Board Computer (AArch64)	OpenStack Cloud Computing	Container batch launch times for operating systems Ubuntu 14.04 Trusty Tahr, Ubuntu 16.04 Xenial Xerus and CirrOS 0.3.4.
Cziva e Pezaros (2017)	XEN, ClickOS, KVM, Container	Commodity Server (x86_64)	Virtual Infrastructure Manager (VIM) and OpenDaylight.	Delay of idle round-trip time (RTT). Time to manage (create, start, and stop) virtual machines and containers. Idle memory consumption.
Li et al. (2017)	Docker and VMWare	Commodity Server (x86_64)	Benchmarks: Iperf, HardInfo, STREAM, Bonnie++	Data throughput in communication. Latency in computation. Data throughput in memory. Transaction speed, Data throughput in disk.
Riggio et al. (2018)	Docker and VirtualBox	Commodity Server (x86_64)	Command Linux: top and stat	Round Trip Time in send a web page request. Amount time required to boot application. CPU and memory utilization.
Richards, Moreira e Silva (2020)	KVM	Single Board Computer (AArch64)	Psutil tool	Percentage of CPU. Memory response. Buffer swap memory. Temperature of CPU. Boot time machines.
Rigazzi et al. (2019)	KVM and LXD	Commodity Server (x86_64)	GPU-Z	GPU load. GPU power consumption. Memory usage. Bandwith consumed.
Richards, Moreira e Silva (2020)	KVM	Single Boad Computer (AArch64)	Psutil tool	Percentage of CPU. Memory response. Buffer swap memory. Temperature of CPU. Boot time machines.
Tambe, Mandge e Antony Franklin (2020)	Docker and KVM	Commodity Server (x86_64)	Application Function (AF), which acts as a registered service.	Instantiation and temination time. Service response time. Service registry time. Traffic rules time. DNS rules time.
Mena et al. (2020)	LXD	Commodity Server (x86_64)	Zabbix: monitoring the virtualized environment.	High CPU. Memory utilization. Network bandwith. TCP Connections. Disk space.

Chapter 3

The GARNET Architecture

The Internet of Things (GUBBI et al., 2013) represents a comprehensive environment that interconnects a large number of heterogeneous physical objects or things such as appliances, facilities, animals, vehicles, farms, factories, etc., in order to enhance the efficiency of the applications such as logistics, manufacturing, agriculture, urban computing, home automation, environment assisted living, and various real-time ubiquitous computing applications.

Commonly, an IoT system follows the architecture of the Cloud-centric Internet in which the physical objects are represented in the form of Web resources that are managed by the servers in the global Internet (CHANG; SRIRAMA; BUYYA, 2016a). Fundamentally, in order to interconnect the physical entities to the Internet, the system will use various front-end devices such as wired or wireless sensors, actuators, and readers to interact with them. Further, the front-end devices have the Internet connectivity via the mediate gateway nodes such as Internet modems, routers, switches, cellular base stations, and so on. In general, the common IoT system involves three major technologies: embedded systems, middleware, and cloud services, where the embedded systems provide intelligence to the front-end devices, middleware interconnects the heterogeneous embedded systems of front-end devices to the cloud and finally, the cloud provides comprehensive storage, processing, and management mechanisms.

Although the Cloud-centric model is a common approach to implement IoT systems, it is facing the growing challenges in IoT. Specifically, Cloud-centric faces challenges like bandwidth, latency, uninterrupted, resource-constraint, and security (CHIANG; ZHANG, 2016).

In the last decade, several approaches have tried to extend the centralized cloud computing to a more geo-distributed manner in which the computational, networking, and storage resources can be distributed to the locations that are much closer to the data sources or end-user applications. For example, the geo-distributed cloud-computing model (SAJJAD et al., 2016) tends to partition the portions of processes to the data centers near the edge network. Further, the mobile cloud computing model introduced the

physical proximity-based cloud computing resources provisioned by the local wireless Internet access point providers. Moreover, academic research projects (LOKE et al., 2015) have experimented with the feasibility of the mobile ad hoc network (MANET)-based cloud using the advanced RISC machine (ARM)-powered devices. Among the various approaches, the industry-led Edge Computing architecture, which was first introduced by Cisco research (BONOMI et al., 2012), has gained the most attention.

For example, a edge-enabled IoT system can distribute the simple data classification tasks to the IoT devices and assign the more complicated context reasoning tasks at the gateway devices. Further, for the analytics tasks that involve terabytes of data, which requires higher processing power, the system can further move the processes to the resources at the core network such as the data centers of wide area network (WAN) service providers or it can utilize the cloud. Certainly, the decision of where the system should assign the tasks among the resources across different tiers depends on efficiency and adaptability. For example, smart systems may need to assign certain decision-making tasks to the edge devices in order to provide timely notification about the situation, such as the patient's condition in the smart healthcare, the security state of the smart home, the traffic condition of the smart city, the water supply condition of smart farming, or the production line operation condition of a smart factory.

The industry has seen edge as the main trend for the practical IoT systems, like energy/utilities, transportation, healthcare, industry, and agriculture, and the leading OpenFog consortium has established collaboration with major industrial standard parties such as European Telecommunications Standards Institute and IEEE Standard for Edge Computing.

3.1 Relevant Technologies

The notion of having computational resources near the data sources may seem not new. Particularly, the term edge computing appeared in 2004 to illustrate a system that distributes program methods and the corresponding data to the network edge towards enhancing performance and efficiency(PANG; TAN, 2004).

Similarly, the notion of having virtualization technology-based computing resources within the Wi-Fi subnet was introduced in 2009. However, the real industrial interest in extending computational resources to the edge network only started after the introduction of cloud computing for IoT (SATYANARAYANAN et al., 2009). Prior to that, applying utility cloud at the edge network was more or less a research topic in academia without explicit definition or architecture and with minor industrial involvement. In contrast, the industry has invested in Edge Computing architecture by establishing OpenFog consortium founded by ARM Holdings, Cisco, Dell, Intel, Microsoft, Princeton University, and over 60 members from major industrial and academic partners in the world. Further, in

collaboration with international standard organizations such as ETSI and IEEE, fog has become a major trend in general information and communication technology today.

Certain other works have been describing Multi-Access Edge Computing as an exchangeable term with edge (TALEB et al., 2017). Essentially, ETSI introduced MEC as a standard from the perspective of telecommunication, in which ETSI specifies the application programming interface standards about how telecommunication companies can provide computing virtualization-based service to their clients based on extending the existing infrastructure used in network function virtualization. Although MEC is exchangeable term with Edge, according to the recent collaboration between OpenFog and ETSI, MEC will become a practical approach to hasten the realization of Edge Computing.

The high-level description of the advantages provided by Edge Computing, especially Multi-Access Edge Computing, is due to some basic mechanisms supported by MEC-enabled devices, namely, storage, compute, and networking.

The MEC storage mechanism corresponds to the temporary storage of data and caching in Multi-Access Edge Computing nodes to improve the performance of information or content delivery.

MEC nodes provide computing engines primarily in two models - infrastructure or platform as a service and Software as a Service. In general, Multi-Access Edge Computing providers offer infrastructure and platform based on two approaches - hypervisor virtual machines or container engines, which allow flexible platforms for customers to deploy the custom software they need in an environment sandbox hosted at edge us. In addition to Software as a Service is also promising in providing edge services.

Multi-Access Edge Computing provides network services with a key concept - virtualization. Network Functions Virtualization transitions network service functions from hardware to software. Together, Multi-Access Edge Computing and Network Functions Virtualization provide an effective solution for modern computing, ensuring speed and reliability for end users. One benefit of using MEC and NFV is the combination of scalability and low latency. NFV provides scalability for network computing, increasing and decreasing network resources depending on the need and usage of the application. Network slicing is possible through NFV. Different partitions (or slices) of the edge can be targeted to specific network functions. This practice ensures that these high-bandwidth applications get the network resources they need, without disrupting the computing functions of other applications that use the same end of the network. A multi-layer MEC NFV architecture ensures quality of service throughout the life-cycle of applications running at the edge of the network. Cutting-edge computing plays a significant role in alleviating and solving network demands like the Internet of Things (LV; XIU, 2020).

MEC nodes enable the interconnection of things and cloud with standard IP-based networking protocols such as request/response based TCP/UDP sockets, HTTP, Con-

strained Application Protocol (CoAP), Message Queue Telemetry Transport (MQTT) and so on. Specifically, IoT devices can operate server-side functions (eg, CoAP server) that allow endpoints, which act as the cloud proxy, to collect data from them and then forward the data to the cloud. In addition, the edge nodes can also act as message intermediaries for the publish-subscribe based protocol that allows IoT devices to publish data streams to the edge nodes and allow the cloud back-end or others IoT devices to subscribe to the data streams from the edge nodes.

3.2 Challenges Addressed

While cloud computing has been a key enabling technology for the IoT, a small increase in the percentage of connected or cyber-physical objects represents dramatic change in the feature space of computing and a potential tsunami of computation and hyper-connectivity, which today's infrastructure will struggle to accommodate at historic levels of quality of service. Large-scale distributed control systems, geo-distributed applications, time-dependent mobile applications, and applications that require very low and predictable latency or interoperability between service providers are just some of the IoT application categories that existing cloud infrastructures are not well-equipped to manage at a hyperscale (BONOMI et al., 2014). Traditional cloud computing architectures were simply not designed with an IoT, characterized by extreme geographic distribution, heterogeneity and dynamism, in mind. As such, a novel approach is required to meet the requirements of IoT including transversal requirements (scalability, interoperability, flexibility, reliability, efficiency, availability, and security) as well as cloud-to-thing computation, storage and communication needs (BOTTA et al., 2016).

In response to the need for a new intermediary layer along the cloud-to-thing continuum, Edge Computing, especially Multi-Access Edge Computing, has emerged as a computing paradigm situated between the cloud and smart end-devices providing data management and/or communications services to facilitate the execution of relevant IoT applications. The ambition for Multi-Access Edge Computing is greater support for interoperability between service providers, real-time processing and analytics, mobility, geographic distribution, and different device or MEC node form factors, and as a result the achievement of quality of service expectations. Despite these advantages, Multi-Access Edge Computing adds a layer of complexity that operators across the cloud-to-thing continuum need to account for, not least resource orchestration and management (ÖSTBERG et al., 2017).

Current research has primarily focused on decentralizing resources away from centralized cloud data centers to the edge of the network and making use of them for improving application performance. Typically, edge resources are configured in an ad hoc manner and an application or a collection of applications may privately make use of them. These

resources are not publicly available, for example, like cloud resources. Additionally, edge resources are not evenly distributed but are sporadic in their geographic distribution. However, ad hoc, private, and sporadic edge deployments are less useful in transforming the global Internet (CHANG; SRIRAMA; BUYYA, 2016a). The benefits of using the edge should be equally accessible to both the developing and developed world for ensuring computational fairness and for connecting billions of devices to the Internet. However, there is minimal discourse on how edge deployments can be brought to bear in a global context – federating them across multiple geographic regions to create a global edge-based fabric that decentralizes data center computation.

Industrial marketing research forecasts that the market value of edge hardware components will reach $7,659 million by the year 2022, which indicates that more Edge-ready equipment such as routers, switches, IP gateway, or hubs will be available in the market (SHI et al., 2016). The edge experience faces challenges in defining standardization for software and hardware. First, edge-based equipment raises a question for the vendors as to what edge platform and related software packages should be included in their products. Second, edge-based software raises a question for the vendors regarding compatibility. Specifically, users may have devices in heterogeneous specification and processing units (e.g. x86, ARM etc.) in which the vendor may need to provide a version for each type of hardware. Moreover, developing and maintaining such an edge-based software can be extremely costly unless a corresponding common specification or standard for hardware exists.

Monitoring of edge resources will be a key to provide the edge advantages. For example, performance metrics will need to be monitored for implementing auto-scaling methods to balance workloads on the edge. Existing monitoring systems for distributed systems either do not scale or are resource consuming (ABDERRAHIM et al., 2017). These are not suitable for large-scale resource-constrained edge deployments. Current mechanisms for auto scaling resources are limited to single-edge nodes and employ lightweight monitoring. However, scaling these mechanisms is challenging.

However, designing a proper monitoring system is a challenging task because Edge infrastructure differ significantly from traditional Cloud infrastructure (ABDERRAHIM et al., 2017). Three intrinsic characteristics differ in their development: First, while a traditional Cloud Computing basically depends on high performance servers and networks placed in very few sites, Edge Computing, especially Multi-Access Edge Computing infrastructure, is deployed in a significant number of sites and the distance that separates its resources can reach hundreds of kilometers affecting latency as well as bandwidth. Second, the infrastructure in Multi-Access Edge Computing is made up of several heterogeneous resources with different characteristics in terms of capacity, reliability and usability. Third, infrastructure resources based on Edge Computing, especially Multi-Access Edge Computing, can be highly dynamic and can permanently enter and exit the

network according to service usage, failures, policies and maintenance operations.

3.3 General Architecture

Therefore, this work proposes and evaluate GARNET (*edGe virtuAlized eveRything fuNction management architEcTure*) an architecture compatible with ETSI Management and Orchestration software stack for managing Virtualized Everything Functions in the edge. This architecture is capable of deploying services across bare-metal servers and low-cost devices in a seamless way. This work also focuses on monitoring application infrastructure until the services, which is a key element for managing virtualized systems. Some research on monitoring tools has been proposed in recent years (RODRIGUES et al., 2016). However, little is investigated from the perspective of Edge Computing, much less about a unified perspective of monitoring in Multi-Access Edge Computing using tools already available. In addition, a unified platform to monitor computing resources in Multi-Access Edge Computing is proposed and implemented.

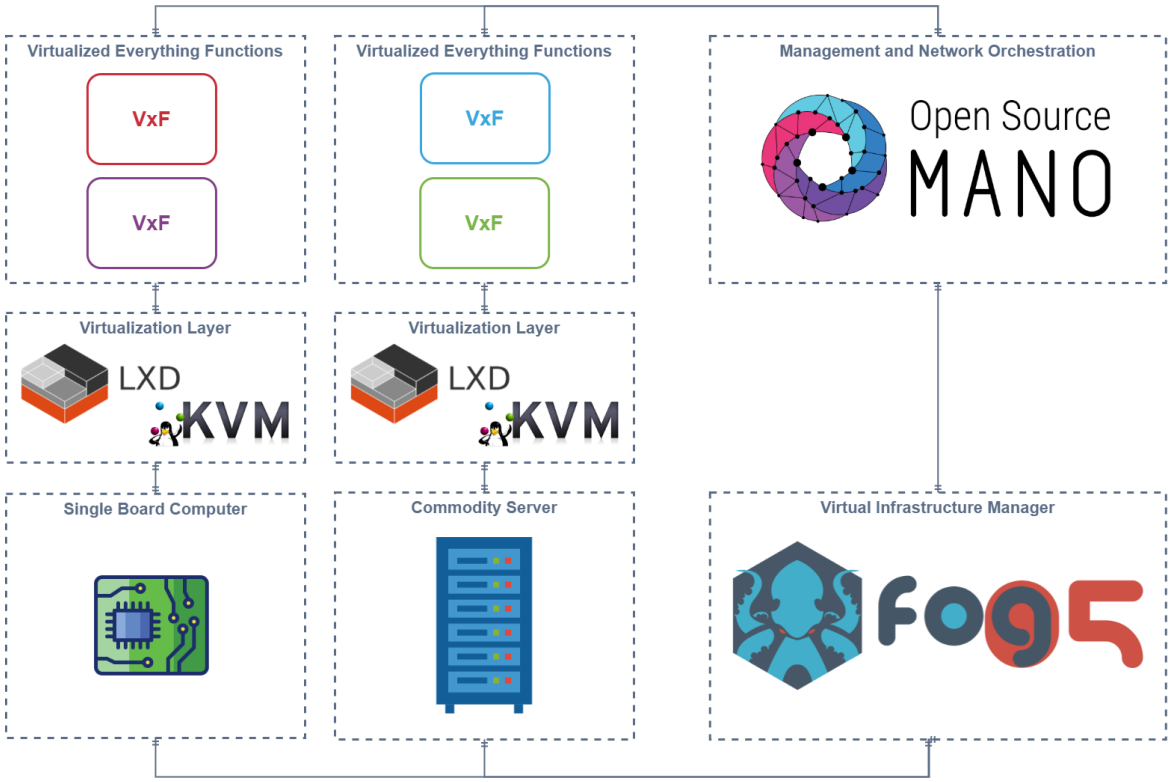

Figure 13 – GARNET components Mapped to ETSI NVF Architecture.

Figure 13 shows the general NFV architecture in Multi-access Edge Computing, highlighting its logical components. Such architecture has several components, each developed as a module with independent submodules.

Open Source MANO is an ETSI-hosted open source stack for NFV (GARCÍA-ROIS et al., 2021), aligned with ETSI NFV Information Models, capable of consuming openly published information models, available to everyone, suitable for all VxFs, operationally significant and VIM-independent.

Eclipse Fog05 is an open source project that allow the end-to-end management of compute, storage, networking and I/O fabric in the Edge and Fog Environment (VALANTASIS et al., 2021). It is based on a decentralized architecture that allows users to manage and deploy different types of applications, packaged as containers, VMs, binaries and so on. All of this can be deployed from big servers to micro-controllers.

An Single Board Computer (BASFORD et al., 2020) is a complete computer built on a single circuit board, with microprocessor(s), memory, Input/Output and other features required of a functional computer. A commodity server is a dedicated computer to running server programs and carrying out associated tasks.

Virtualization is the technology used to run multiple operating systems or applications on top of a single physical infrastructure by providing each of them an abstract view of the hardware (BEHRAVESH; CORONADO; RIGGIO, 2019). It enables these applications or OSs to run in isolation while sharing the same hardware resources. For this project, two virtualization approaches will be enabled, namely LXD (Linux Containers) and KVM. LXD is a next generation system container manager. It offers a user experience similar to virtual machines but using Linux containers instead (LI et al., 2017). KVM is a full virtualization solution containing virtualization extensions (Intel VT or AMD-V). Using KVM, a single node compute can run multiple virtual machines, each one has private virtualized hardware: a network card, disk, graphics adapter, etc (BABU et al., 2014).

Virtualized Everything Functions (SILVA et al., 2019) are one or more virtual machines (or containers like LXD, for example) that implement specialized software that run with resources from standard high capacity servers, networked computers and storage systems, instead of being implemented in specialized physical systems devices.

3.4 Monitoring Architecture

To properly manage the complex scenarios as a result of adopting the Multi-Access Edge Computing paradigm, some crucial tasks take place; Multi-Access Edge monitoring is one of them. Monitoring at the edge gives infrastructure and service providers the ability to look at the virtual / physical resources granted / allocated. Through monitoring, service providers can show edge information to customers. The information retrieved from edge monitoring is also the basis for decisions to, for example, reduce energy consumption, automated service provisioning, traffic management and analysis, data management and security. Figure 14 shows all the components necessary to monitor and collect information about the behavior of the infrastructure for the present project.

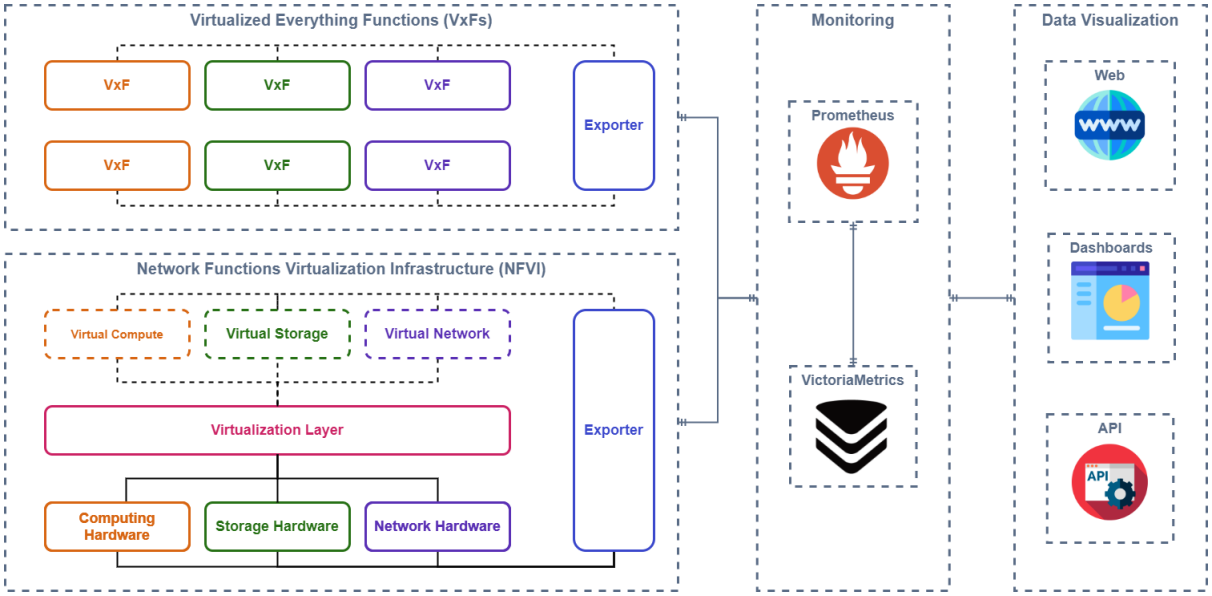

Figure 14 – GARNET monitoring components.

Network Function Virtualization Infrastructure is represented by Raspberry Pi 4 (AArch64) or the Commodity Server (x86_64). The infrastructure components - such as computing, storage and networking - are used to support the software needed to run network applications. This software is represented by the Virtualization Layer and can be a hypervisor like KVM or a container management platform like LXD, for example.

The exporter is a software whose capacity is to obtain the relevant metrics about parts of a system in OpenMetrics format. It creates an endpoint on a specific port where metrics can be sent, via HTTP requests. There are a number of pre-packaged exporters for specific purposes. Alternatively, you can consider developing your own custom exporter using programming languages such as Go, Python, and more. This way it is possible to obtain several metrics at once.

Prometheus works for recording any purely numeric time series. It fits in machine-centric monitoring, designed for reliability, to be the system you go to during an outage to allow you to quickly diagnose problems (BRAZIL, 2018). VictoriaMetrics is a fast, cost-effective and scalable monitoring solution and time series database (ARIZA-PORRAS; KUZNETSOV; LEGGER, 2021). Prometheus was configured to send all data collected to VictoriaMetrics.

All data collect on Network Function Virtualized Infrastructure and Virtualized Everything Functions can be integrated, analyzed and viewed by third party software, like web applications, dashboards and APIs.

Chapter 4

Experimental Evaluation and Analysis

Most NFV platforms aim to exploit traditional or specialized VMs to host VxFs typically found in remote and over-provisioned data centers. However, as a programmable network edge is gaining traction (YOUSEFPOUR et al., 2019), there is a need for lightweight NFV technologies that can exploit the benefits offered by Multi-Access Edge Computing (e.g., localized, high-throughput, and network connectivity, low latency). Service providers have the ability to run customized, high-performance network services, reducing the growing cost of core network management and operations.

Therefore, this work's main goal is to propose and evaluate the GARNET (Edge Virtualized Everything Function Management Architecture), an architecture compatible with ETSI Management and Orchestration software stack for managing Virtualized Everything Functions. This architecture is independent of the underlying hardware, Commodity Server or Single Board Computers (x86_64 or AArch64) and virtualization platform (LXD or KVM) for deploying and monitoring VxFs. Thus, allowing the continuous use of VxFs both on Commodity Servers as well as for Single Board Computers, using Open Source MANO, Eclipse Fog05 and standard service descriptors. In addition, the architecture can provide a complete infrastructure capable of collecting a variety of metrics in OpenMetrics format related to the supporting infrastructure through to the deployed services.

To show GARNET's ability to manage and orchestrate VxFs in Multi-Access Edge Computing, an experimental evaluation was conducted. Therefore, the behavior of a Single Board Computer in supporting multiple VxFs encapsulated both in container and in virtual machines was verified. Experimental tests have shown that using a single-board computer as a next-generation device can be an excellent alternative, centering some processing at the edge and alleviating network congestion. In this way, issues such as bandwidth management, dispersion, low geographic latency and privacy can be minimized.

To show GARNET's ability to manage and orchestration unified of VxFs independent of virtualization hardware, an experimental evaluation was carried out. Therefore, the

behavior of two different infrastructures, located in Multi-Access Edge Computing, was verified, namely a Single Board Computer (AArch64) and a Commodity Server (x86_64). A series of metrics that best describe the behavior of each infrastructure were collected and analyzed, highlighting their similarities as well as their differences. Experimental tests showed that regardless of the hardware (Single Board Computer or Commodity Server) and the virtualization platform (LXD or KVM), both virtualization infrastructures are capable of supporting VxFs in a transparent, homogeneous and analogous way, simplifying all their management and orchestration in the life-cycle of a VxF.

Furthermore, to show GARNET's ability to full monitoring capability, an experimental evaluation of an API Gateway for IoT was performed. The IoT device is manufactured by EnergyNow Technologies and is called Prodbox, which are discrete sensors to count the number of items that cross the production line between the installed transmitter and receiver. The data generated by the device can be used to better understand the pace of production and, based on that, to promote future sales projections. API Gateway, deployed in Multi-Access Edge Computing as a VxF, is an API management tool that sits between the client and a back-end service. It works as an intermediary that accepts all API calls, aggregates the various services needed to run them, and returns the appropriate result. Although full GARNET monitoring in Multi-Access Edge Computing generates a variety of possible metrics in OpenMetrics format, a few metrics were selected for this project.

4.1 Management and Orchestration of Virtualized Everything Functions in Multi-Access Edge Computing

To show GARNET's ability to manage and orchestrate VxFs in Multi-Access Edge Computing, an experimental evaluation was conducted. The experiment includes a Raspberry Pi 4 and a bare metal desktop computer that are together at the edge of the network. The bare metal desktop runs Virtual Infrastructure Manager, represented by Eclipse Fog05. On the other hand, the Raspberry Pi 4 represents Single Board Computing, capable of providing computational resources to support VxFs. It is noteworthy that any ARM64 (or AArch64) device with virtualization capability can be used in place of the Raspberry Pi 4, so the experimental setup here replicated using several other platforms commonly used in the edge.

The Raspberry Pi 4 is a Model B with a quad-core Cortex-A72 (ARMv8) 64-bit 1.5 GHz processor, 4 gigabytes of RAM. The VIM bare-metal computer is an Intel Core 2 Quad with 8 gigabytes of RAM. The Raspberry Pi 4 has an Ubuntu 18.04.5 LTS arm64 server, with support for hardware virtualization. The bare-metal desktop computer uses

a Ubuntu 18.04.5 LTS 64-bit (AMD64) desktop image.

All the hardware was interconnected using a local physical network. The network's router was configured to support the communication between the Virtual Infrastructure Manager, the VxFs on Raspberry Pi 4, and the external world.

4.1.1 Experiment Description

Two Network Service Descriptor was uploaded to OSM for the experiment. These service descriptors indicated the creation of virtual environments on the Raspberry Pi 4 located in the Multi-Access Edge Computing. OSM interacts with the Virtual Infrastructure Manager, which is represented by the Eclipse Fog05. For the creation of the VxFs, the Eclipse Fog05 interacts with the Raspberry Pi 4, which in turn, will run the VxFs over the LXD or KVM.

For the present project the Table 3 summarizes the service descriptors used for each of the virtualization platforms, namely, number of virtual CPUs, amount of RAM memory, maximum storage capacity and network interface for management of VxF's internal activities.

Table 3 – Service Descriptors Summary.

Virtualization Platform	**Operational System**	**vCPUs**	**Memory (MB)**	**Storage (GB)**	**Network**
LXD	Alpine Edge	1	256	1	eth0
KVM	CirrOS 0.5.0	1	256	1	eth0

To analyze the infrastructure's behavior, several Network Services were launched, as much as possible, so that such infrastructure was operational and could respond to new requests. Thus, the launch of any Network Service would not compromise the functioning of an existing one. Therefore, it was always established that, for both cases, the launch of a new Network Service was only possible after the previous service was in an operational state. In total, 8 VxFs were launched on the KVM platform and 27 VxFs on the LXD platform. These limits were experimentally obtained before running the experiments detailed in this section.

4.1.2 Experiment Analysis

CPU: the present work measured how many seconds are spent on the CPU with the operating system's operating modes, that is, kernel mode and user mode. Figure 15 shows the time spent in user mode and kernel mode for each platform. In both cases (KVM and LXD), the time spent in user mode is greater than the time spent in kernel mode. However, this difference is most noticeable with the LXD container manager. It can also be noted that the time spent for KVM is longer in both user mode and kernel mode. This can be considered an improvement of KVM over LXD since more time spent in one

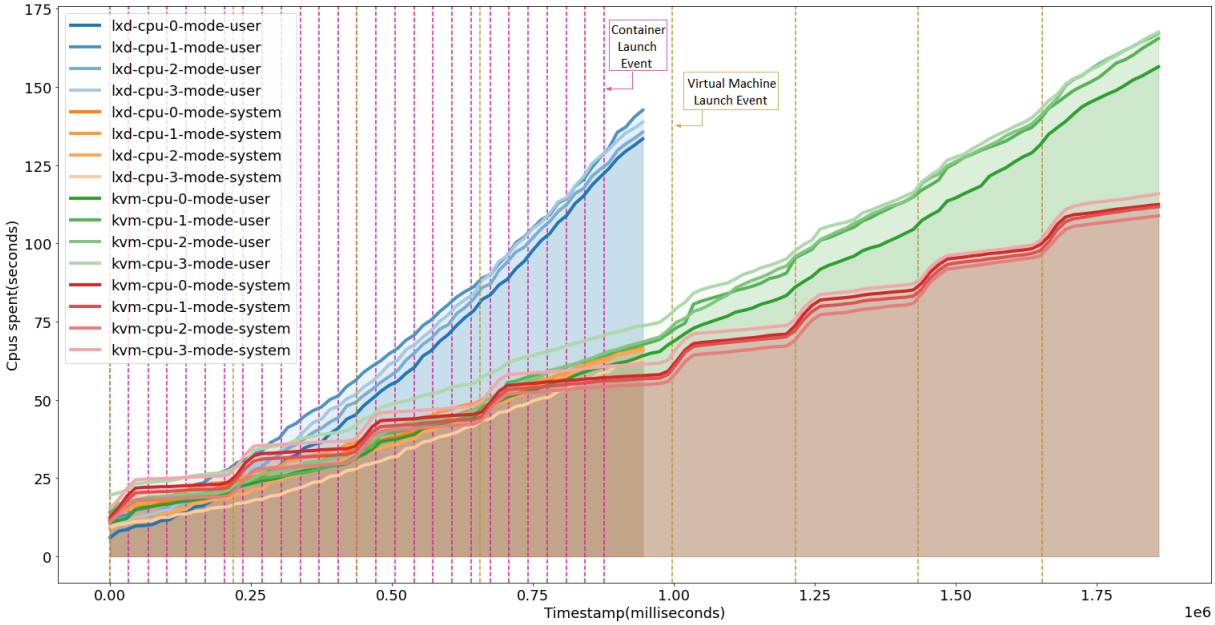

Figure 15 – Seconds the cpus spent in each mode.

mode reduces the changes in the work-space exchange, reducing the overhead that exists in the exchange between user mode and kernel mode. Another important factor is how each processor core is worked. It can see that for LXD, regardless of the mode (kernel or user), the time spent on each processor core is similarly distributed. That is, they present similar behavior throughout the experiment. On the other hand, KVM has a similarity in the kernel mode, in contrast to the user mode. Such mode presents an asymmetry to the cores, probably due to the specialization in its architecture for VxFs components' virtualization.

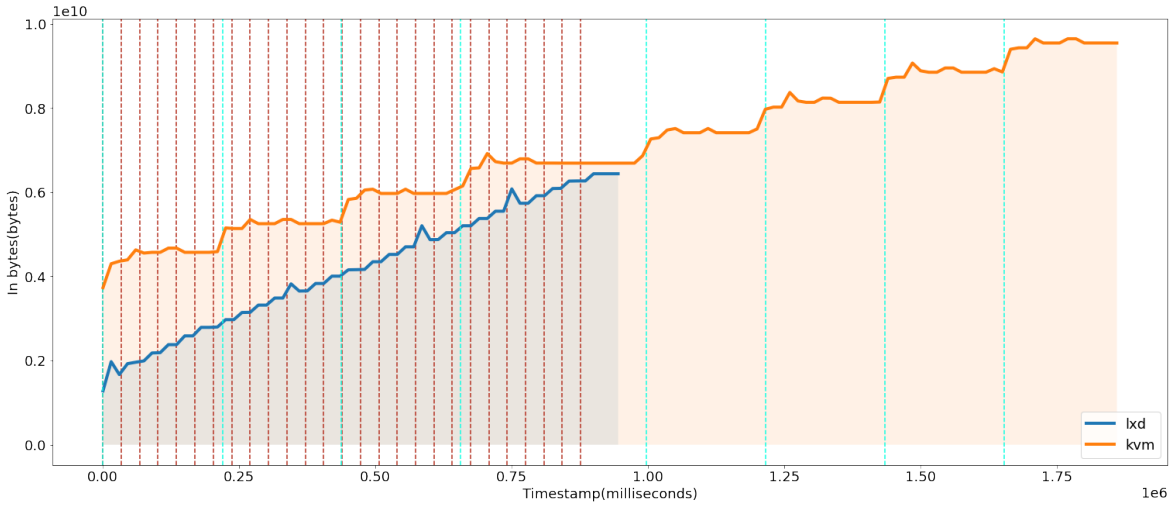

Figure 16 – The amount of memory presently allocated on the system.

Memory: for the present project, the amount of memory currently allocated in the system was measured. Figure 16 show the committed memory for each platform. Committed memory is the sum of all memory that has been allocated by the processes, even if it has not yet been "used" by them. A process that allocates 1 gigabyte of memory, but uses only 300 megabytes of that memory, will appear using only 300 megabytes of memory, even if you have the address space allocated for the entire 1 gigabyte. It appears that in both cases the whereas new VxFs are launched, more memory is consumed. However, it is noted that Network Services were established with the same address space, that is, 1 gigabyte, less consumption for the LXD and the KVM. Therefore, it is concluded that there is an optimization of the memory on the part of the LXD concerning the KVM, thus supporting more VxFs in an Edge Computing environment. Besides, it is noteworthy that such optimization can cause only information more relevant to VxF to be present in memory, creating low rate swap operations, thereby reducing I/O operations.

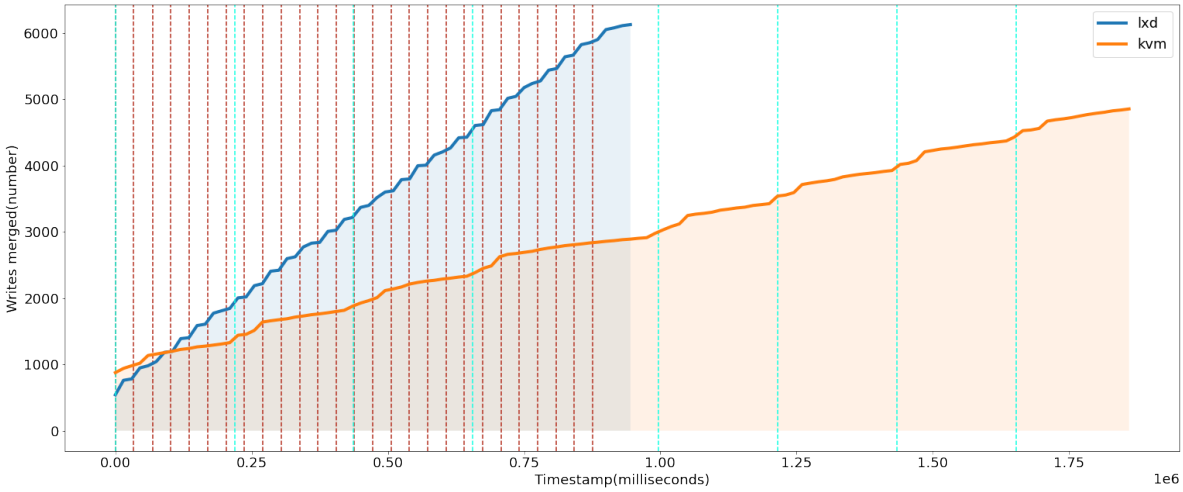

Figure 17 – The number of writes merged.

Disk: for the present project, the number of adjacent writings that can be combined for efficiency was measured. Thus, two 4-kilobyte recordings can become an 8-kilobyte recording before it is finally delivered to the disk, and, therefore, they will be counted (and queued) as just an I/O operation. This metric is critical. Figure 17 clearly shows that LXD creates highly performing writing operations concerning KVM. This performance can be a milestone for VxFs with a high storage rate in Edge Computing, which it can do, certain restrictions are not met. Or even more, writing congestion can cause the CPU to be idle, and the processes do not make progress, degrading the system's entire performance.

Network: Figure 18 shows the number of packets received by each interface. We can see that for both platforms (KVM and LXD), the lo interface (Loop-back) has grown over time. Although for LXD, the growth is dizzyingly greater. This is due to the intensive

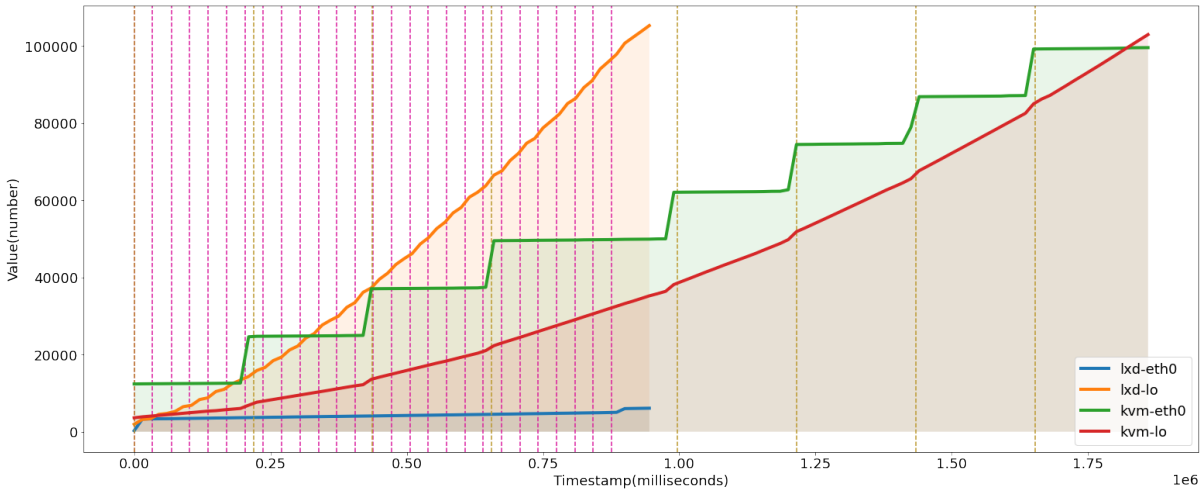

Figure 18 – Network received packets.

use of data for control and management of EclipseFog05 and Zenoh (software responsible for the communication between the parts of EclipseFog05) present in the Raspberry Pi. Besides, the number of VxFs in LXD is strictly higher than in KVM, leading to the conclusion that more information must be exchanged between EclipseFog05 and VxFs for control management. All communication between VxFs and the outside world is done through the eth0 interface on a virtual network. It can be noted that for KVM, there is an increasing number of packets received as new VxFs are launched. The network metrics are the cumulative counter type. There is no receipt of new packages as soon as each of the VxFs is launched in the KVM. The same is true for LXD, although the number of packets received over the network is much smaller. This concludes that for KVM, data packets are received, perhaps by OSM MANO, for the communication, management, and execution of NS primitives.

4.2 Unified Management of Virtualized Functions in Hardware Independent

To show GARNET's ability to manage and orchestrate unified VxFs independent of hardware virtualization, an experimental evaluation was carried out on two different infrastructures, namely a Single Board Computer (AArch64) and a Commodity Server (x86_64). Both infrastructures are capable of providing computational resources to support VxFs. Also, both are indistinguishable for Eclipse Fog05 and therefore for OSM MANO. In this way, regardless of hardware virtualization, it is possible to deploy VxFs seamlessly.

The Raspberry Pi 4 is a B model with a Broadcom BCM2711 system on a chip with a quad-core Cortex-A72 (AArch64) 1.5GHz 64-bit processor, 4 gigabytes of LPDDR4

RAM. The Commodity Server is an Intel Core 2 Quad (x86_64) with 8 gigabytes of RAM. The Raspberry Pi 4 features Ubuntu 18.04.5 LTS arm64 server, with support for hardware virtualization. The Commodity Server uses a 64-bit Ubuntu 18.04.5 LTS (AMD64) desktop image.

All hardware was interconnected through a physical LAN. A router has been configured to support communication between the Virtual Infrastructure Manager (namely Eclipse Fog05), VxFs deployed on the Raspberry Pi 4 or Commodity Server.

4.2.1 Experiment Description

For the present project the Table 4 summarizes the service descriptors used for each of the virtualization platforms, namely, number of virtual CPUs, amount of RAM memory, maximum storage capacity and network interface for management of VxF's internal activities.

Table 4 – Service Descriptors Summary.

Virtualization Platform	**Operational System**	**vCPUs**	**Memory (MB)**	**Storage (GB)**	**Network**
LXD	Alpine Edge	1	256	1	eth0
KVM	CirrOS 0.5.0	1	256	1	eth0

Metrics are the main material processed by monitoring systems to build a cohesive view of monitored systems. Knowing which components are worth monitoring and which specific characteristics to look out for is the first step in designing a system that can provide reliable and actionable insights into the state of software and hardware.

There are several metrics in OpenMetrics format that a unified architecture can provide, such as CPU, memory, disk space, network packets and motherboard temperature. Figure 19 shows the monitoring sequence diagram of the architecture. This model allows for continuous monitoring of the infrastructure. All metrics are extracted via an HTTP request. All metrics are stored as time series for further integration, analysis and visualization.

To analyze the infrastructure's behavior, several Network Services were launched, as much as possible, so that such infrastructure was operational and could respond to new requests. Thus, the launch of any Network Service would not compromise the functioning of an existing one. Therefore, it was always established that, for all cases, the launch of a new Network Service was only possible after the previous service was in an operational state. Table 5 shows the number of VxF by NFVI type.

Table 5 – Number of VxF by NFVI type.

Infrastructure	**Full-Virtualization - KVM**	**Containerization - LXD**
Commodity Server	41	55
Single Board Computer	8	27

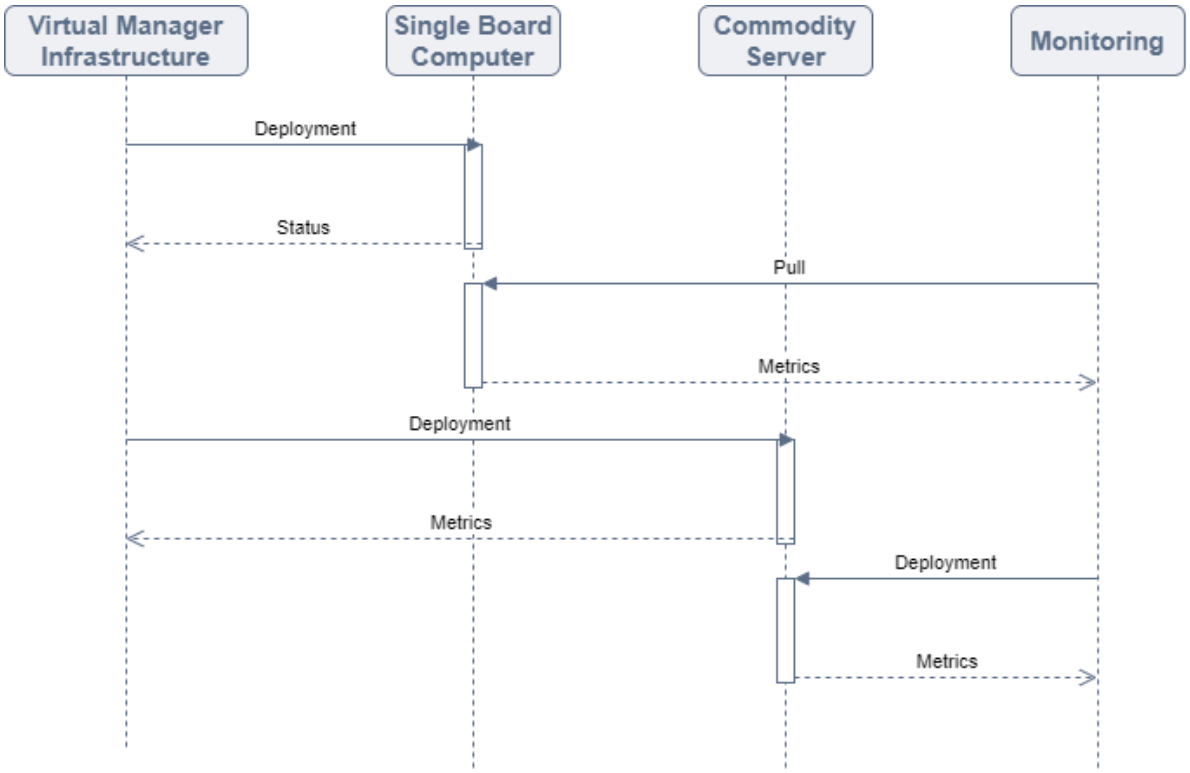

Figure 19 – Sequence diagram for unified monitoring architecture.

Although unified architecture monitoring has a variety of possible metrics, a few metrics were selected for this project. The goal of these metrics is to demonstrate support for VxFs transparently and seamlessly regardless of virtualization hardware. Among the selected metrics are: the time required to deploy a VxF, the amount of network packets sent and received, and the amount of space (in bytes) used by TCP and UDP sockets.

4.2.2 Experiment Analysis

Figure 20 shows the cumulative time to start the VxFs in each of the NFVIs. For this analysis, the Student's t-distribution [1] with a confidence interval of 95% is considered. In both cases, given the number of VxFs per NFVI, the average time to deploy new VxFs by the Single Board Computer proved to be superior compared to Commodity Server, especially using the KVM platform. This is due to the inherent limited resources of the Single Board Computer as well as the high-throughput processing cores present in Commodity Server. On the other hand, it is worth noting that, for both hardware, the deployment time using containerization presents a smaller variation, compared to full-virtualization.

[1] In probability and statistics, Student's t-distribution (or simply the t-distribution) is any member of a family of continuous probability distributions that arise when estimating the mean of a normally-distributed population in situations where the sample size is small and the population's standard deviation is unknown.

Thus, although both infrastructures support VxFs, the form of virtualization (containerization or full-virtualization) and the capacity of the underlying hardware (Single Board Computer or Commodity Server) will directly affect the deployment time of new services and, consequently, their use in a network with virtualized services.

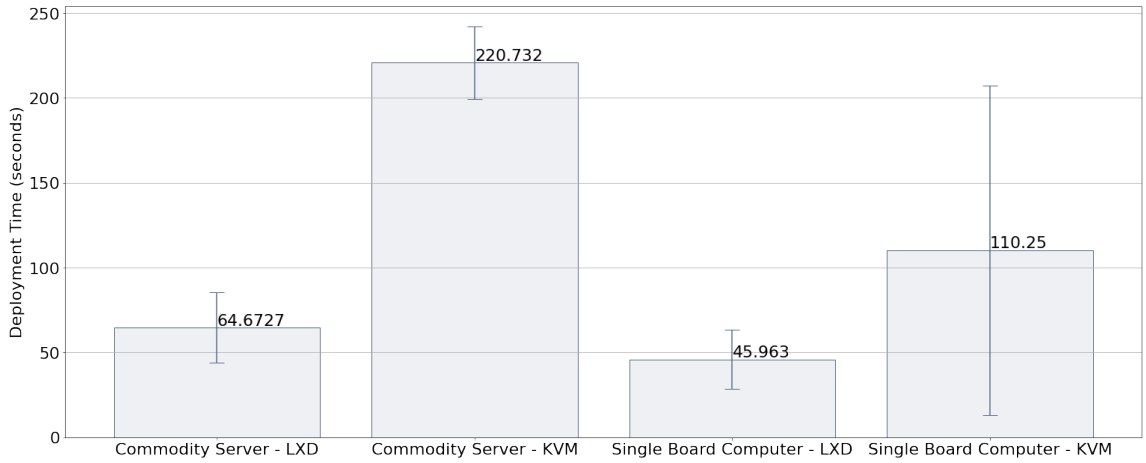

Figure 20 – Virtualized service deployment time.

Figure 21 and Figure 22 show the cumulative amount of network packets received through a network interface. There are two network interfaces on the Single Board Computer, namely wlan0 (infrastructure management) and eth0 (external communication). There are also two network interfaces on the Commodity Server: enp2s0 (infrastructure management) and enp4s0 (external communication). In both cases, you can see an increase in the number of packages received as new VxFs are released. Also in both cases, there is a clear difference between packets received by communication interfaces, using full-virtualization versus containerization. And last but not least, regardless of NFVI, one can notice a similarity in the small amount of packets received by the management interfaces. Therefore, such infrastructures show similarities in their behavior, demonstrating that from a management point of view, regardless of virtualization hardware, they are indistinguishable and usable.

Figure 23 and Figure 24 show the cumulative amount of network packets sent through a network interface. There are two network interfaces on the Single Board Computer, namely, wlan0 (infrastructure management) and eth0 (external communication). There are also two network interfaces on the Commodity Server: enp2s0 (infrastructure management) and enp4s0 (external communication). Again, in both cases, you can see an increase in the number of packets received as new VxFs are released. Also, in both cases, there is a clear difference between packets sent by communication interfaces, using full-virtualization versus containerization. Furthermore, regardless of NFVI, there is a similarity in the small amount of packets sent by the management interfaces, due to the need to respond to requests previously made by the infrastructure manager. Therefore, it

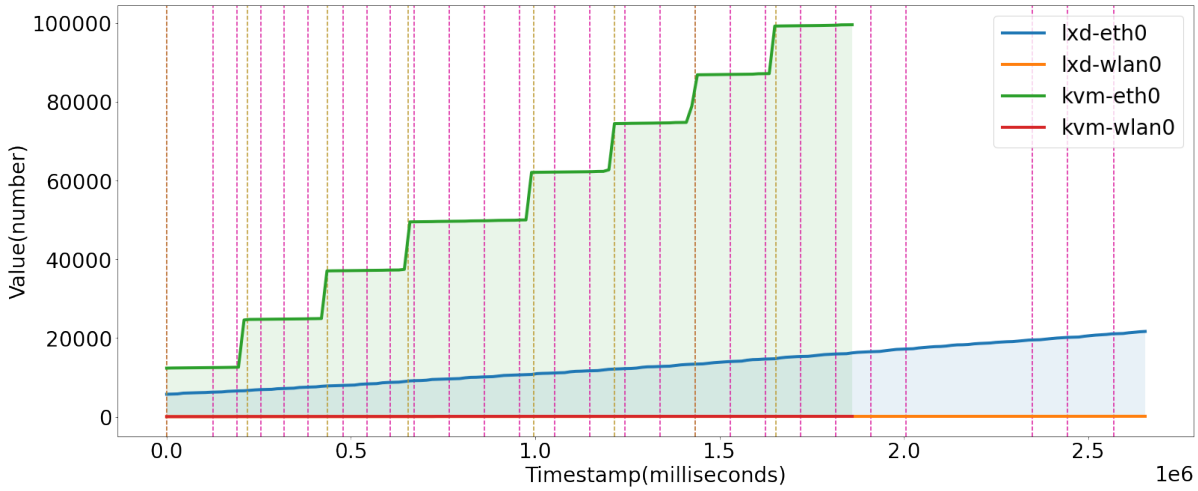

Figure 21 – Cumulative time for packet receipt for each network interface on Single Board Computer.

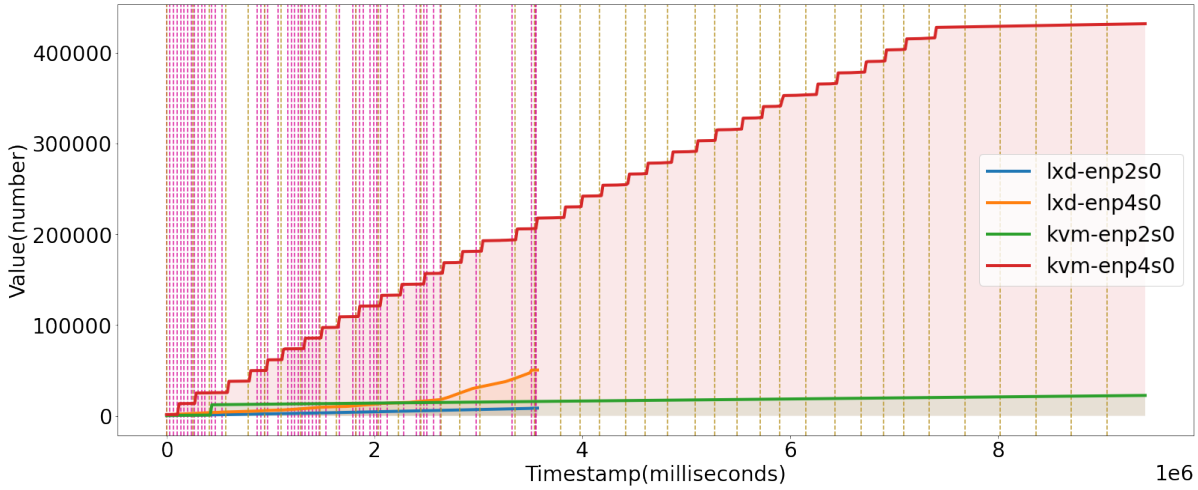

Figure 22 – Cumulative time for packet receipt for each network interface on Commodity Server.

is noted that such infrastructures have similarities in their behavior, creating a unified and homogeneous view of the entire infrastructure, regardless of the virtualization hardware supporting VxFs.

Figure 25 and Figure 26 show how many bytes network sockets are using at any given time. In both cases, we see a higher usage of bytes in TCP-type sockets than sockets in UDP, either in Single Board Computer or Commodity Server. Furthermore, it is noteworthy that, despite small variations and outliers, neither of the two cases had such discrepant usage levels, on the contrary, they behaved similarly in the use of network sockets presenting a limit close to 100,000 bytes. This shows that, regardless of the type of virtualization hardware (AArch64 or x86_64), and virtualization platform (containerization or full-virtualization), the network resources required for managing and

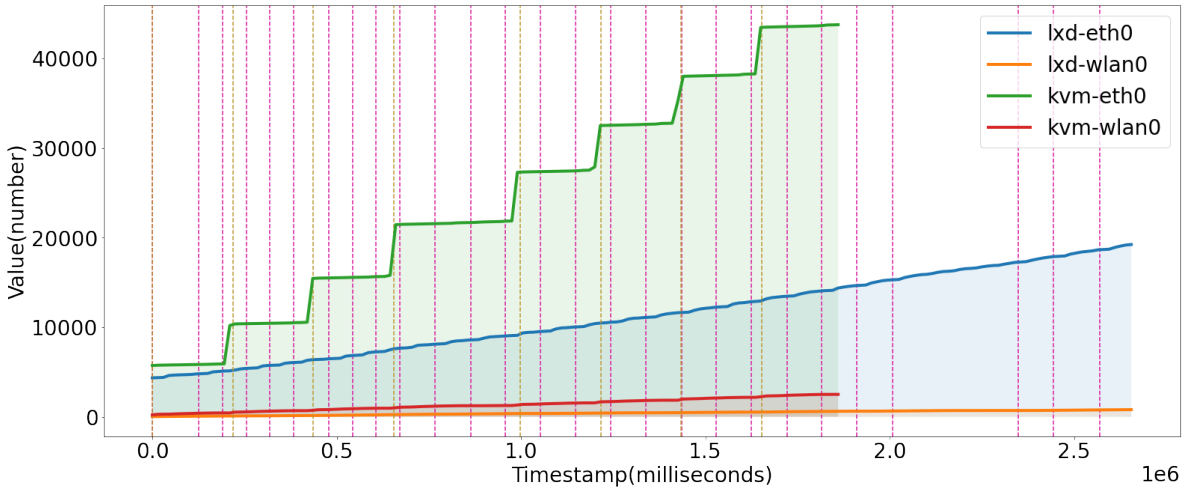

Figure 23 – Cumulative time for sending packets through each network interface on Single Board Computer.

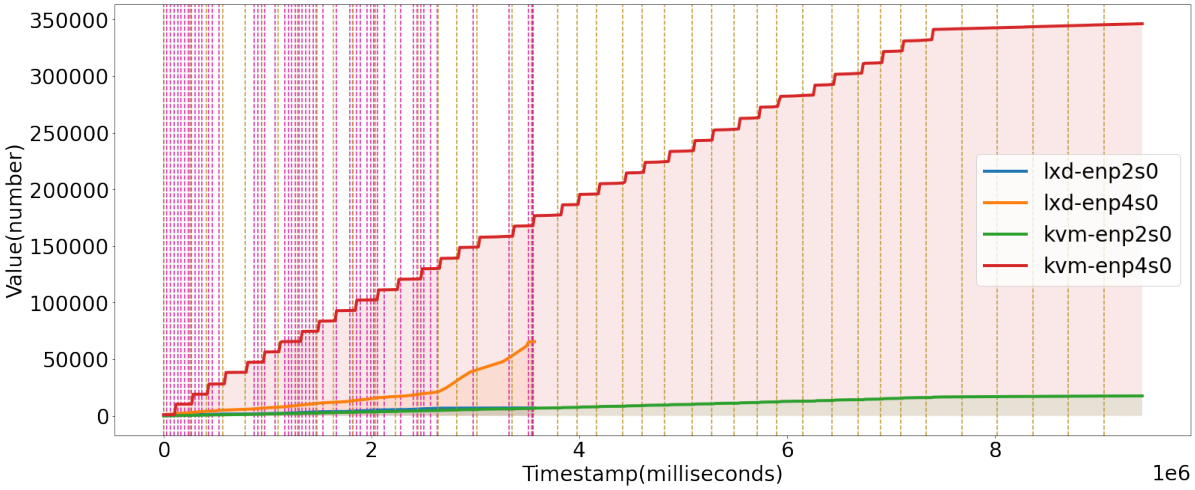

Figure 24 – Cumulative time for sending packets through each network interface on Commodity Server.

supporting virtualized functions are simple and indistinguishable from the point of view of management.

4.3 The Full Monitoring of Virtualized Functions in Multi-Access Edge Computing

To show the full monitoring capability of GARNET in Multi-Access Edge Computing, an experimental evaluation was conducted. The experiment includes a Raspberry Pi 4 and a bare metal desktop computer acting in a complementary way. The bare metal desktop, representing a highly provisioned Cloud Computing with high storage and processing

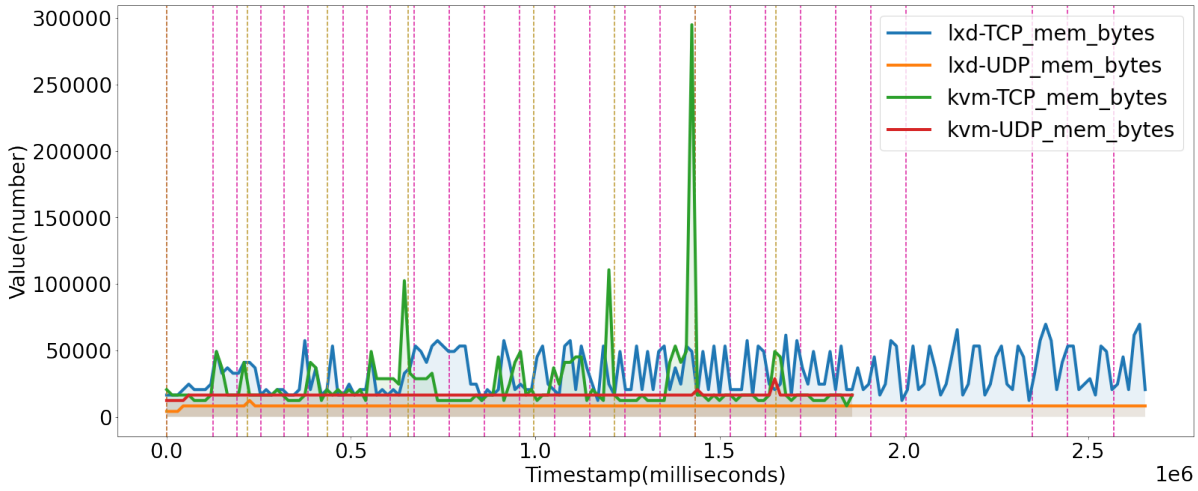

Figure 25 – Number of bytes in network sockets on a Single Board Computer

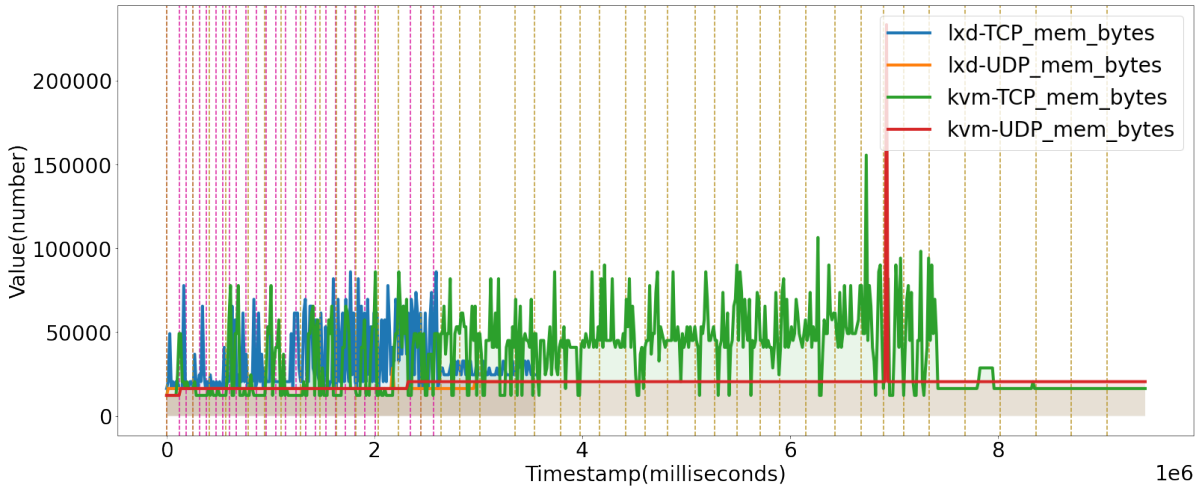

Figure 26 – Number of bytes in network sockets on a Commodity Server.

capacity. On the other hand, the Raspberry Pi 4 represents infrastructure in Multi-Access Edge Computing, with limited resources in processing and storage, yet capable of providing enough resources to support VxFs. It is noteworthy that any AArch64 or x86_64 device with virtualization capability can be used in place of the Raspberry Pi 4, so the experimental setup here replicated using several other platforms commonly used in the Multi-Access Edge Computing.

The Raspberry Pi 4 is a Model B with a quad-core Cortex-A72 (AArch64) 64-bit 1.5 GHz processor, 4 gigabytes of RAM. The bare metal desktop computer is an Intel Core 2 Quad with 8 gigabytes of RAM. The Raspberry Pi 4 has an Ubuntu 18.04.5 LTS arm64 server, with support for hardware virtualization. The bare-metal desktop computer uses a Ubuntu 18.04.5 LTS 64-bit (AMD64) desktop image.

All the hardware was interconnected using a local physical network. The network's

router was configured to support the communication between the bare metal desktop computer, the VxF on Raspberry Pi 4, and the external world.

4.3.1 Experiment Description

To show the full monitoring capability of GARNET, an experimental evaluation of API Gateway, deployed in the as a VxF, was conducted. Figure 27 shows how the experiment works, as well as each of its components and their locations.

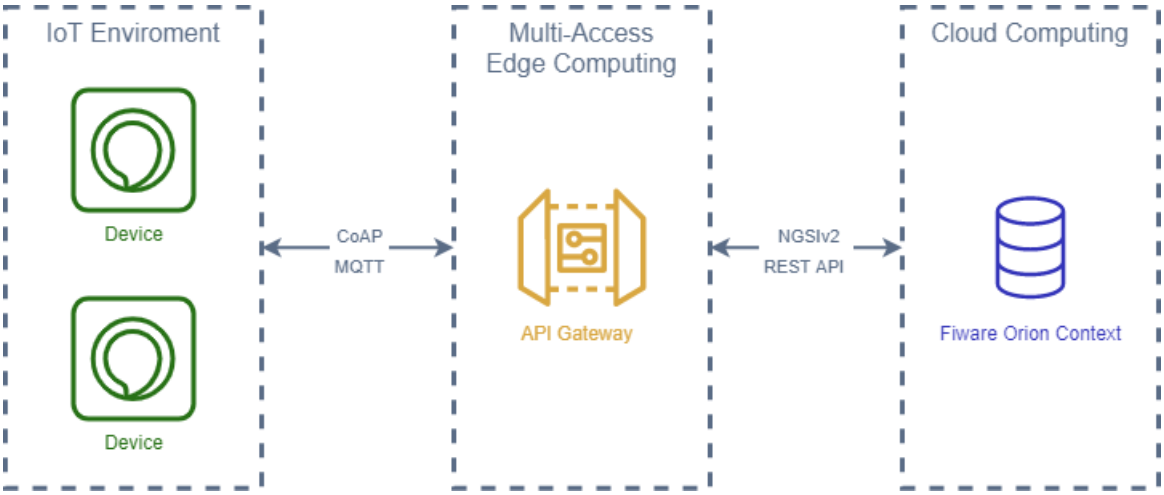

Figure 27 – An API Gateway deployed in Multi-Access Edge Computing as a VxF.

The IoT device is manufactured by EnergyNow Technologies and is called Prodbox, which operates as a device used to intensify productivity and point out strategic ways to modify variables that interfere in the management view of production. The device uses unobtrusive sensors to count the number of items that cross the detection line generated between the installed transmitter and receiver. The data generated through the device can be used to better understand variables about the pace of production and, based on them, promote production projections, calculating the relationship between items produced and the period of time required (seconds, minutes, hours, days , weeks, etc).

The API Gateway, deployed in Multi-Access Edge Computing as a VxF, is an API management tool that sits between the client and a back-end service. It works as a reverse proxy server [2], which accepts all API calls, aggregates the various services needed to perform them, and returns the appropriate result. Protocol gateways, as used in the experiment, are small software that function primarily as translators for various protocols so that distributed systems can communicate interchangeably. For the present project,

[2] A proxy server is a go-between or intermediary server that forwards requests for content from multiple clients to different servers across the Internet. A reverse proxy server is a type of proxy server that typically sits behind the firewall in a private network and directs client requests to the appropriate back-end server. A reverse proxy provides an additional level of abstraction and control to ensure the smooth flow of network traffic between clients and servers.

communication is made between IoT devices, which use MOM protocols such as CoAP and MQTT, and Cloud Computing, which uses RESTful protocols such as NGSIv2 REST API.

The Fiware Orion Context service (SALHOFER, 2018), deployed in Cloud Computing, allows management of the entire life-cycle of context information, including updates, queries, registrations and subscriptions. It is a server implementation to manage context information and its availability. Using the Orion Context Broker, you can create context elements and manage them through NGSIv2 REST API (DOYLE; COSGROVE, 2019) queries and updates. In addition, it is possible to sign context information so that when some condition occurs (eg context elements change), entire systems can be notified.

Due to the difficulties of real experimentation using a Prodbox in a production chain, a data set (ALVES, 2020) was used, containing information to simulate a production chain during the 12 hour interval. All records are sent to API Gateway through two protocols: MQTT and CoAP. Here is a list with the details of the fields present in the data set:

- ❏ **ID**: Sequential generic identifier of the record in the database.
- ❏ **ProductionLineID**: Identifier of the production line where the data was collected.
- ❏ **DeviceID**: Identifier of the device to which the record refers.
- ❏ **Count**: "0" signals that there was no item crossing the line of production in the last moments. Therefore, it can represent a downtime of the production line. "1" signals the item identification by traversing the line of production at the time of registration.
- ❏ **CreatedAt**: Time when the registration occurred.

API Gateway, deployed in Multi-Access Edge Computing as a VxF, is proprietary software packaged in two formats: container using LXD and as a virtual machine using KVM. The API Gateway protocol translation in virtualized environment was tested. This activity consists of forwarding data received from IoT devices through MQTT and CoAP protocols to the cloud using the NGSIv2 REST API protocol. The Fiware Orion Context, deployed in Cloud Computing, is a third-party application deployed on a bare-metal computer as a containerized service.

Although full GARNET monitoring in Multi-Access Edge Computing generates a variety of possible metrics in OpenMetrics format, a few metrics were selected for this project, namely: time required to process requests and responses, how many CPU seconds are spent in kernel mode and user mode , the total time in seconds that a process cannot progress due to memory congestion, the cumulative time spent on I/O operations, the Round-Trip Time (RTT) of a request/response measured in seconds and the cumulative amount of packets sent/received through each of the interface. The purpose of these

metrics is to demonstrate the ability to monitor the VxF deployment hardware, as well as the virtualized environment and the deployed service itself. In this way, metrics at different levels can be correlated to gain a better understanding of service provisioning and infrastructure management.

4.3.2 Experiment Analysis

As for the present experiment it is possible to use two virtualization platforms (LXD and KVM), as well as using two types of protocols for transporting data from the IoT environment (MQTT and CoAP), a total of four use cases were tried. For each of the use cases, an analysis of the collected metrics as well as their correlations was performed.

4.3.2.1 Using MQTT protocol and LXD virtualization platform

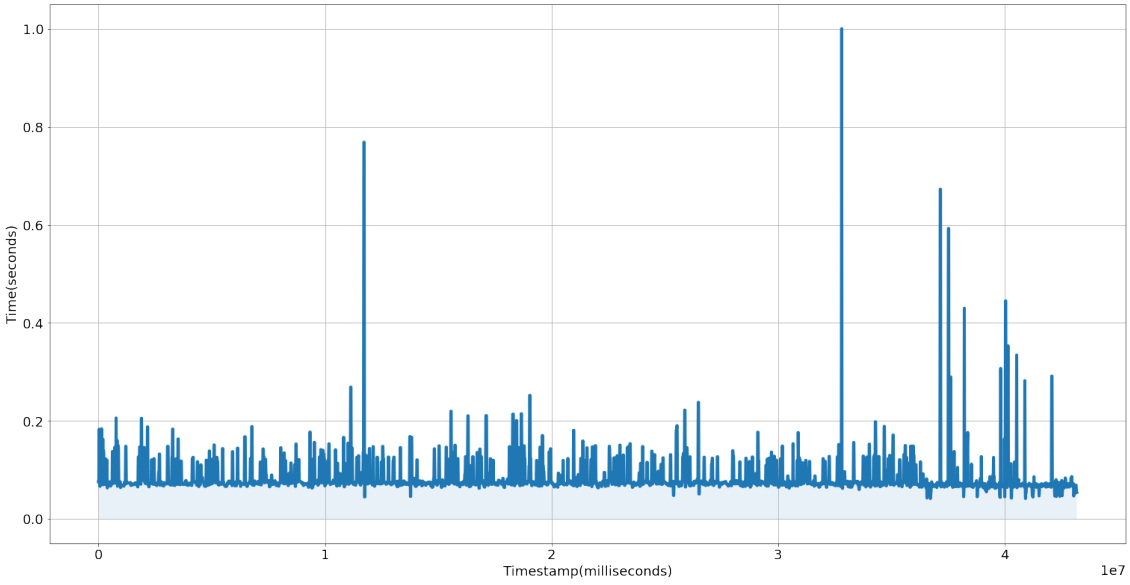

Figure 28 – The amount of time required to process a request.

Request and Response: Figure 28 and Figure 29 shows the time required to process requests and responses, respectively. It can be noted that despite the outliers and small variations, the processing time of requests and responses remained constant over time. Also, it is important to note that the time to process a request is much longer than the time to process a response. This is due to the need for API Gateway, deployed over Multi-Access Egde Computing in VxF form, to store the mapping between requests in asynchronous responses.

CPU: Figure 30 and Figure 31 shows how many CPU seconds are spent in kernel mode and user mode, respectively by the container and the host machine's operating system. It can be noted that the time spent in user mode of the container, which is the only one in

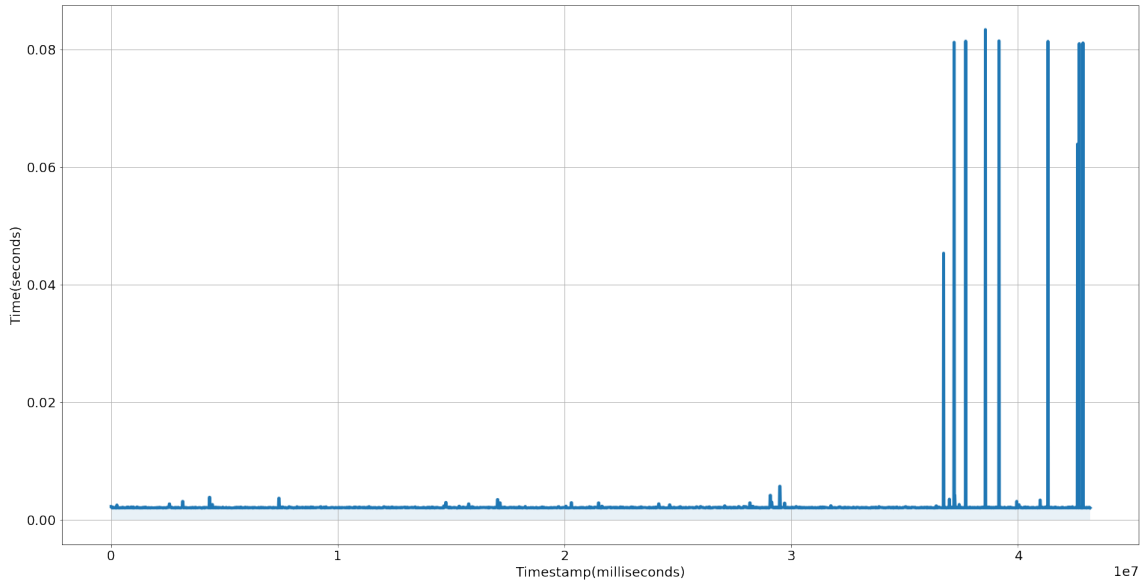

Figure 29 – The amount of time required to process a response.

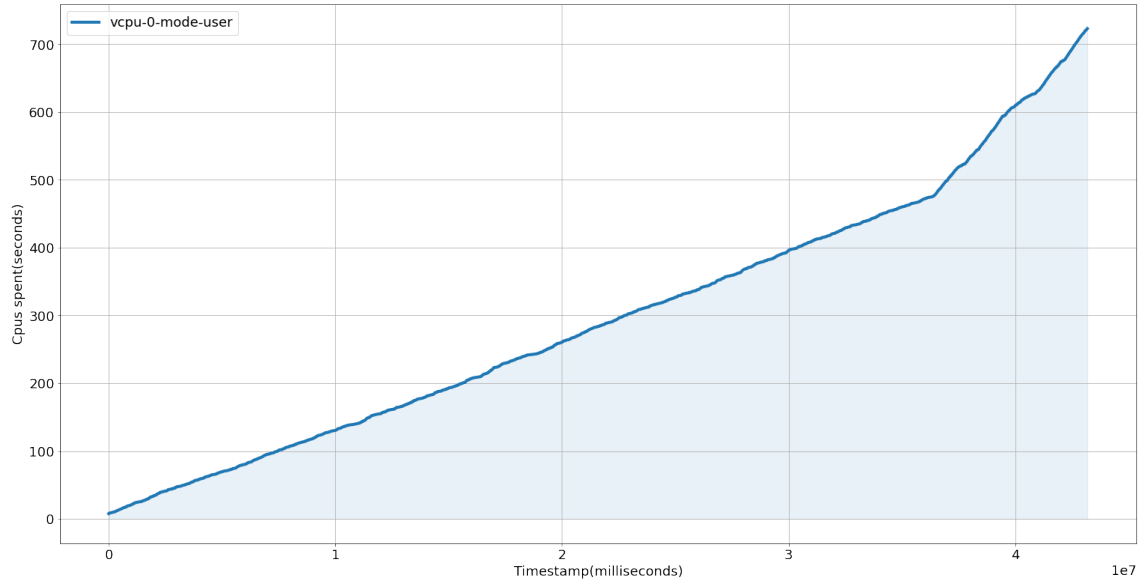

Figure 30 – Seconds the virtual cpus spent in each mode.

a virtualized environment, is similar to the time spent in user mode of the host machine's operating system. This is due to the fact that containerization has part of its operation tied to the operating system of the host machine.

Memory: Figure 32 and Figure 33 shows the total time in seconds that a process cannot progress due to memory congestion, respectively by the container and the host machine's operating system. It can be noted that the bottleneck of container processes is similar to the bottleneck of processes on the host machine. Again, this is due to the fact that the containerization has part of its operation linked to the host machine's operating

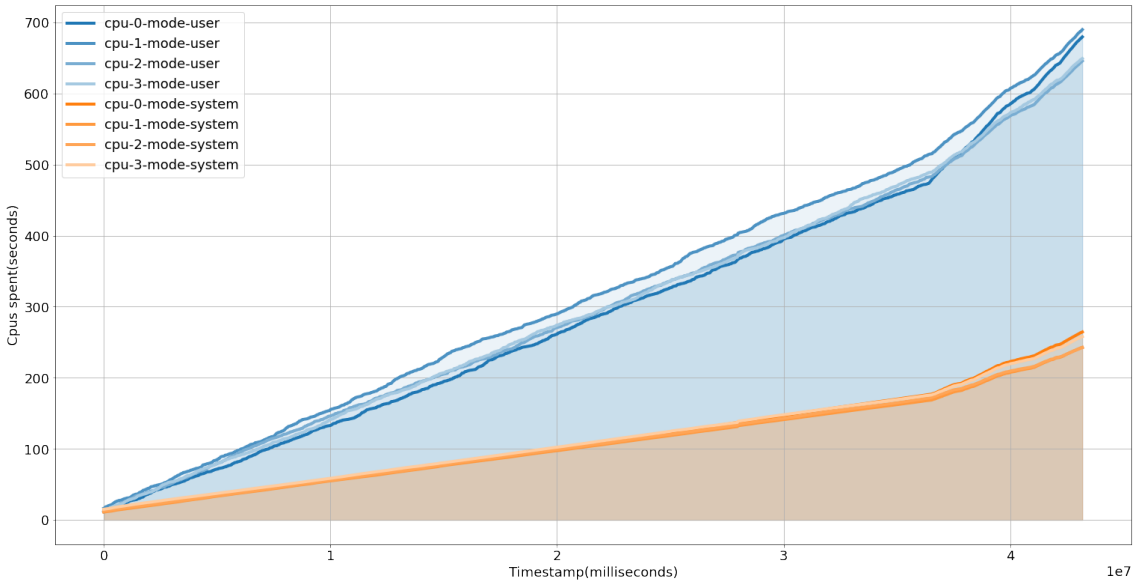

Figure 31 – Seconds the cpus spent in each mode.

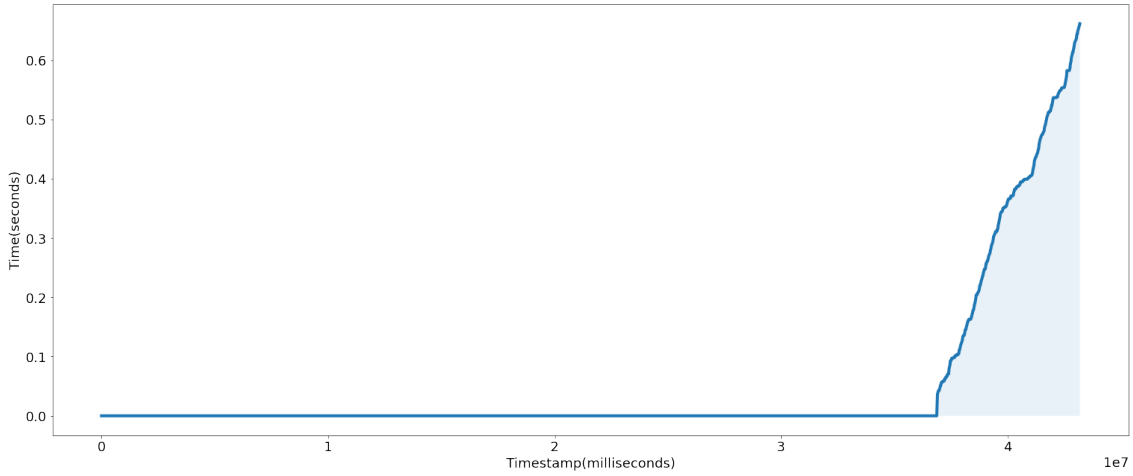

Figure 32 – Total process congestion time in the container.

system, and often the container is considered just a host machine process. The increase in congestion caused at the end of the experiment is due to the decrease in available memory due to the accumulation of requests and responses over time. In this way, once noticed, preventive actions can be taken so that such congestion does not happen again, and thus, guaranteeing quality of service.

Disk: Figure 34 and Figure 35 shows the time taken by all disk write operations, respectively by the container and the host machine's operating system. It can be noted that time spent on write operations on the container is similar to time spent on write operations on the host machine. Again, this is due to the fact that containerization has part of its operation tied to the host machine's operating system, and often the

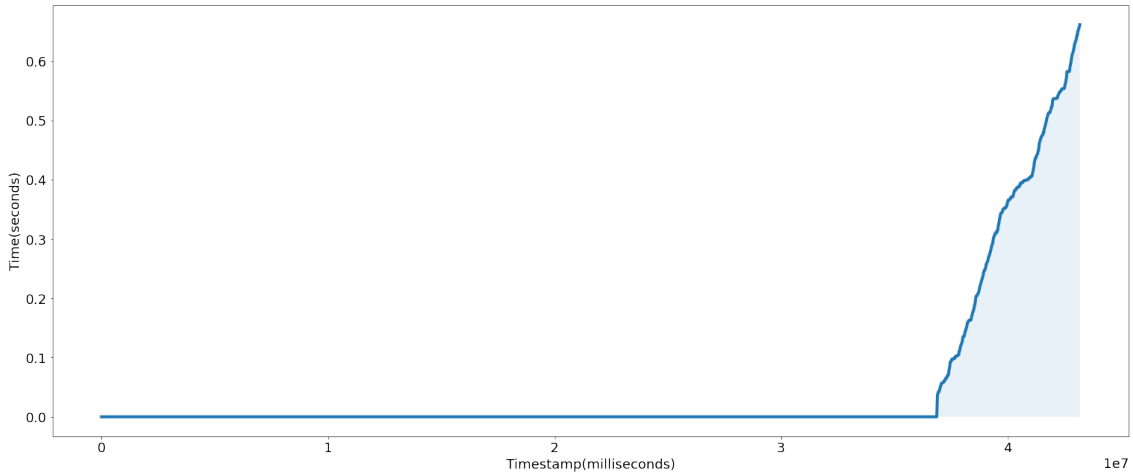

Figure 33 – Total process congestion time in the infrastructure.

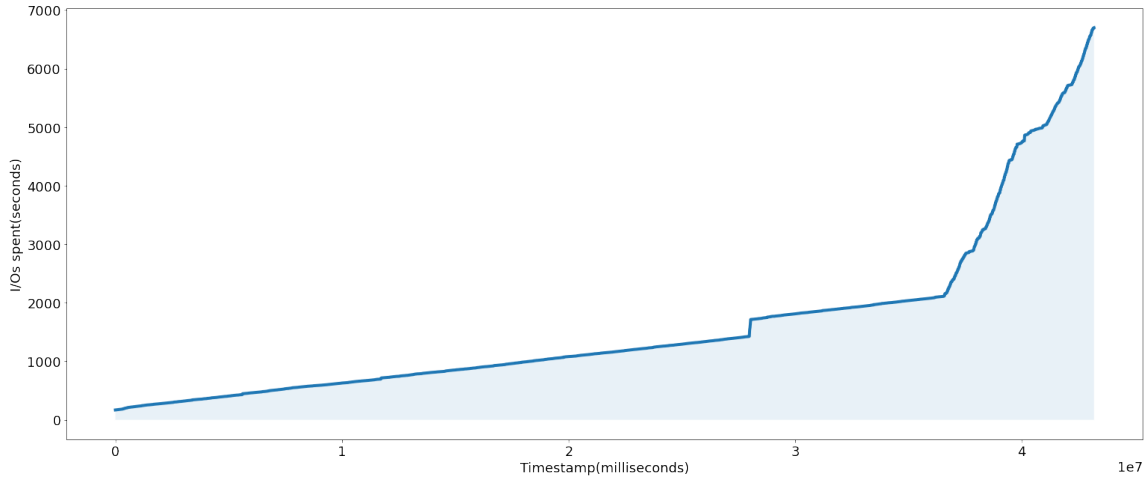

Figure 34 – Total time spent on disk I/O operations in the container.

container is considered just a host machine process. The increase in the time spent on I/O operations caused at the end of the experiment may be linked to the decrease in available memory, as demonstrated above. As a result, several pages of memory must be written back to disk before they can be freed for use. This can be a bottleneck in API Gateway performance, perhaps in critical moments of operation in the supply chain. Again, once noticed, preventive actions can be taken so that such a bottleneck does not happen, thus ensuring the quality of the service.

Round-Trip Time: Figure 36 shows the RTT measured in seconds, from the time a request is sent to the Fiware Orion Context to the time a response is received by the API Gateway. It can be noted that despite the outliers and small variations, the Round-Trip Time remained constant over time. This could perhaps be changed if the API Gateway, in the form of VxF, could migrate throughout the experiment, to different infrastructures and still keep the service in a working state.

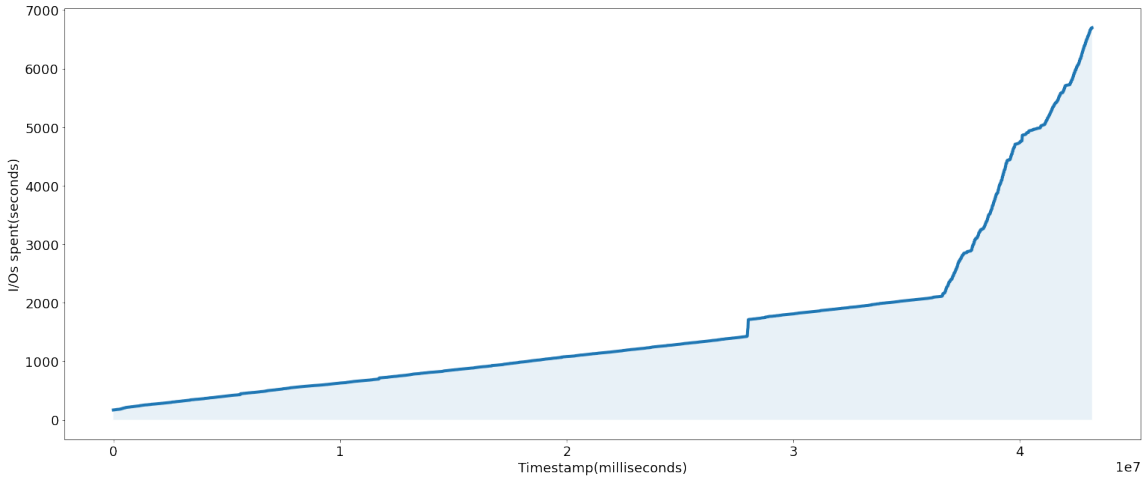

Figure 35 – Total time spent on disk I/O operations in the infrastructure.

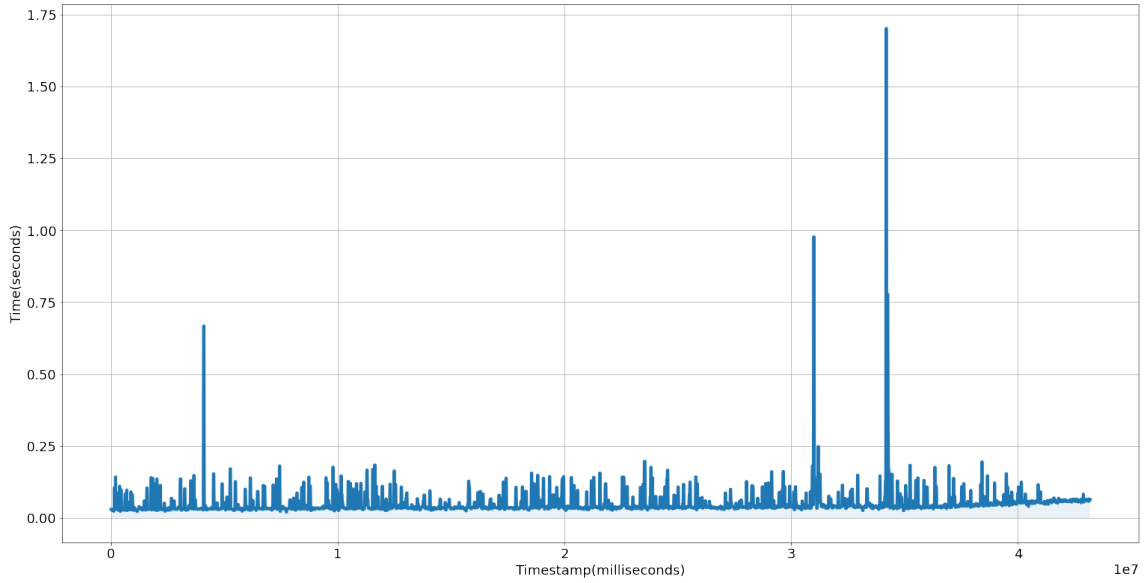

Figure 36 – The Round-Trip Time from API Gateway to Fiware Orion Context.

Network Transmit Packets: Figure 37 and Figure 38 shows the cumulative amount of packets sent through each of the interfaces on the container and on the host machine, respectively. In the container environment, there are two network interfaces, namely lo (environment management) and eth0 (external communication). In this case, more packets are sent through the management interface than through the external communication interface, this is due to the communication between processes that make up the API Gateway for correct functioning. In the case of infrastructure, there are two network interfaces, namely lo (infrastructure management) and wlan0 (external communication). In this case, there is a greater sending of packets through the external communication interface than through the management interface, this evidently collaborates with the need

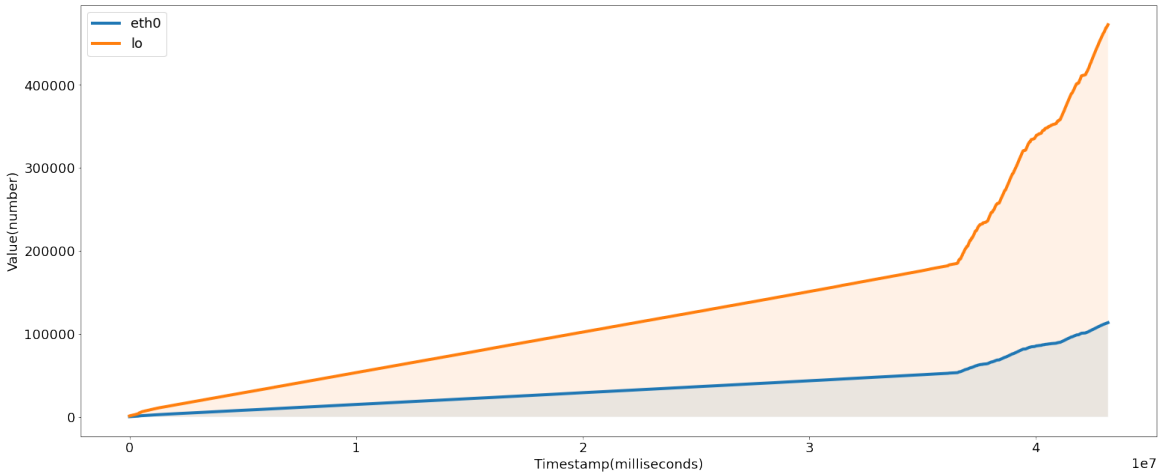

Figure 37 – Cumulative number of packets sent by each container interface.

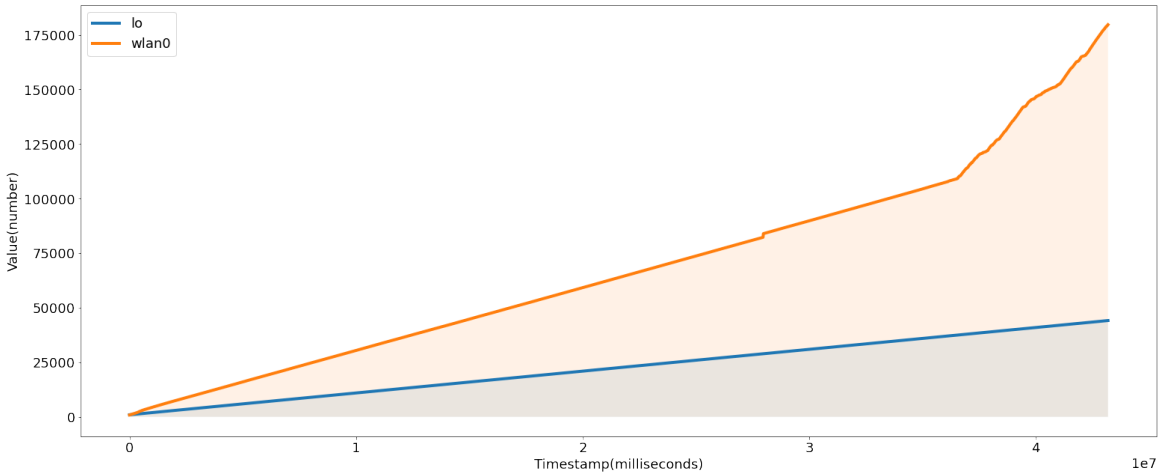

Figure 38 – Cumulative number of packets sent by each infrastructure interface.

for the API Gateway, deploying on the Multi-Access Edge Computing in the form of VxF, needing to send information to the Fiware Orion Context, located in Cloud Computing.

Network Received Packets: Figure 39 and Figure 40 shows the cumulative amount of packets received through each of the interfaces on the container and on the host machine, respectively. In the container environment, there are two network interfaces, namely lo (environment management) and eth0 (external communication). In this case, there is a greater reception of packets through the management interface than through the external communication interface, this is due to responses between the processes that make up the API Gateway for correct functioning. In the case of infrastructure, there are two network interfaces, namely lo (infrastructure management) and wlan0 (external communication). In this case, there is a greater reception of packets through the external communication interface than through the management interface. This evidently collaborates with the need for API Gateway, deploying on Multi-Access Edge Computing in VxF form, needing

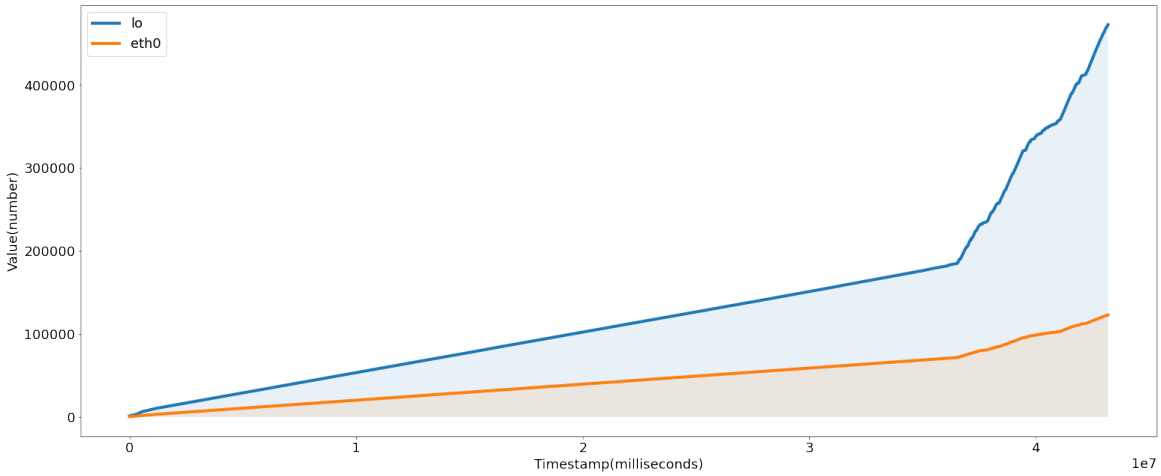

Figure 39 – Cumulative number of packets received by each container interface.

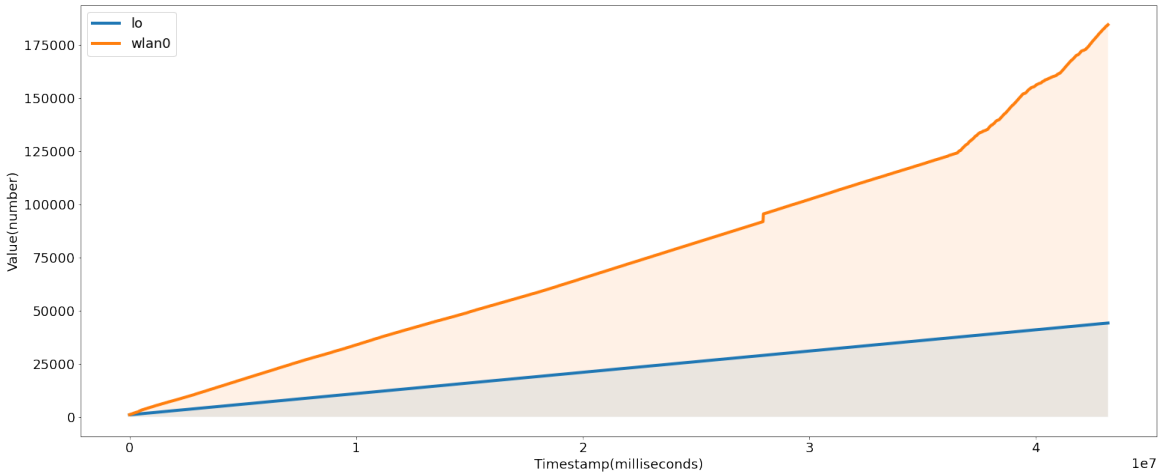

Figure 40 – Cumulative number of packets received by each infrastructure interface.

to receive information from requests previously made to the Fiware Orion Context, located in Cloud Computing.

4.3.2.2 Using MQTT protocol and KVM virtualization platform

Request and Response: Figure 41 and Figure 42 show the time required to process requests and responses, respectively. It can be noted that despite the outliers and there was a wide variation in the processing time of requests and responses over time. Also, it is important to note that the time to process a request is much longer than the time to process a response. This is due to the need for API Gateway, deployed over Multi-Access Edge Computing in VxF form, to store the mapping between requests in asynchronous responses.

CPU: Figure 43 and Figure 44 shows how many seconds of CPU are spent in kernel

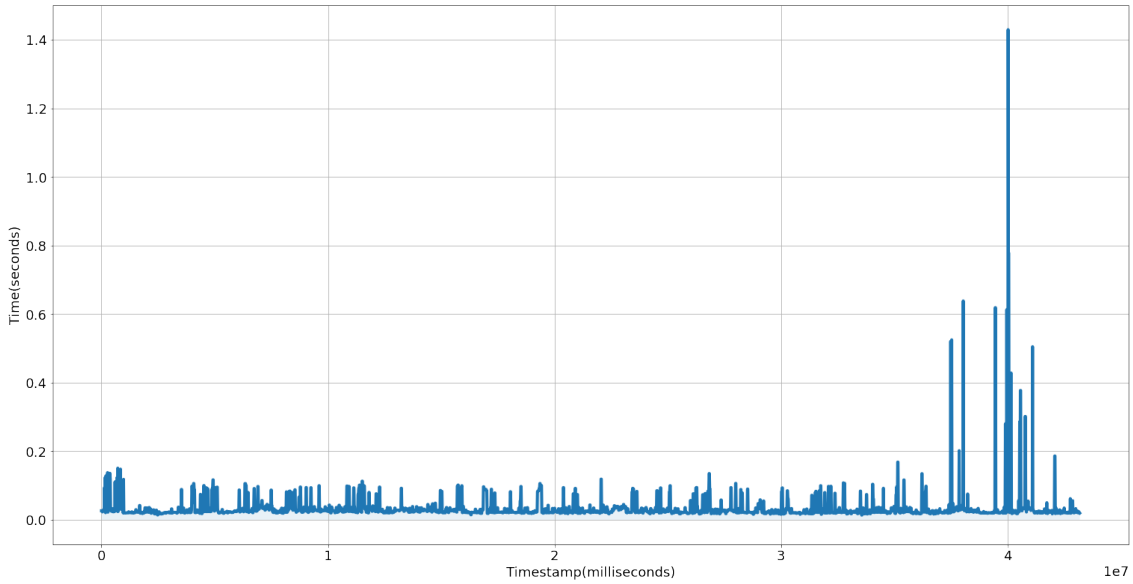

Figure 41 – The amount of time required to process a request.

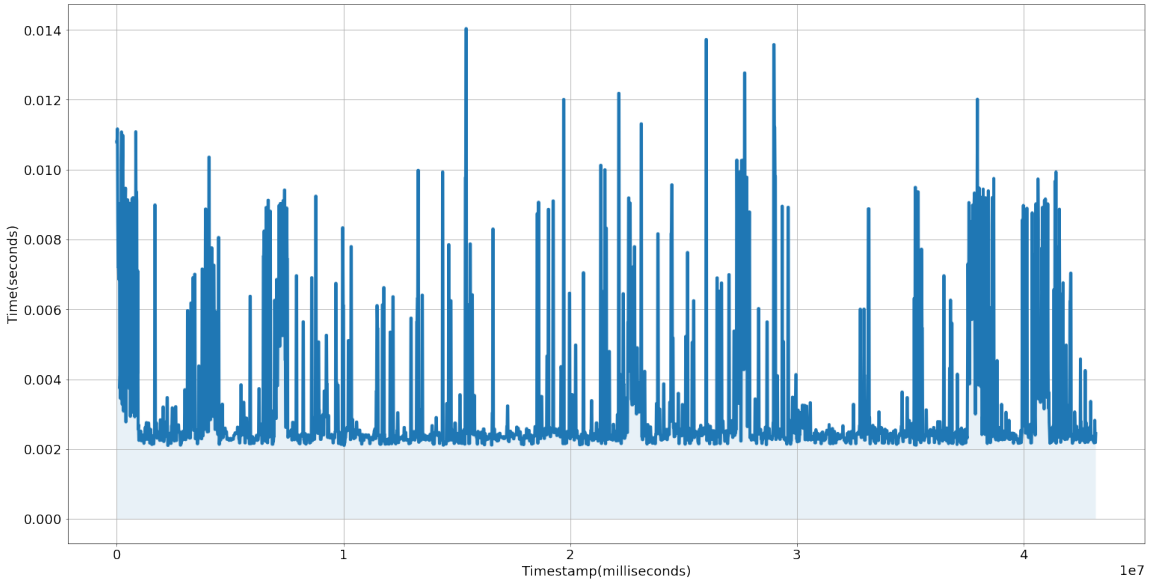

Figure 42 – The amount of time required to process a response.

mode and user mode, respectively, by the virtual machine and the host machine's operating system. It can be noted that the time spent in user mode of the virtual machine is greater than the time spent in kernel mode. Thus, the Gateway API deployed in a virtual machine, using the MQTT protocol, presents an optimization in relation to the overhead caused between context switches between user mode and kernel mode. Likewise, it can be seen that the time spent in user mode on the host machine's operating system is much greater than the time spent in kernel mode. This of course supports the idea that the API Gateway's virtualized environment runs on top of the user-mode infrastructure, not

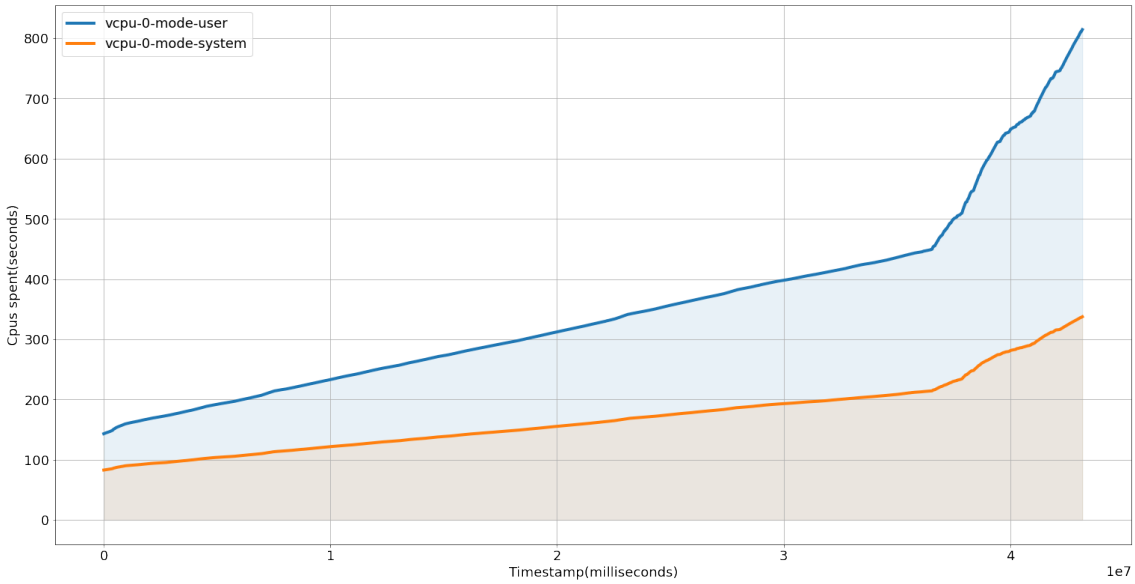

Figure 43 – Seconds the virtual cpus spent in each mode.

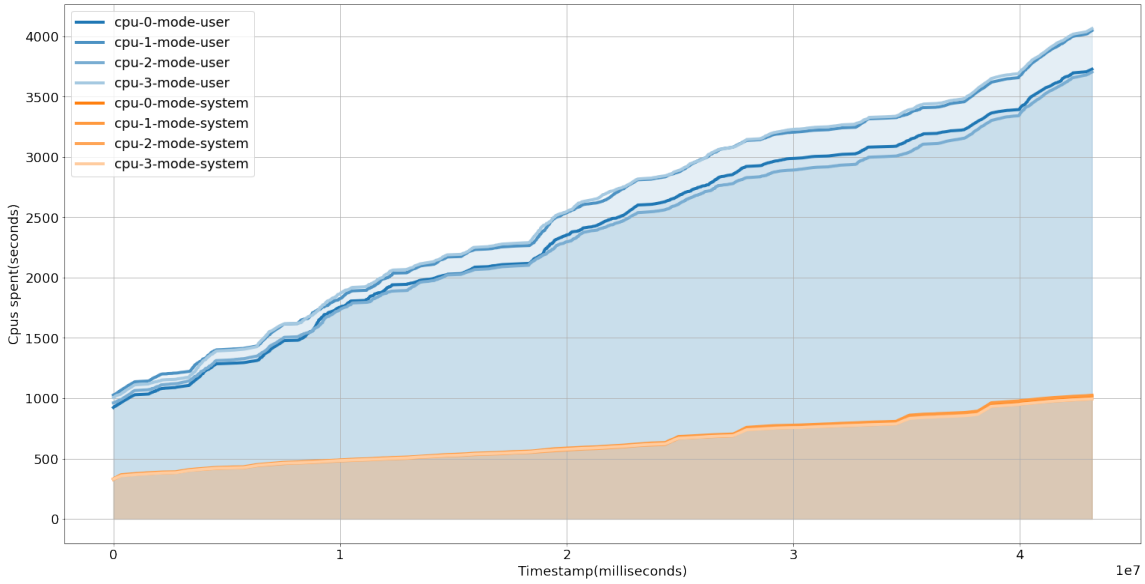

Figure 44 – Seconds the cpus spent in each mode.

kernel-mode. In this way, all context and message exchanges between the VxF operating system are fully controlled and decoupled from the host machine.

Memory: Figure 45 and Figure 46 shows the amount of RAM memory available, respectively by the virtual machine and the host machine's operating system. It can be noticed that in both cases, as the experiment was executed, there was a decrease in the amount of available ram. Also, it is noteworthy that there are slight variations in the amount of memory available in the infrastructure than in the virtual machine. This is likely due to processes present on the host machine, which compete with the virtualized

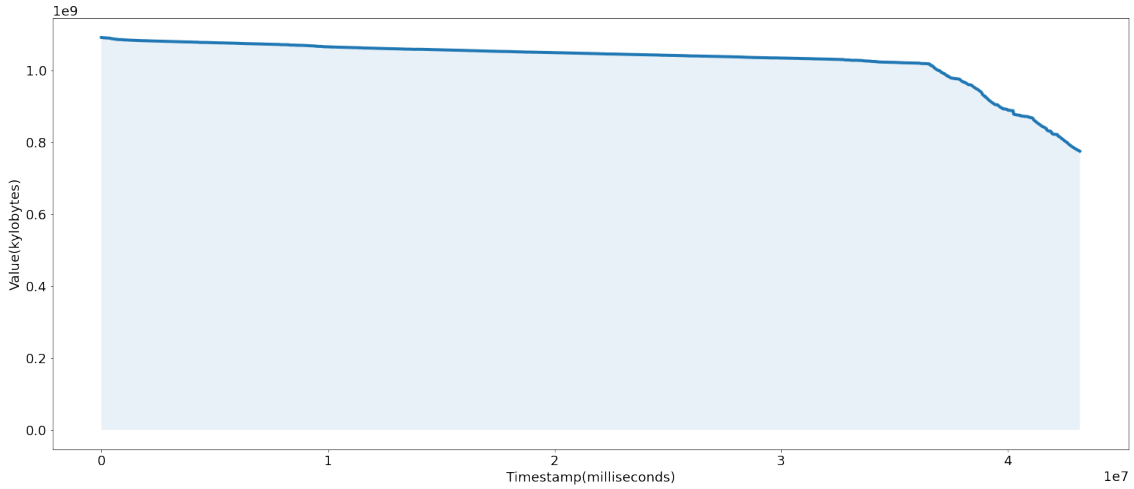

Figure 45 – The amount of RAM available on the virtual machine.

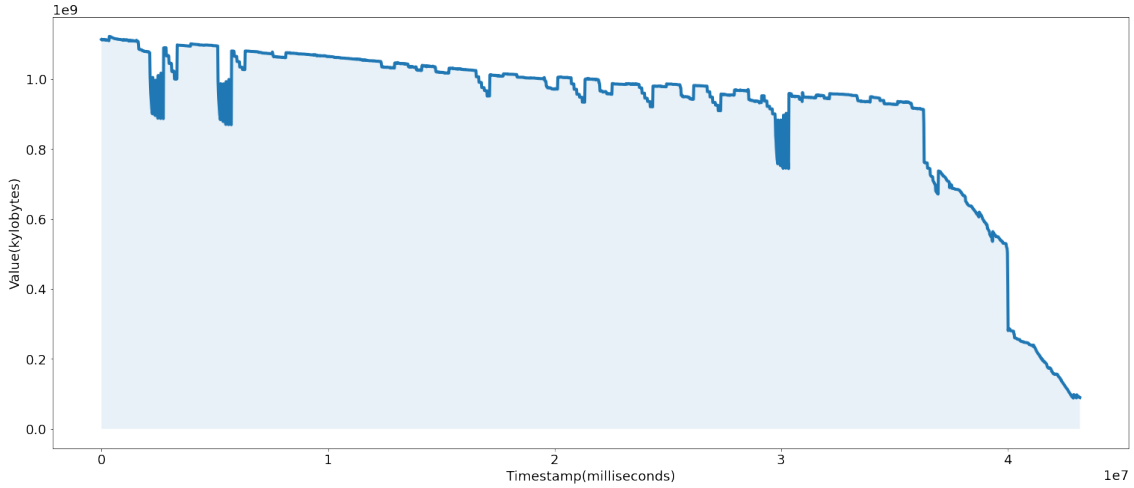

Figure 46 – The amount of RAM available on the infrastructure.

environment. In this way, it is possible to verify the need for corrective actions for the correct functioning of the VxF, in the absence of memory for its functioning, or in the possibility of implementing new collaborative VxFs.

Disk: Figure 47 and Figure 48 shows the cumulative time spent on disk I/O operations by the virtual machine and the host machine's operating system respectively. It can be noted that in both cases, as the experiment was performed, there was an increase in the time spent on disk I/O operations, although in the case of infrastructure this was greater. This is due to the amount of processes existing in the host machine for its functioning. However, it is noteworthy, that at the end of the experiment, you can see a close relationship between the increase in time spent on disk I/O operations in both the virtualized environment and the infrastructure. In this way, it is possible to verify the reasons for infrastructure degradation and its impacts on other VxFs.

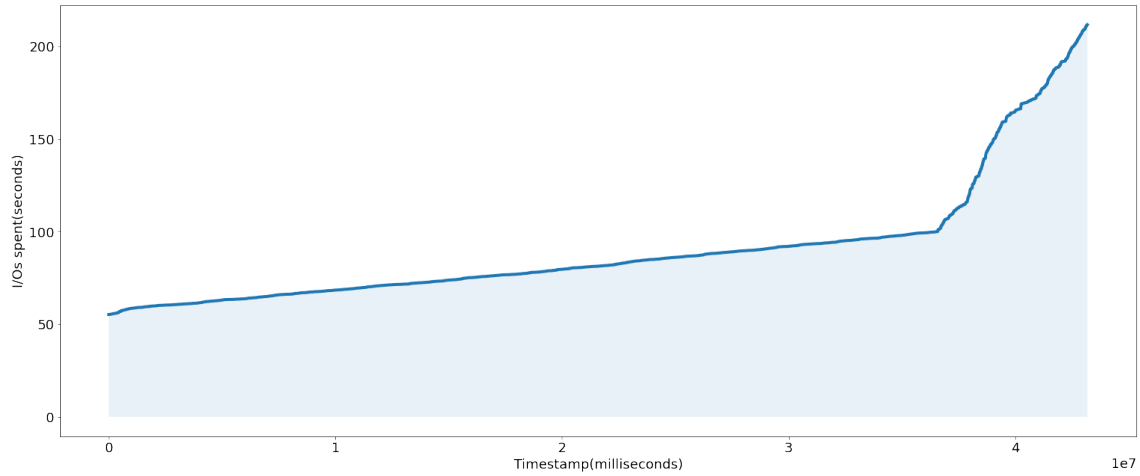

Figure 47 – Total time spent on disk I/O operations in the virtual machine.

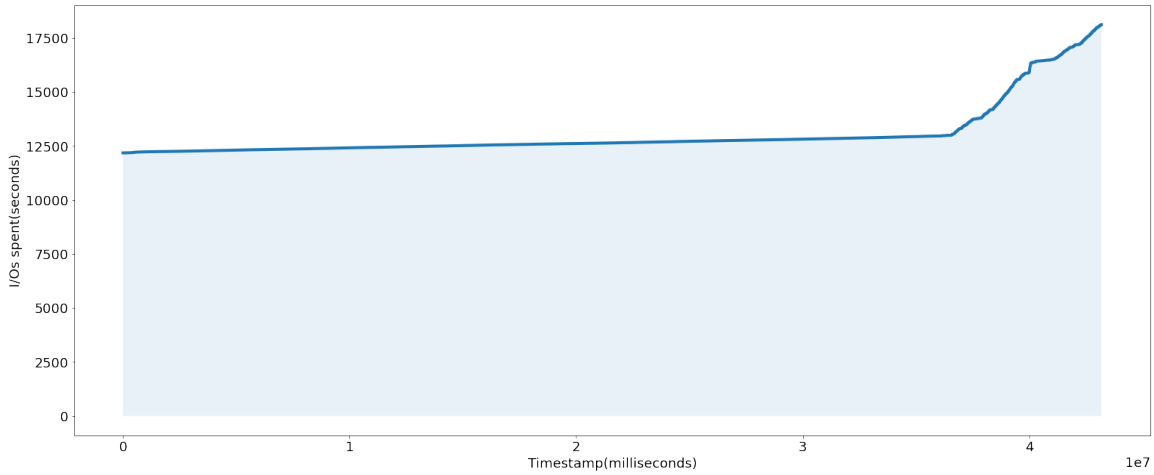

Figure 48 – Total time spent on disk I/O operations in the infrastructure.

Round-Trip Time: Figure 49 shows the RTT measured in seconds, from the time a request is sent to the Fiware Orion Context to the time a response is received by the API Gateway. It can be noted that there was a large variation in Round-Trip Time over time. This is perhaps due to the large overhead caused by the additional layer between the virtualization environment, the virtualization layer, and the host machine. In addition, Full-Virtualization is highly decoupled from the underlying infrastructure, causing large context switches and thus increasing processing time.

Network Transmit Packets: Figure 50 and Figure 51 shows the cumulative amount of packets sent through each of the interfaces on the container and on the host machine, respectively. In the virtualized environment, there are two network interfaces, namely lo (environment management) and enp1s0 (external communication). In this case, more packages are sent through the management interface than through the external communication interface, this is due to the communication between processes that make up the

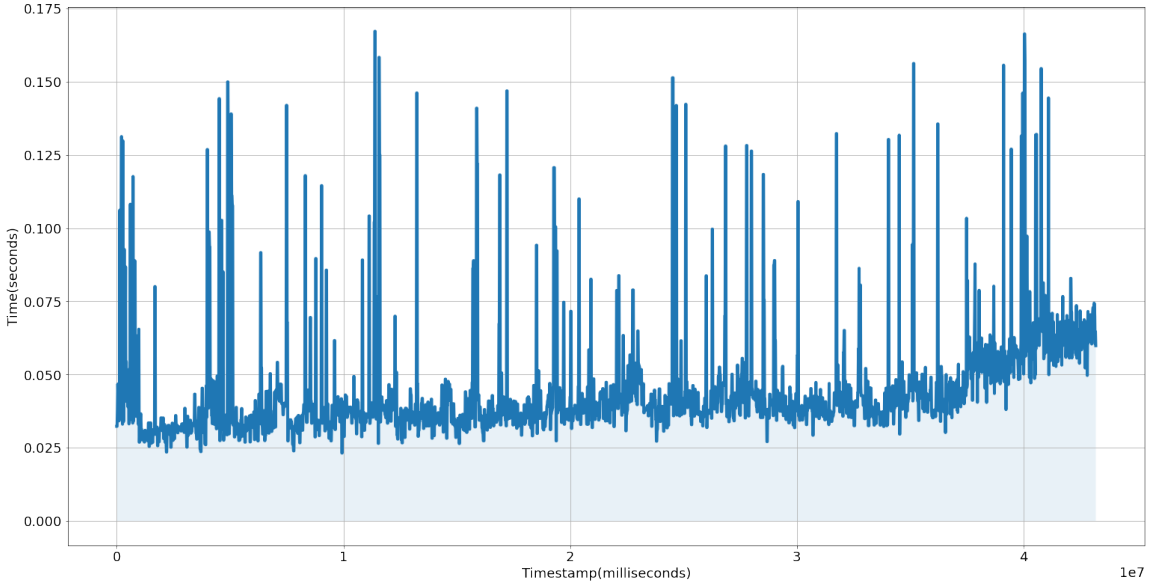

Figure 49 – The Round-Trip Time from API Gateway to Fiware Orion Context.

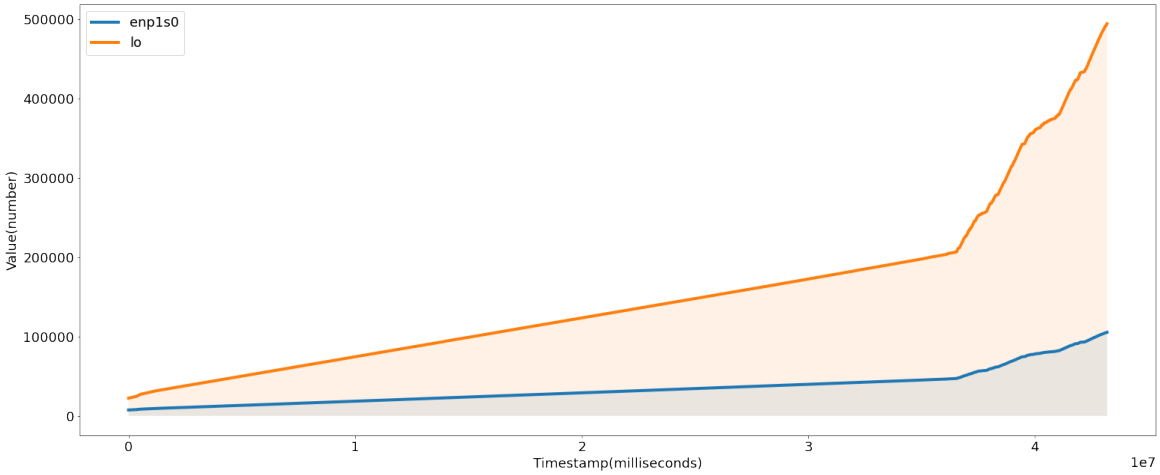

Figure 50 – Cumulative number of packets sent by each virtual machine interface.

API Gateway for correct functioning. In the case of infrastructure, there are two network interfaces, namely lo (infrastructure management) and wlan0 (external communication). In this case, there is a greater sending of packets through the external communication interface than through the management interface, this evidently collaborates with the need for the API Gateway, deploying on the Multi-Access Edge Computing in the form of VxF, needing to send information to the Fiware Orion Context, located in Cloud Computing.

Network Received Packets: Figure 52 and Figure 53 shows the cumulative amount of packets received through each of the interfaces on the container and on the host machine, respectively. In the virtual machine, there are two network interfaces, namely lo (environment management) and enp1s0 (external communication). In this case, there is a

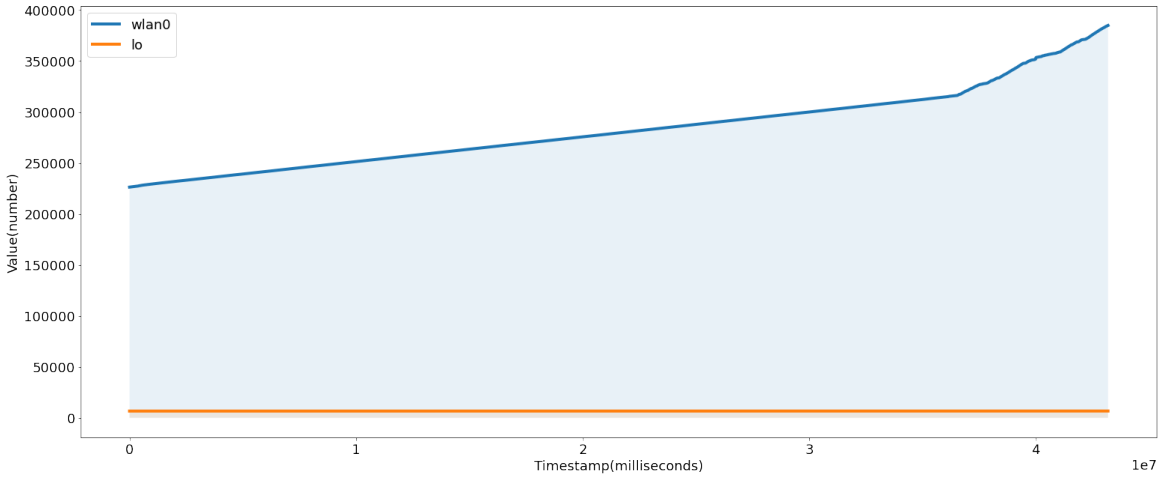

Figure 51 – Cumulative number of packets sent by each infrastructure interface.

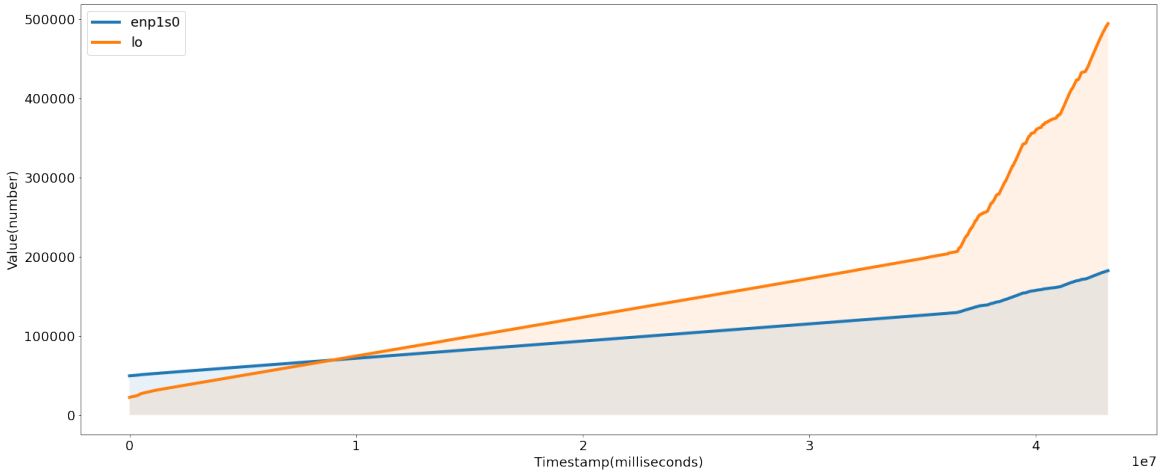

Figure 52 – Cumulative number of packets received by each virtual machine interface.

greater reception of packets through the management interface than through the external communication interface, this is due to responses between the processes that make up the API Gateway for correct functioning. In the case of infrastructure, there are two network interfaces, namely lo (infrastructure management) and wlan0 (external communication). In this case, there is a greater reception of packets through the external communication interface than through the management interface. This evidently collaborates with the need for API Gateway, deploying on Multi-Access Edge Computing in VxF form, needing to receive information from requests previously made to the Fiware Orion Context, located in Cloud Computing.

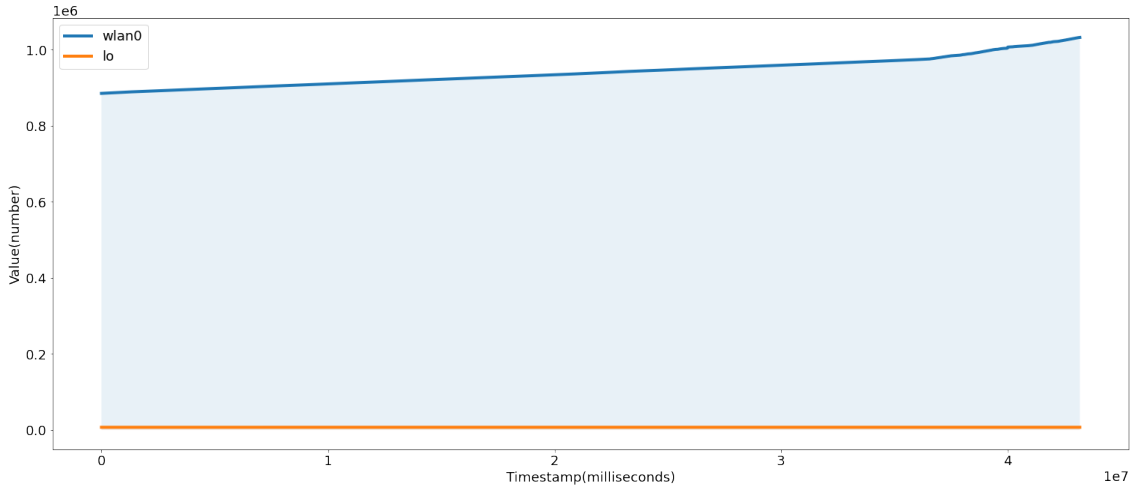

Figure 53 – Cumulative number of packets received by each infrastructure interface.

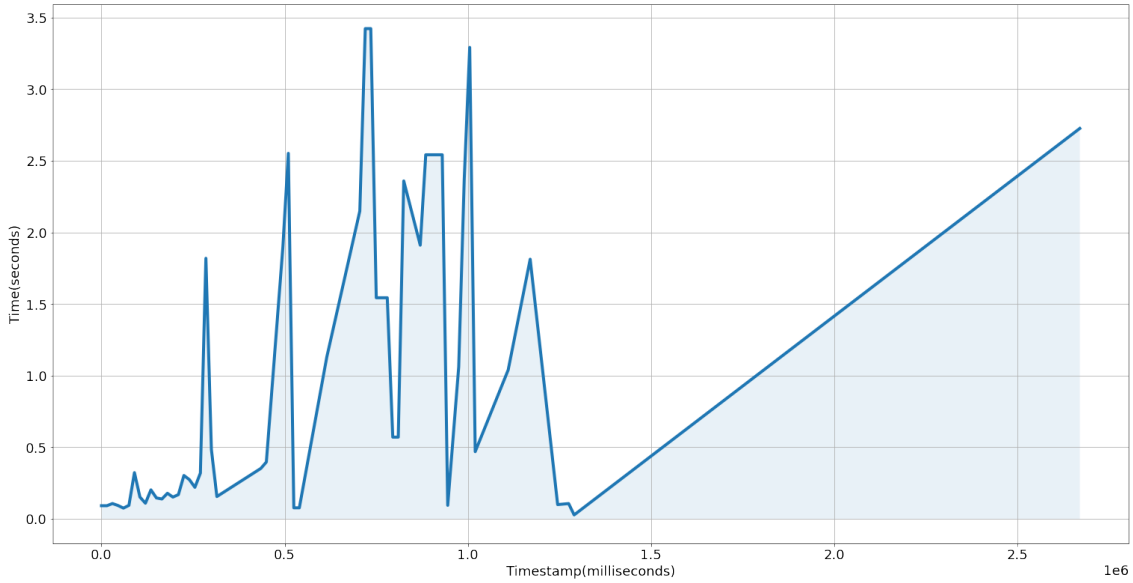

Figure 54 – The amount of time required to process a request.

4.3.2.3 Using CoAP protocol and LXD virtualization platform

Request and Response: Figure 54 and Figure 55 show the time required to process requests and responses, respectively. It can be noticed that both cases present processing peaks of equal magnitude, although for the processing time of a request it is considerably longer than the processing time of a response. Also, it is important to note that the time to process a request, given the underlying graph, is much longer than the time to process a response. This is due to the need for API Gateway, deployed over Multi-Access Egde Computing in VxF form, to store the mapping between requests in asynchronous responses.

CPU: Figure 56 and Figure 57 shows how many seconds of CPU are spent in kernel

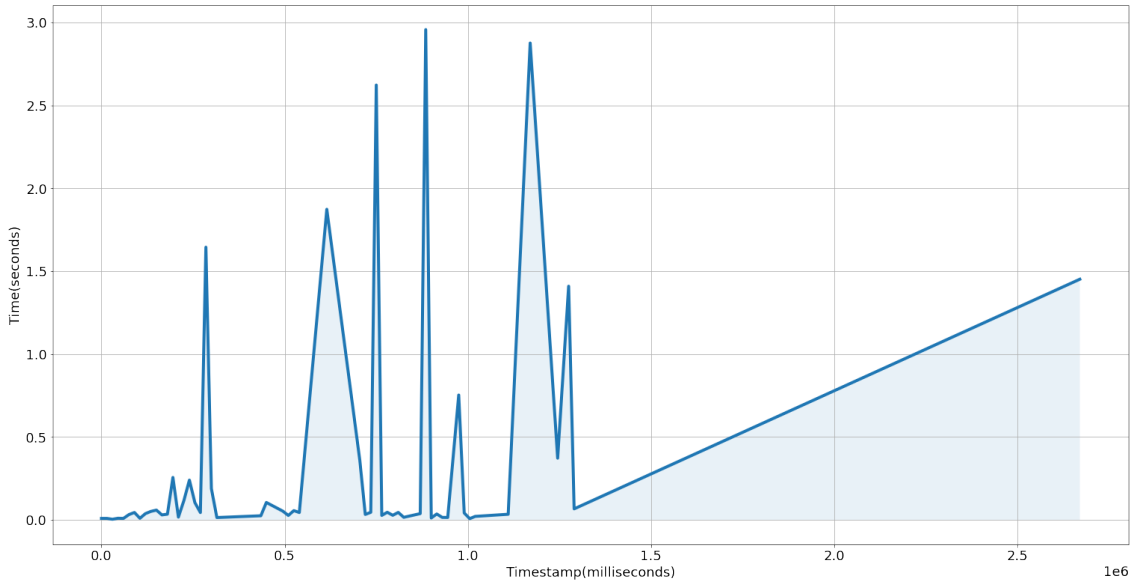

Figure 55 – The amount of time required to process a response.

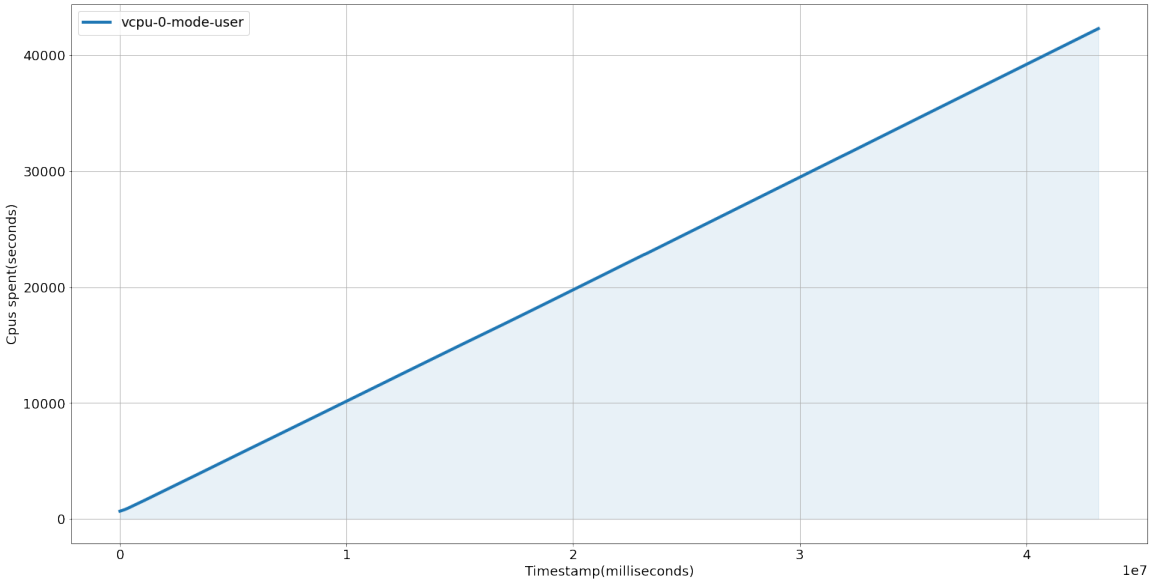

Figure 56 – Seconds the virtual cpus spent in each mode.

mode and user mode, respectively, by the host machine's container and operating system. It can be noted that the time spent in user mode of the container, which is unique in a virtualized environment, is similar to time spent in kernel mode of the host machine's operating system. This is probably due to the various calls of third-party libraries in the API Gateway operation, using the CoAP protocol. As such, API Gateway performance may degrade over time, and necessary measures can be taken to ensure quality of service.

Memory: Figure 58 and Figure 59 show the total time in seconds that a process cannot progress due to memory congestion, respectively by the container and the host

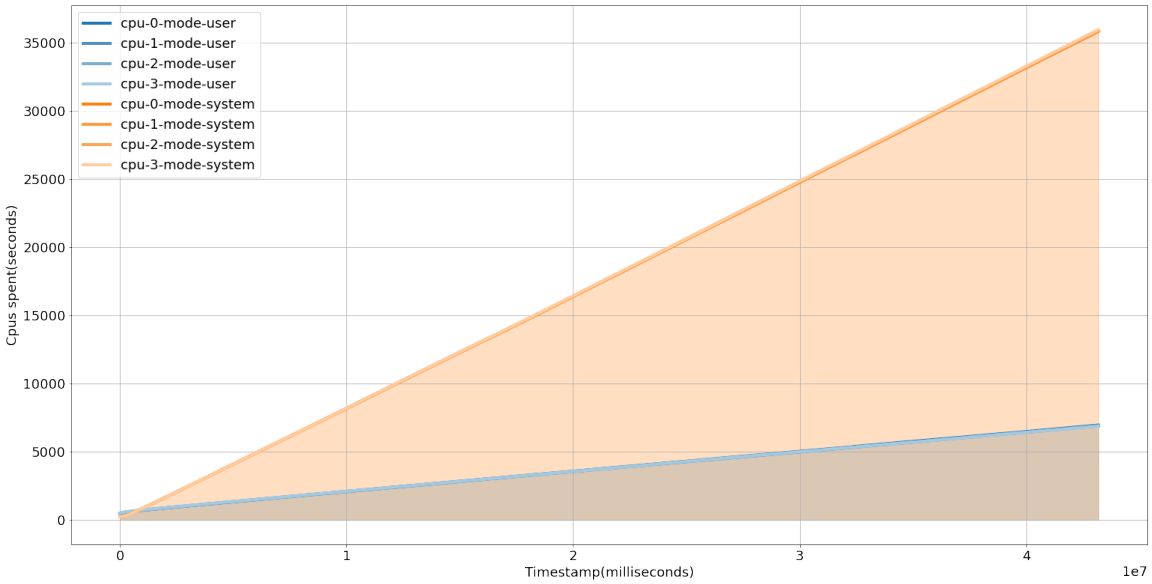

Figure 57 – Seconds the cpus spent in each mode.

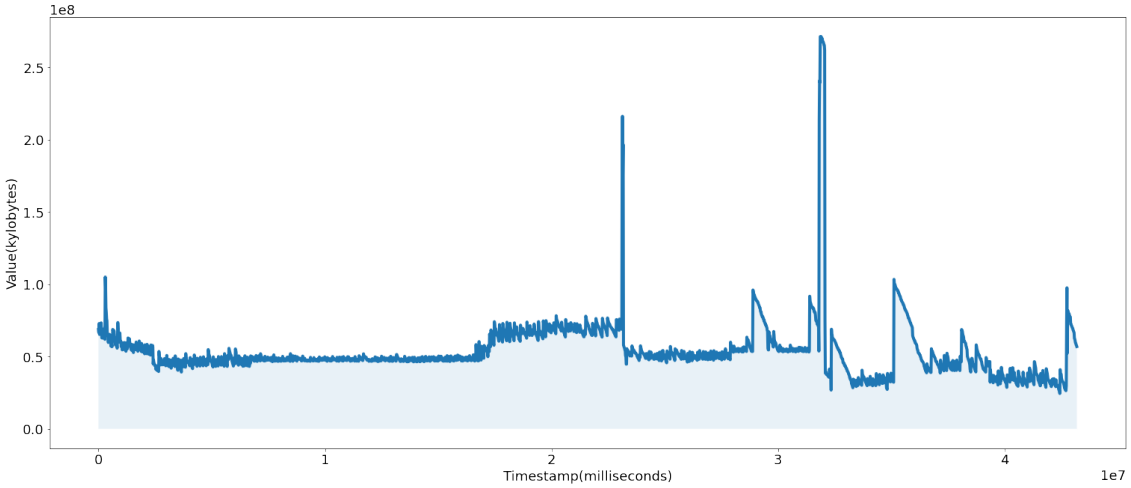

Figure 58 – Total process congestion time in the container.

machine's operating system. It can be noted that the bottleneck of container processes is similar to the bottleneck of processes on the host machine. This is due to the fact that containerization has part of its operation linked to the host machine's operating system, and often the container is considered just a host machine process. In this way, once the similarity is noticed, it is possible to verify problems in the infrastructure, linked to running VxFs.

Disk: Figure 60 and Figure 61 show the cumulative time taken by all disk read and write operations, respectively by the container and the host machine's operating system. It can be noted that time spent on read and write operations from the container is similar to time spent on write operations on the host machine. Again, this is due to the fact that

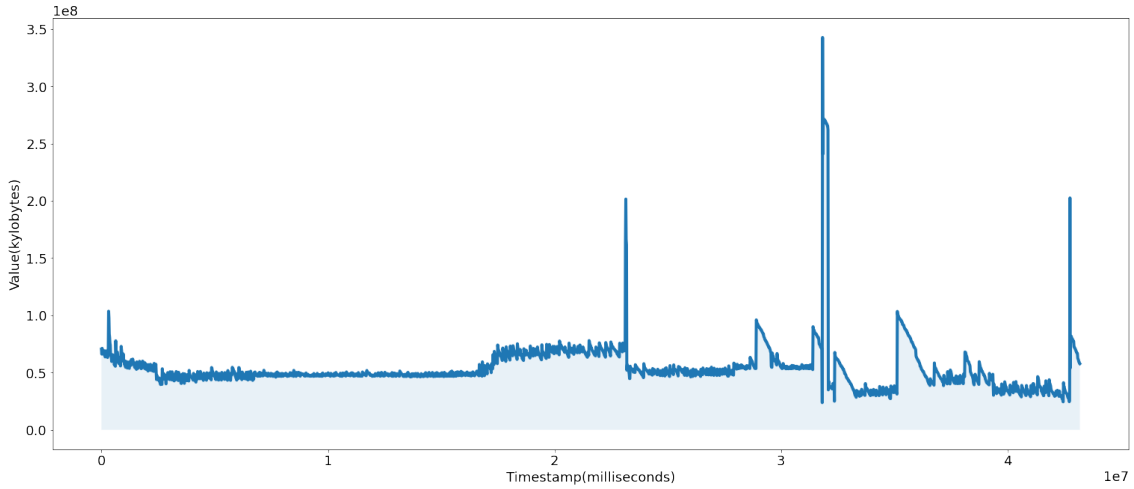

Figure 59 – Total process congestion time in the infrastructure.

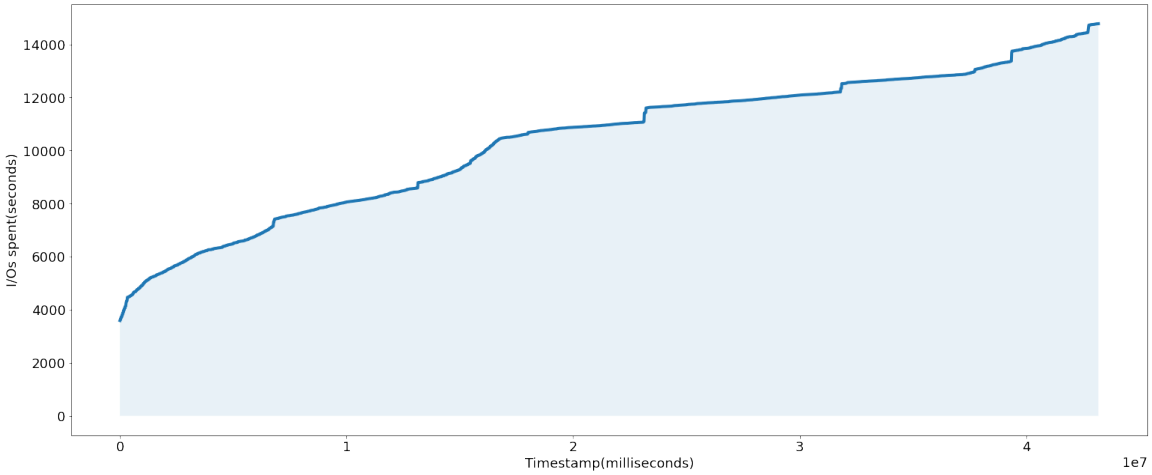

Figure 60 – Total time spent on disk I/O operations in the container.

containerization has part of its operation tied to the host machine's operating system, and often the container is considered just a host machine process. The continuous increase in the time spent on read and write operations caused at the end of the experiment can become a performance problem for the API Gateway, and consequently a problem in other VxFs. Therefore, once noticed, preventive actions can be taken so that such a bottleneck does not happen, thus guaranteeing the API Gateway's service quality as well as other VxFs.

Round-Trip Time: Figure 62 shows the RTT measured in seconds, from the moment a request is sent to the Fiware Orion Context to the moment a response is received from the Fiware Orion Context. It can be noted that there was a large variation in Round-Trip Time, with a large increase in time at the end of the experiment. This can be a problem in delivering a quality functional service. This way, other VxFs that could process the

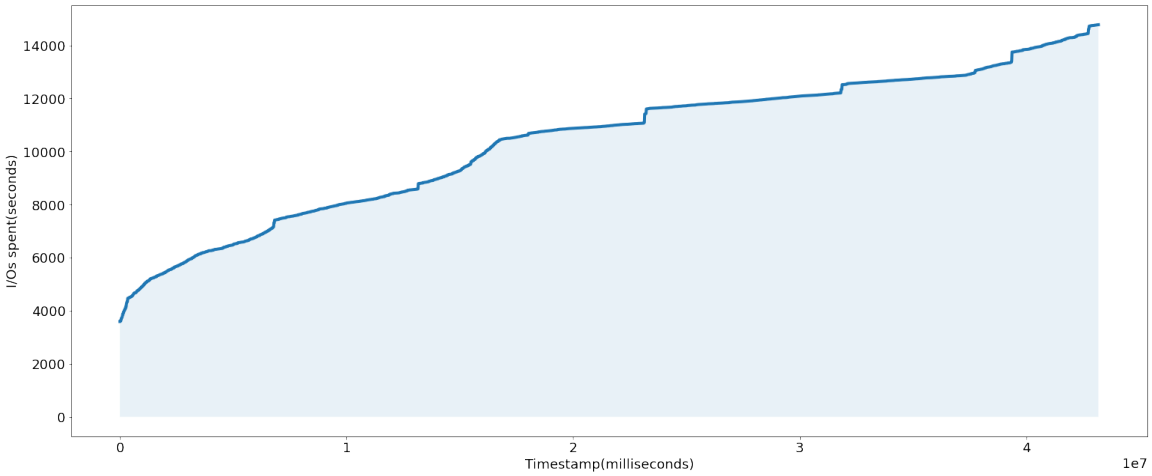

Figure 61 – Total time spent on disk I/O operations in the infrastructure.

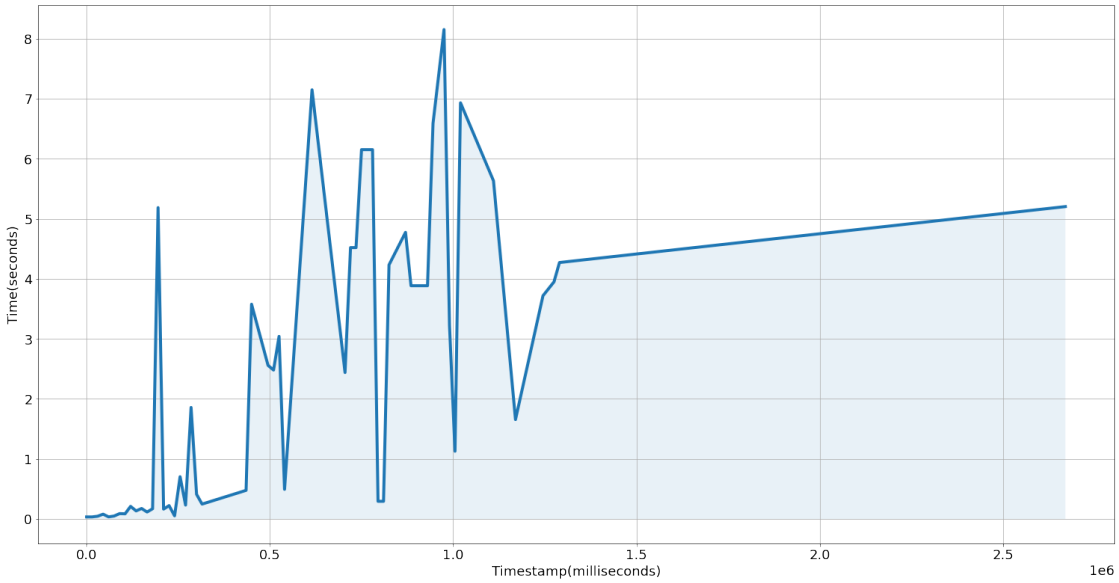

Figure 62 – The Round-Trip Time from API Gateway to Fiware Orion Context.

requests along the network would reduce the application's Round-Trip Time, solving the application's quality problems.

Network Transmit Packets: Figure 63 and Figure 64 shows the cumulative amount of packets sent through each of the interfaces on the container and on the host machine, respectively. In the container environment, there are two network interfaces, namely lo (environment management) and eth0 (external communication). In this case, more packets are sent through the management interface than through the external communication interface, this is due to the communication between processes that make up the API Gateway for correct functioning. In the case of infrastructure, there are two network interfaces, namely lo (infrastructure management) and wlan0 (external communication).

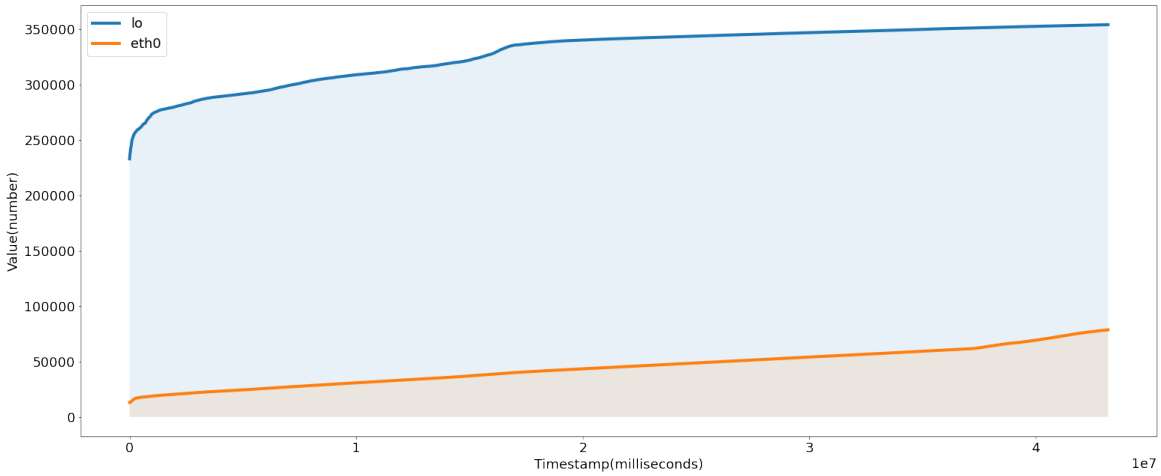

Figure 63 – Cumulative number of packets sent by each container interface.

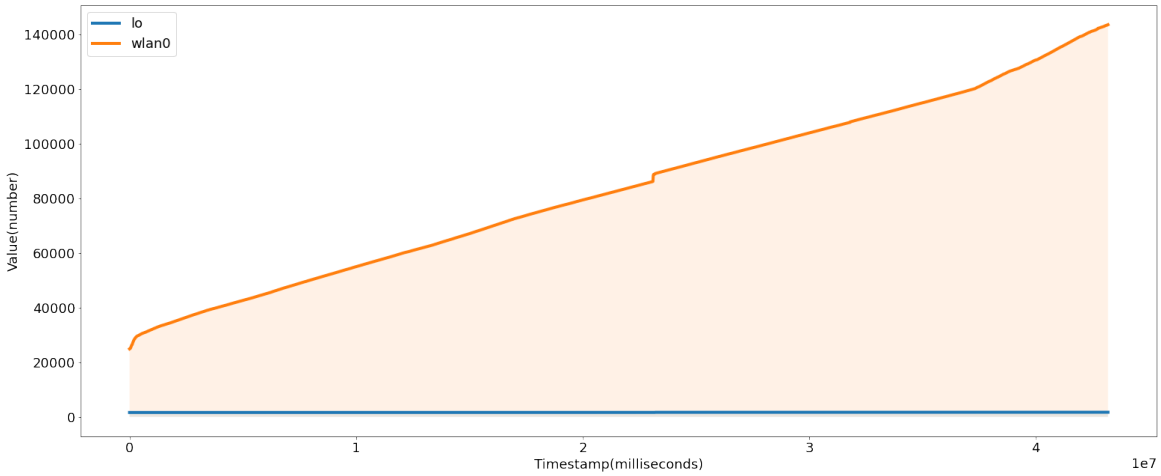

Figure 64 – Cumulative number of packets sent by each infrastructure interface.

In this case, there is a greater sending of packets through the external communication interface than through the management interface, this evidently collaborates with the need for the API Gateway, deploying on the Multi-Access Edge Computing in the form of VxF, needing to send information to the Fiware Orion Context, located in Cloud Computing.

Network Received Packets: Figure 65 and Figure 66 shows the cumulative amount of packets received through each of the interfaces on the container and on the host machine, respectively. In the container environment, there are two network interfaces, namely lo (environment management) and eth0 (external communication). In this case, there is a greater reception of packets through the management interface than through the external communication interface, this is due to responses between the processes that make up the API Gateway for correct functioning. In the case of infrastructure, there are two network interfaces, namely lo (infrastructure management) and wlan0 (external communication). In this case, there is a greater reception of packets through the external communication

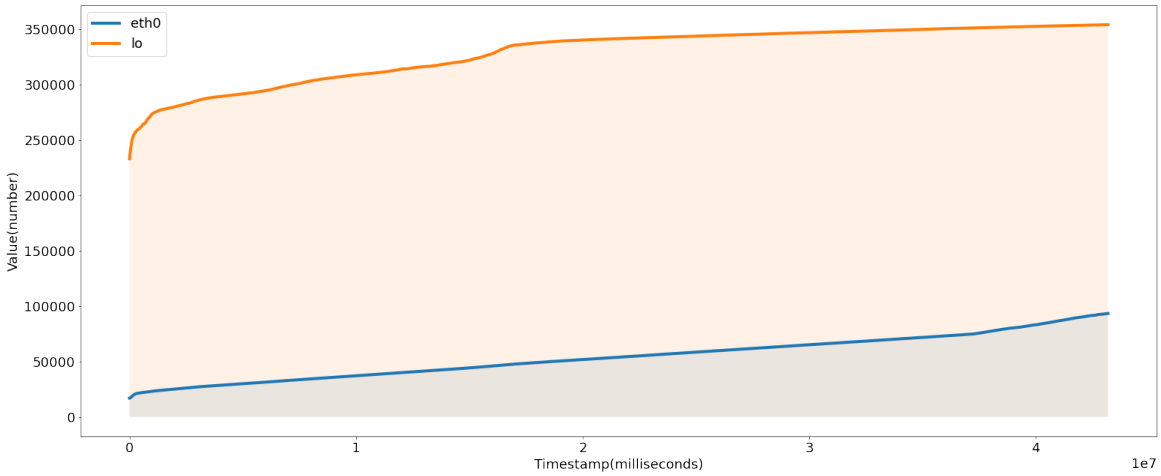

Figure 65 – Cumulative number of packets received by each container interface.

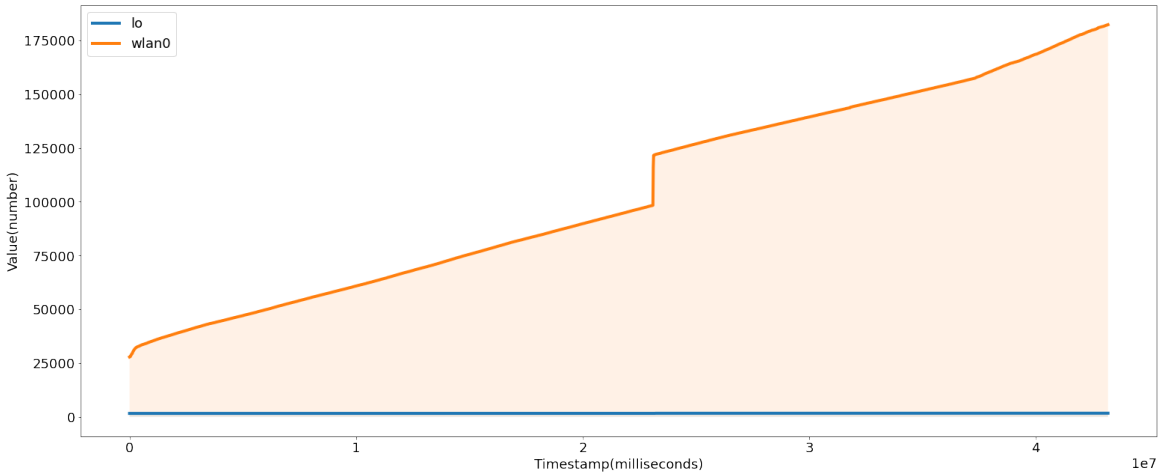

Figure 66 – Cumulative number of packets received by each infrastructure interface.

interface than through the management interface. This evidently collaborates with the need for API Gateway, deploying on Multi-Access Edge Computing in VxF form, needing to receive information from requests previously made to the Fiware Orion Context, located in Cloud Computing.

4.3.2.4 Using CoAP protocol and KVM virtualization platform

Request and Response: Figure 67 and Figure 68 show the time required to process requests and responses, respectively. It can be noted that despite the outliers and there was a wide variation in the processing time of requests and responses over time. Also, it is important to note that the time to process a request is much longer than the time to process a response. This is due to the need for API Gateway, deployed over Multi-Access Edge Computing in VxF form, to store the mapping between requests in asynchronous

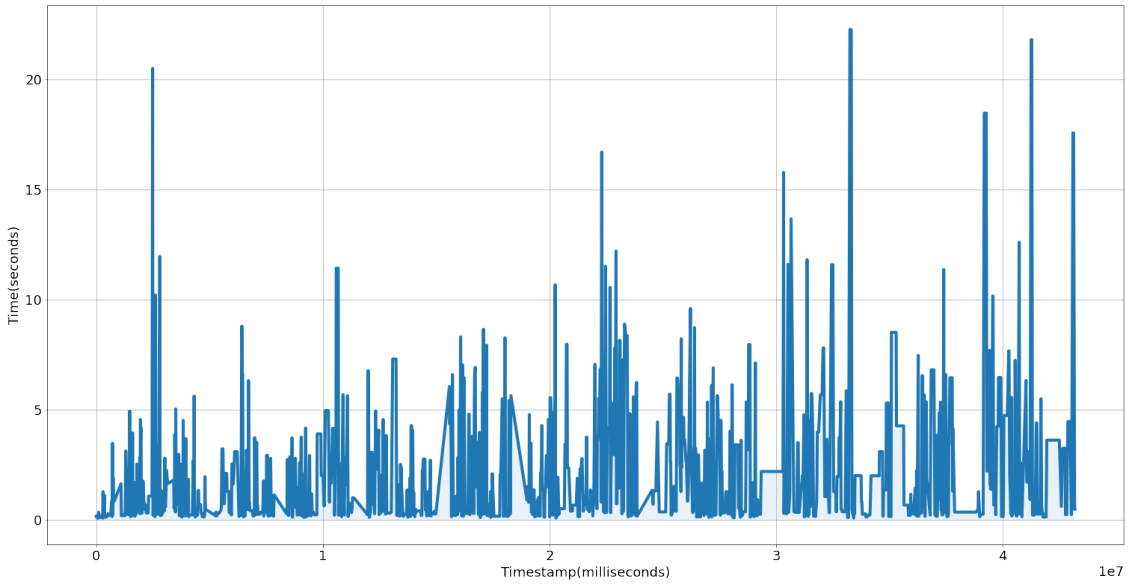

Figure 67 – The amount of time required to process a request.

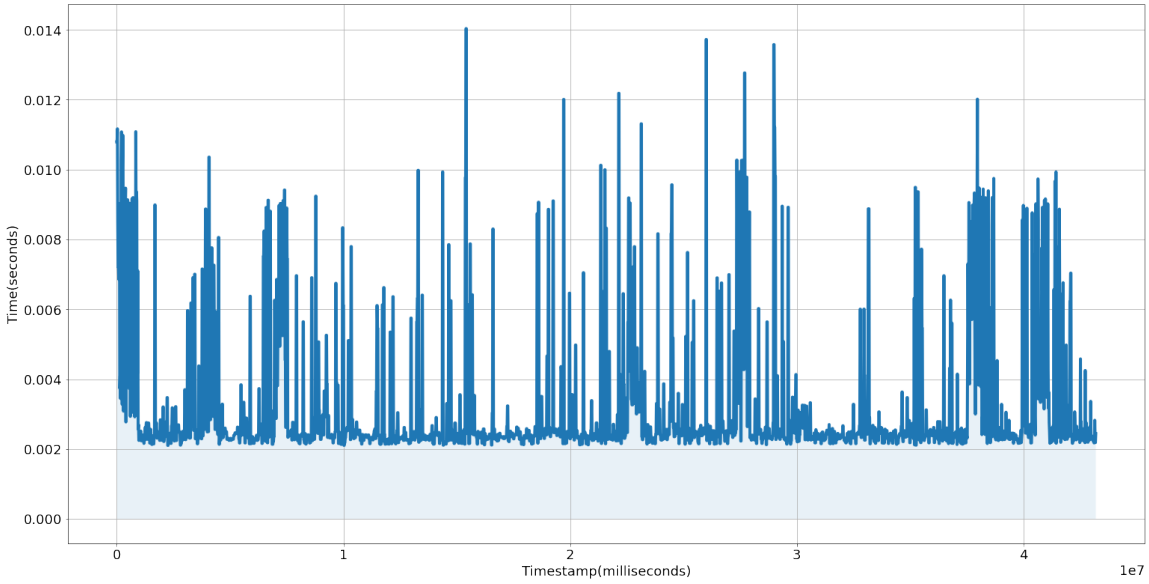

Figure 68 – The amount of time required to process a response.

responses.

CPU: Figure 69 and Figure 70 show how many seconds of CPU are spent in kernel mode and user mode, respectively, by the virtual machine and the host machine's operating system. It can be noted that the time spent in user mode of the virtual machine is greater than the time spent in kernel mode. Thus, the Gateway API deployed in a virtual machine, using the CoAP protocol, presents a loss of performance in relation to the overhead caused between context switches between user mode and kernel mode. On the other hand, it can be seen that the time spent in user mode on the host machine's

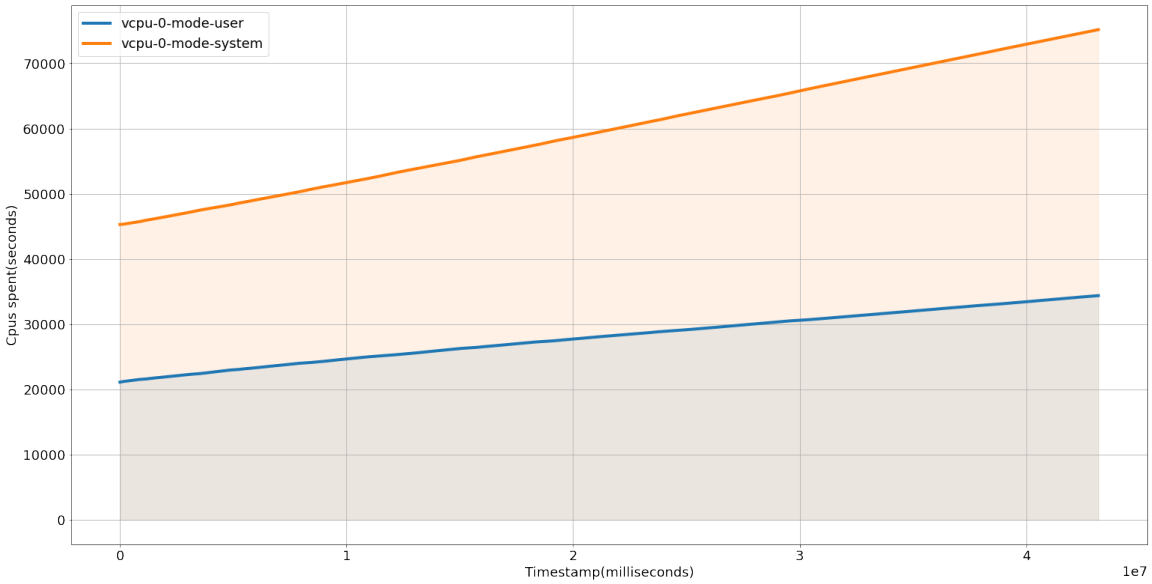

Figure 69 – Seconds the virtual cpus spent in each mode.

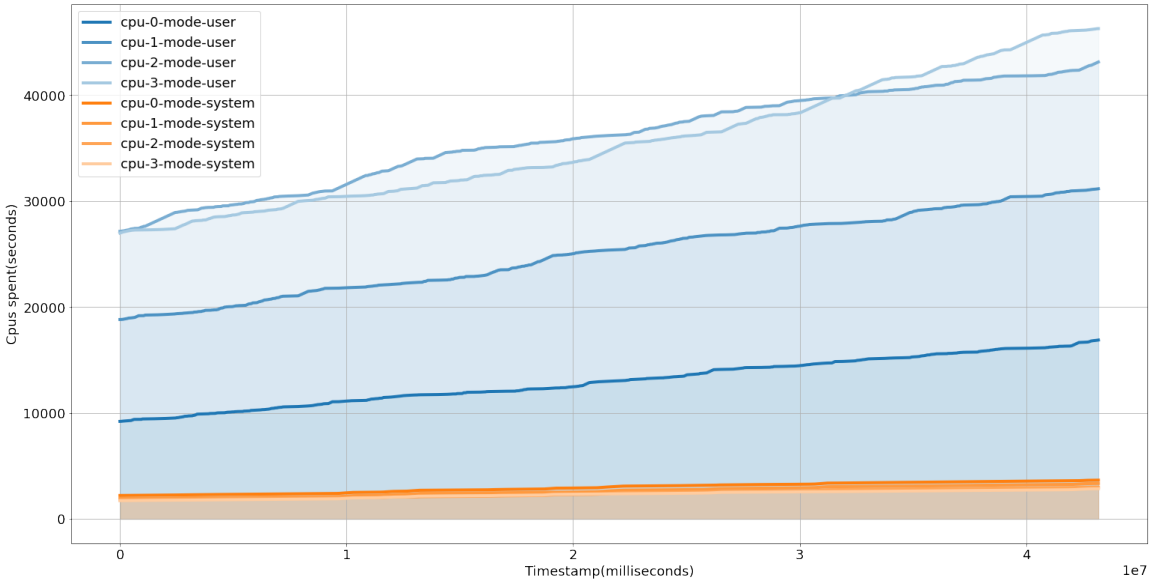

Figure 70 – Seconds the cpus spent in each mode.

operating system is much greater than the time spent in kernel mode. Furthermore, it is noteworthy that the way processing is distributed among the infrastructure processing cores is not evenly distributed. This of course supports the idea that the API Gateway's virtualized environment runs on top of the user-mode infrastructure, not kernel-mode. In this way, all context and message exchanges between the VxF operating system are fully controlled and decoupled from the host machine.

Memory: Figure 71 and Figure 72 shows the amount of RAM memory available, respectively by the virtual machine and the host machine's operating system. It can be

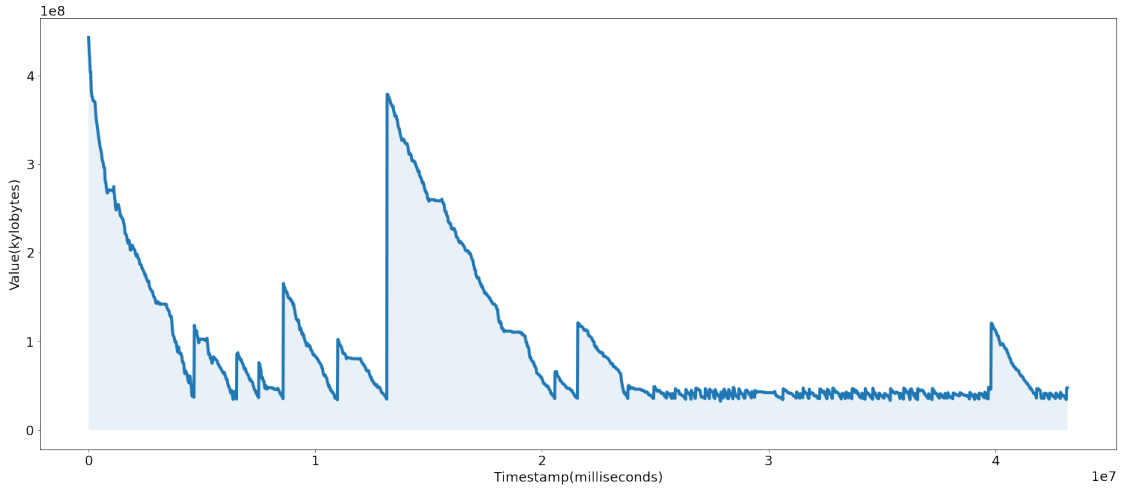

Figure 71 – The amount of RAM available on the container.

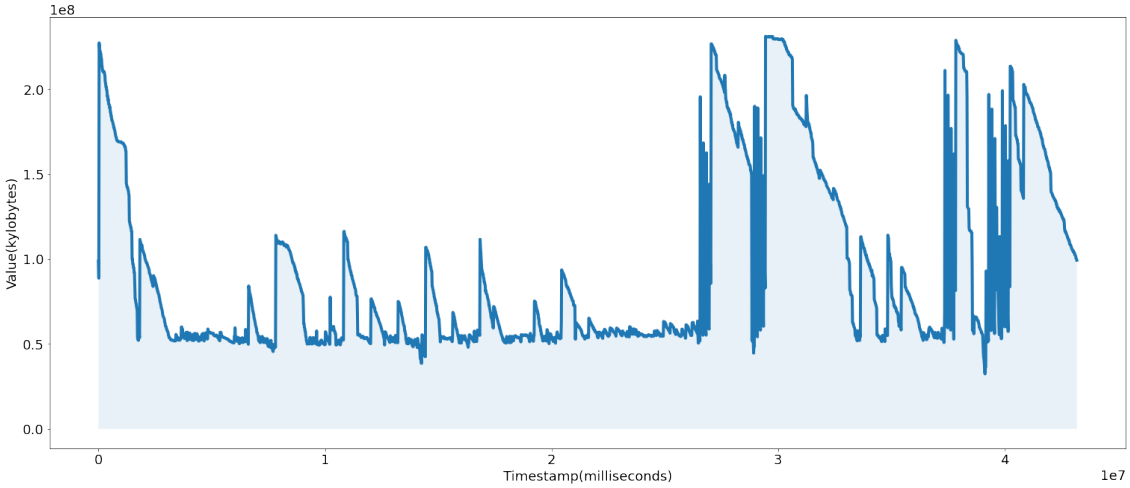

Figure 72 – The amount of RAM available on the infrastructure.

noticed that in both cases, as the experiment was executed, there was a decrease in the amount of available ram. Also, it is noteworthy that there are large variations in the amount of memory available in the infrastructure than in the virtual machine. This is likely due to processes present on the host machine, which compete with the virtualized environment. In this way, it is possible to verify the need for corrective actions for the correct functioning of the VxF, in the absence of memory for its functioning, or in the possibility of implementing new collaborative VxFs.

Disk: Figure 73 and Figure 74 shows the cumulative time spent on I/O operations by the virtual machine and the host machine's operating system respectively. It can be noted that in both cases, as the experiment was performed, there was an increase in the time spent on I/O operations, although in the case of infrastructure this was greater. This is due to the amount of processes existing in the host machine for its functioning. Thus, it

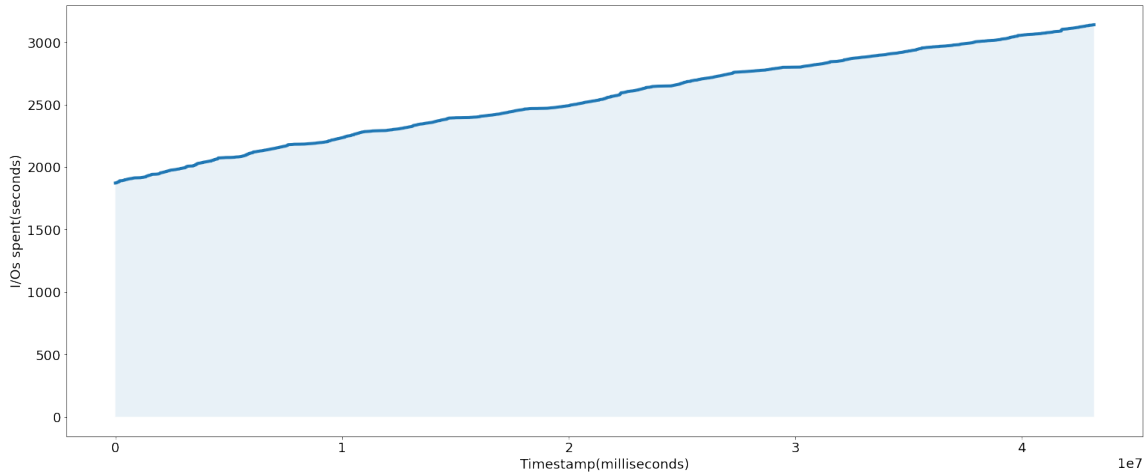

Figure 73 – Total time spent on disk I/O operations in the container.

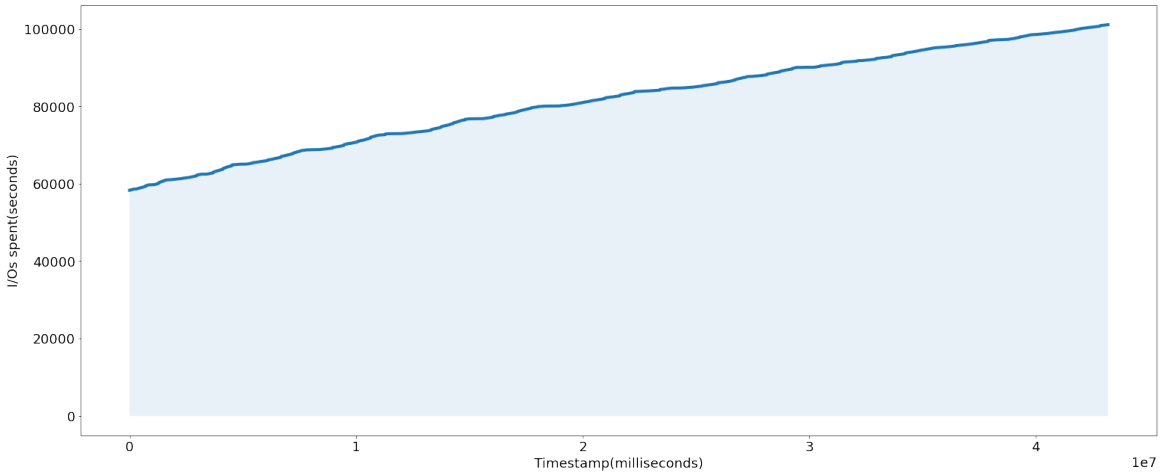

Figure 74 – Total time spent on disk I/O operations in the infrastructure.

is possible to verify the reasons for infrastructure performance degradation with respect to congestion caused by input and output operations and their impacts on other VxFs.

Round-Trip Time: Figure 75 shows the RTT measured in seconds, from the time a request is sent to the Fiware Orion Context to the time a response is received by the API Gateway. It can be noted that there was a large variation in Round-Trip Time over time. This is perhaps due to the large overhead caused by the additional layer between the virtualization environment, the virtualization layer, and the host machine. In addition, Full-Virtualization is highly decoupled from the underlying infrastructure, causing large context switches and thus increasing processing time.

Network Transmit Packets: Figure 76 and Figure 77 shows the cumulative amount of packets sent through each of the interfaces on the container and on the host machine, respectively. In the virtualized environment, there are two network interfaces, namely lo (environment management) and enp1s0 (external communication). In this case, more

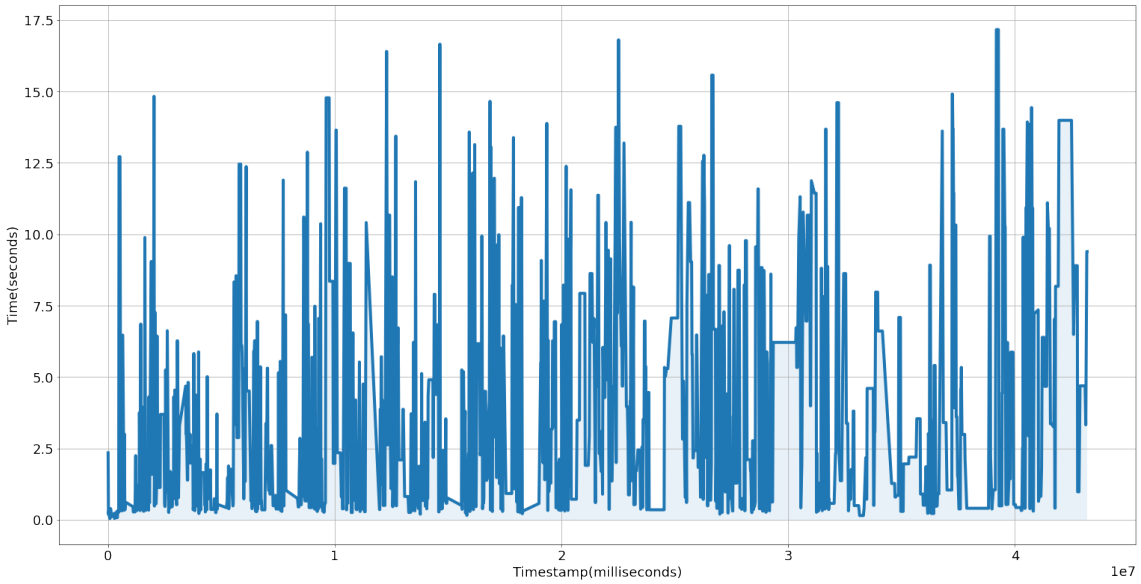

Figure 75 – The Round-Trip Time from API Gateway to Fiware Orion Context.

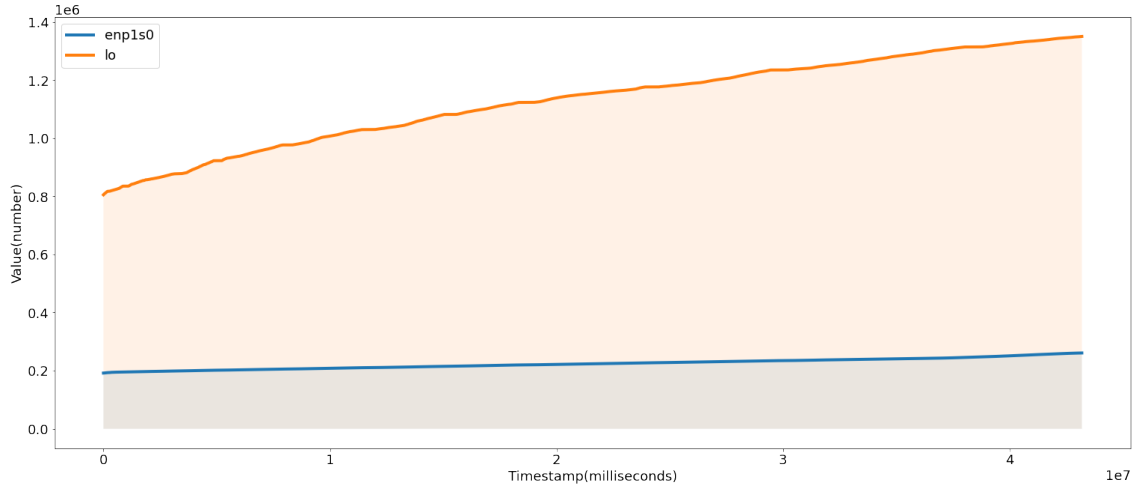

Figure 76 – Cumulative number of packets sent by each virtual machine interface.

packages are sent through the management interface than through the external communication interface, this is due to the communication between processes that make up the API Gateway for correct functioning. In the case of infrastructure, there are two network interfaces, namely lo (infrastructure management) and wlan0 (external communication). In this case, there is a greater sending of packets through the external communication interface than through the management interface, this evidently collaborates with the need for the API Gateway, deploying on the Multi-Access Edge Computing in the form of VxF, needing to send information to the Fiware Orion Context, located in Cloud Computing.

Network Received Packets: Figure 78 and Figure 79 shows the cumulative amount of packets received through each of the interfaces on the container and on the host machine,

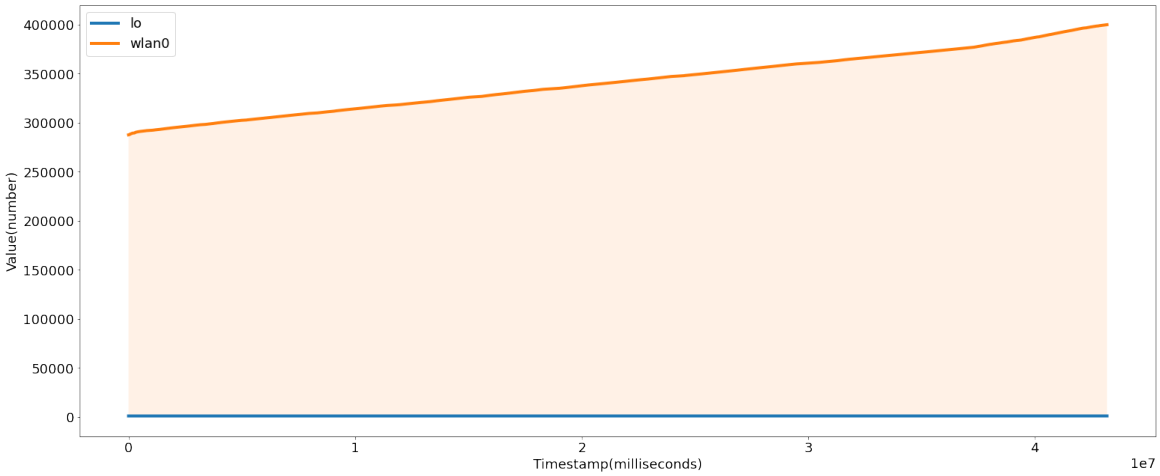

Figure 77 – Cumulative number of packets sent by each infrastructure interface.

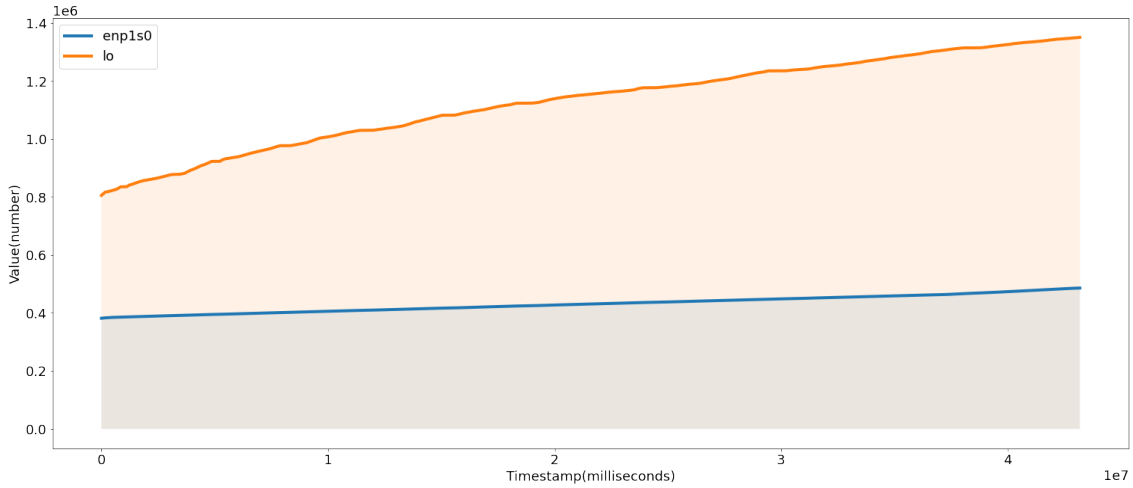

Figure 78 – Cumulative number of packets received by each container interface.

respectively. In the virtual machine, there are two network interfaces, namely lo (environment management) and enp1s0 (external communication). In this case, there is a greater reception of packets through the management interface than through the external communication interface, this is due to responses between the processes that make up the API Gateway for correct functioning. In the case of infrastructure, there are two network interfaces, namely lo (infrastructure management) and wlan0 (external communication). In this case, there is a greater reception of packets through the external communication interface than through the management interface. This evidently collaborates with the need for API Gateway, deploying on Multi-Access Edge Computing in VxF form, needing to receive information from requests previously made to the Fiware Orion Context, located in Cloud Computing.

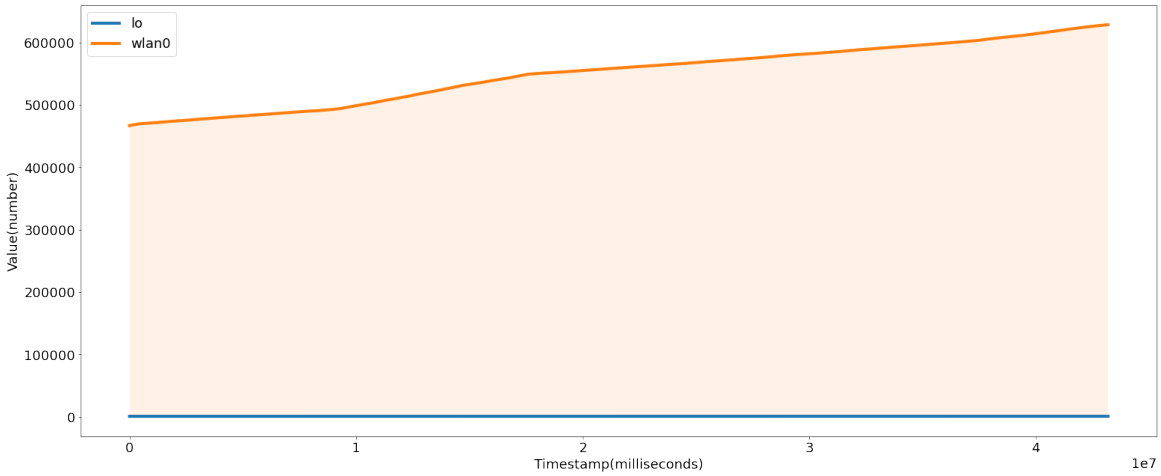

Figure 79 – Cumulative number of packets received by each infrastructure interface.

4.4 Overall Analysis

There are several lines of research in machine virtualization, and some are applied to the Edge Computing environment. Table 6 highlights a comparison between the different solutions found in the state of the art. At the end, a comparison of the proposed study and its differences with the studies presented above is presented. For each proposal, some aspects were analyzed. First, the virtualization platform, be it LXD, Docker, KVM, Xen and others. Second, the type of infrastructure used to make the virtual machines or containers run is on a Single Board Computer or Commodity Server. Third, the way used to collect metrics across the infrastructure has been verified. Fourth and last, the metrics collected from the infrastructure to the service itself.

As seen from the Table 6, GARNET advances the state of the art by proposing and evaluating an architecture compatible with the ETSI Management and Orchestration software stack for managing Virtualized Everything Functions. This architecture is independent of the underlying hardware, Commodity Server (x86_64) or Single Board Computers (AArch64), as well as independent of virtualization platform i.e. Full-Virtualization (KVM) or Containerization (LXD). Thus, allowing continuous use of VxFs using existing market tools. Furthermore, the architecture can provide a complete infrastructure capable of collecting a variety of metrics under the OpenMetrics format. Such metrics in turn can be related to the support infrastructure to the deployed services.

Table 6 – State of the art summary.

	Virtualization	**Infrastructure**	**Monitoring**	**Metrics**
Ramalho e Neto (2016)	Docker and KVM	Single Board Computer (AArch64)	Benchmarks: NBench, SysBench, Bonnie++, LINPACK, STREAM	Schedule thread, floating-point operations, I/O operantions, memory bandwidth.
Ahmad, Alowibdi e Ilyas (2017)	LXD	Single Board Computer (AArch64)	OpenStack Cloud Computing	Container batch launch times for operating systems Ubuntu 14.04 Trusty Tahr, Ubuntu 16.04 Xenial Xerus and CirrOS 0.3.4.
Cziva e Pezaros (2017)	XEN, ClickOS, KVM, Container	Commodity Server (x86_64)	Virtual Infrastructure Manager (VIM) and OpenDaylight.	Delay of idle round-trip time (RTT). Time to manage (create, start, and stop) virtual machines and containers. Idle memory consumption.
Li et al. (2017)	Docker and VMWare	Commodity Server (x86_64)	Benchmarks: Iperf, HardInfo, STREAM, Bonnie++	Data throughput in communication. Latency in computation. Data throughput in memory. Transaction speed, Data throughput in disk.
Riggio et al. (2018)	Docker and VirtualBox	Commodity Server (x86_64)	Command Linux: top and stat	Round Trip Time in send a web page request. Amount time required to boot application. CPU and memory utilization.
Richards, Moreira e Silva (2020)	KVM	Single Board Computer (AArch64)	Psutil tool	Percentage of CPU. Memory response. Buffer swap memory. Temperature of CPU. Boot time machines.
Rigazzi et al. (2019)	KVM and LXD	Commodity Server (x86_64)	GPU-Z	GPU load. GPU power consumption. Memory usage. Bandwith consumed.
Richards, Moreira e Silva (2020)	KVM	Single Boad Computer (AArch64)	Psutil tool	Percentage of CPU. Memory response. Buffer swap memory. Temperature of CPU. Boot time machines.
Tambe, Mandge e Antony Franklin (2020)	Docker and KVM	Commodity Server (x86_64)	Application Function (AF), which acts as a registered service.	Instantiation and temination time. Service response time. Service registry time. Traffic rules time. DNS rules time.
Mena et al. (2020)	LXD	Commodity Server (x86_64)	Zabbix: monitoring the virtualized environment.	High CPU. Memory utilization. Network bandwith. TCP Connections. Disk space.
This work	LXD and KVM	Commodity Server (x86_64) and Single Board Computer (AArch64)	Any exporter capable of implementing the OpenMetrics standard. Anywhere in the deployment stack.	Any metric provided by the exporter, regardless of location in the deployment stack.

Chapter 5

Conclusion

IoT devices generate a large amount of data at high speeds and varied formats, and require additional processing and storage capacity. In this sense, cloud computing offers "unlimited" capacity for IoT devices based on the existing model. However, this integration is complex for a number of reasons, such as performance, communication delay, quality of service, and so on. Edge computing adds a layer between the cloud and the IoT environment to solve these issues, as devices at the edge are located locally closer to the IoT devices.

Edge Computing, especially Multi-Access Edge Computing, drives the world of the Internet of Things, extending the cloud computing model to the edge of the network, where intermediate network nodes such as routers, switches, firewall and hubs participate in information processing and decision making to improve network security, agility, latency and efficiency.

With this new paradigm, new challenges and opportunities arise in the development and standardization of systems in Multi-Access Edge Computing, regarding the integration of systems in different providers in high-end terminals and IoT. Furthermore, the operational challenge comes from the heterogeneous device management cycle in terms of performance, deployment, monitoring and compliance.

This work proposed and evaluated GARNET (*edGe virtuAlized all function management architEcTure*), an architecture compatible with the ETSI Management and Orchestration software stack for managing Virtualized Everything Functions in edge. This architecture is capable of deploying services on bare-metal servers to low-cost devices on an ongoing basis. This work also focused on the complete monitoring of the infrastructure from the application to the service, a fundamental element for the management of virtualized systems. Furthermore, a unified system to monitor computing resources in Multi-Access Edge Computing is proposed and implemented.

Therefore, this work provides an overview and introductory overview of the state of the art in the ability of Multi-Access Edge Computing to provide computing, storage and network services enabled by Network Function Virtualization. Together, Multi-Access

Edge Computing and Network Functions Virtualization provide an effective solution for modern computing, ensuring speed and reliability for end users.

5.1 Main Contributions

The main contributions of this work are: (1) an architecture that allows the continuous use of VxFs in devices present at the edge of the network; (2) an infrastructure for continuous monitoring of computing resources, in physical and virtualized environments, as well as the monitoring of specific applications that are at the edge of the network; (3) an experimental study of the feasibility of VxFs, using different virtualization platforms, on Multi-Access Edge Computing devices such as a Raspberry Pi 4; (4) an experimental study of the native unified monitoring and management of VxFs deployed in different infrastructures present at the edge of the network; (5) an experimental study of the complete monitoring capability of an environment capable of supporting VxFs, correlated metrics at the physical infrastructure level, virtualized environment and at the application level;

5.2 Scientific Publications

5.2.1 Accepted Papers

CUNHA, H. G. V. O. da; MOREIRA, R.; SILVA, F. de O. A comparative study between containerization and full-virtualization of virtualized everything functions in edge computing. In: BAROLLI, L.; WOUNGANG, I.; ENOKIDO, T. (Ed.).**Advanced Information Networking and Applications**. Cham: Springer International Publishing, 2021. p. 771–782. ISBN 978-3-030-75075-6.

5.2.2 Submitted Papers

CUNHA, H. G. V. O. da; MOREIRA, R.; SILVA, F. de O. An architecture for managing and experimentally researching virtualized functions for edge computing. In: XII Future Internet Experimental Research Workshop (WPEIF 2021)(SBRC 2021 - WPEIF). Uberlândia, Brazil: 2021.

5.3 Future Work

The Internet of Things has advanced recently from an experimental technology to what will become the backbone of future customer value for products and services business sector. The emergence of Multi Access Edge Computing technology aims to extend

cloud computing resources to the edge of radio network access, providing real-time, high-bandwidth, and low-latency access to radio network resources. IoT is identified as a key use case for MEC, given MEC's ability to provide cloud platform and gateway services at the edge of the network. MEC will inspire the development of a multitude of applications and services that require ultra-low latency and high quality of service due to its dense geographic distribution and broad support for mobility. Here are some applications for future work:

- ❑ **Health**: Humanoid robots sitting next to an elderly person may need 1ms latency tactile feedback for their care services. Mission-critical use cases, such as remote surgery, require ultra-low latency, uninterrupted communication links, and collaborations between surgeons present in different locations. Remote patient monitoring is another use case that allows consultants in major cities to interact with patients residing far from the medical center. Frequent updates of health records for an elderly person or someone with a chronic illness need to proceed ubiquitously and securely. with size potential use cases and scenarios, the role of the MEC in health and the welfare industries become more evident.

- ❑ **Autonomous Vehicles**: Use cases in these categories include autonomous and semi-autonomous driving, vehicle maintenance, and vehicle information. To operate an efficient and reliable vehicle network, a number of features must be improved, including real-time traffic monitoring, continuous in-vehicle sensing, application support and improved safety. In this regard, the upcoming MEC systems are expected to offer a higher level of flexibility, leveraging emerging technologies related to network software.

- ❑ **Gaming, AR and VR**: Mixed Reality (MR) combines Virtual Reality (VR) and Augmented Reality (AR) technologies enabling humans to interact more naturally with the virtual worlds based on data aggregated by IoT devices. With IoT, AR technologies are able to benefit directly from the high end interconnection of objects that characterize the IoT environment through which users can extend their interactions from the real world to the virtual world]. While operating VR devices over wireless links and deploying the VR control center at MEC server, the tracking accuracy can be increased with round trip latency of 1 ms and high reliability. Migrating computationally intensive tasks to edge servers will increase the computational capacity of VR devices and save their battery-life. Furthermore, MEC will allow VR devices to access cloud resources in an on-demand fashion.

Bibliography

ABDERRAHIM, M.; OUZZIF, M.; GUILLOUARD, K.; FRANCOIS, J.; LEBRE, A. A holistic monitoring service for fog/edge infrastructures: A foresight study. In: **2017 IEEE 5th International Conference on Future Internet of Things and Cloud (FiCloud)**. [S.l.: s.n.], 2017. p. 337–344.

Ahmad, M.; Alowibdi, J. S.; Ilyas, M. U. viot: A first step towards a shared, multi-tenant iot infrastructure architecture. In: **2017 IEEE International Conference on Communications Workshops (ICC Workshops)**. [S.l.: s.n.], 2017. p. 308–313.

AI, Y.; PENG, M.; ZHANG, K. Edge computing technologies for internet of things: a primer. **Digital Communications and Networks**, v. 4, n. 2, p. 77–86, 2018. ISSN 2352-8648. Disponível em: <https://www.sciencedirect.com/science/article/pii/S2352864817301335>.

ALAM, I.; SHARIF, K.; LI, F.; LATIF, Z.; KARIM, M.; BISWAS, S.; NOUR, B.; WANG, Y. A survey of network virtualization techniques for internet of things using sdn and nfv. **ACM Computing Surveys (CSUR)**, ACM New York, NY, USA, v. 53, n. 2, p. 1–40, 2020.

ALVES, Y. **EnergyNow | IoT aplicada ao fomento da Indústria 4.0**. IEEE Dataport, 2020. Disponível em: <https://dx.doi.org/10.21227/r83n-kw91>.

AMRI, A. E.; MEDDEB, A. Optimal traffic routing in the network virtualization context. **International Journal of Communication Systems**, Wiley Online Library, p. e4846, 2021.

ARIZA-PORRAS, C.; KUZNETSOV, V.; LEGGER, F. The CMS monitoring infrastructure and applications. **Computing and Software for Big Science**, Springer Science and Business Media LLC, v. 5, n. 1, jan. 2021. Disponível em: <https://doi.org/10.1007/s41781-020-00051-x>.

ASVIJA, B.; ESWARI, R.; BIJOY, M. Security in hardware assisted virtualization for cloud computing—state of the art issues and challenges. **Computer Networks**, v. 151, p. 68–92, 2019. ISSN 1389-1286. Disponível em: <https://www.sciencedirect.com/science/article/pii/S1389128618302998>.

AULIYA, Y. A.; NURDINSYAH, Y.; WULANDARI, D. A. R. Performance comparison of docker and LXD with ApacheBench. **Journal of Physics:**

Conference Series, IOP Publishing, v. 1211, p. 012042, apr 2019. Disponível em: <https://doi.org/10.1088/1742-6596/1211/1/012042>.

BABU, S. A.; HAREESH, M. J.; MARTIN, J. P.; CHERIAN, S.; SASTRI, Y. System performance evaluation of para virtualization, container virtualization, and full virtualization using xen, openvz, and xenserver. In: **2014 Fourth International Conference on Advances in Computing and Communications**. [S.l.: s.n.], 2014. p. 247–250.

BARAKABITZE, A. A.; AHMAD, A.; MIJUMBI, R.; HINES, A. 5g network slicing using sdn and nfv: A survey of taxonomy, architectures and future challenges. **Computer Networks**, v. 167, p. 106984, 2020. ISSN 1389-1286. Disponível em: <https://www.sciencedirect.com/science/article/pii/S1389128619304773>.

BASFORD, P. J.; JOHNSTON, S. J.; PERKINS, C. S.; GARNOCK-JONES, T.; TSO, F. P.; PEZAROS, D.; MULLINS, R. D.; YONEKI, E.; SINGER, J.; COX, S. J. Performance analysis of single board computer clusters. **Future Generation Computer Systems**, v. 102, p. 278–291, 2020. ISSN 0167-739X. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0167739X1833142X>.

BEHRAVESH, R.; CORONADO, E.; RIGGIO, R. Performance evaluation on virtualization technologies for nfv deployment in 5g networks. In: **2019 IEEE Conference on Network Softwarization (NetSoft)**. [S.l.: s.n.], 2019. p. 24–29.

BENEDICTIS, M. D.; LIOY, A. On the establishment of trust in the cloud-based etsi nfv framework. In: IEEE. **2017 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN)**. [S.l.], 2017. p. 280–285.

BESERRA, D.; PINHEIRO, M. K.; SOUVEYET, C.; STEFFENEL, L. A.; MORENO, E. D. Performance evaluation of os-level virtualization solutions for hpc purposes on soc-based systems. In: **2017 IEEE 31st International Conference on Advanced Information Networking and Applications (AINA)**. [S.l.: s.n.], 2017. p. 363–370.

BEYER, B.; JONES, C.; PETOFF, J.; MURPHY, N. R. **Site reliability engineering: How Google runs production systems**. [S.l.]: " O'Reilly Media, Inc.", 2016.

BHARDWAJ, A.; KRISHNA, C. R. Virtualization in cloud computing: Moving from hypervisor to containerization—a survey. **Arabian Journal for Science and Engineering**, Springer, p. 1–17, 2021.

BONOMI, F.; MILITO, R.; NATARAJAN, P.; ZHU, J. Fog computing: A platform for internet of things and analytics. In: **Big Data and Internet of Things**. [S.l.: s.n.], 2014.

BONOMI, F.; MILITO, R.; ZHU, J.; ADDEPALLI, S. Fog computing and its role in the internet of things. In: **Proceedings of the First Edition of the MCC Workshop on Mobile Cloud Computing**. New York, NY, USA: Association for Computing Machinery, 2012. (MCC '12), p. 13–16. ISBN 9781450315197. Disponível em: <https://doi.org/10.1145/2342509.2342513>.

BOTTA, A.; de Donato, W.; PERSICO, V.; PESCAPé, A. Integration of cloud computing and internet of things: A survey. **Future Generation Computer Systems**, v. 56, p. 684–700, 2016. ISSN 0167-739X. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0167739X15003015>.

BRAZIL, B. **Prometheus: Up & Running: Infrastructure and Application Performance Monitoring**. [S.l.]: " O'Reilly Media, Inc.", 2018.

BRUSCHI, R.; DAVOLI, F.; LAGO, P.; LOMBARDO, A.; LOMBARDO, C.; RAMETTA, C.; SCHEMBRA, G. An sdn/nfv platform for personal cloud services. **IEEE Transactions on Network and Service Management**, v. 14, n. 4, p. 1143–1156, 2017.

BUYYA, R.; SRIRAMA, S. N. Internet of things (iot) and new computing paradigms. In: _____. **Fog and Edge Computing: Principles and Paradigms**. [S.l.: s.n.], 2019. p. 1–23.

CALLEGATI, F.; CERRONI, W.; CONTOLI, C.; FORESTA, F. Performance of intent-based virtualized network infrastructure management. In: IEEE. **2017 IEEE International Conference on Communications (ICC)**. [S.l.], 2017. p. 1–6.

CHAKRABORTY, M.; KUNDAN, A. P. Prometheus. In: _____. **Monitoring Cloud-Native Applications: Lead Agile Operations Confidently Using Open Source Software**. Berkeley, CA: Apress, 2021. p. 99–131. ISBN 978-1-4842-6888-9. Disponível em: <https://doi.org/10.1007/978-1-4842-6888-9_4>.

CHANG, C.; SRIRAMA, S. N.; BUYYA, R. Mobile cloud business process management system for the internet of things: A survey. **ACM Comput. Surv.**, Association for Computing Machinery, New York, NY, USA, v. 49, n. 4, dez. 2016. ISSN 0360-0300. Disponível em: <https://doi.org/10.1145/3012000>.

_____. Mobile cloud business process management system for the internet of things: A survey. **ACM Comput. Surv.**, Association for Computing Machinery, New York, NY, USA, v. 49, n. 4, dez. 2016. ISSN 0360-0300. Disponível em: <https://doi.org/10.1145/3012000>.

CHAYAPATHI, R. **Network functions virtualization (NFV) with a touch of SDN**. Boston: Addison-Wesley, 2017. ISBN 9780134463056.

CHEN, B.; WAN, J.; CELESTI, A.; LI, D.; ABBAS, H.; ZHANG, Q. Edge computing in iot-based manufacturing. **IEEE Communications Magazine**, v. 56, n. 9, p. 103–109, 2018.

CHEN, Y.; KUNZ, T. Performance evaluation of iot protocols under a constrained wireless access network. In: **2016 International Conference on Selected Topics in Mobile Wireless Networking (MoWNeT)**. [S.l.: s.n.], 2016. p. 1–7.

CHIANG, M.; HA, S.; RISSO, F.; ZHANG, T.; CHIH-LIN, I. Clarifying fog computing and networking: 10 questions and answers. **IEEE Communications Magazine**, v. 55, n. 4, p. 18–20, 2017.

CHIANG, M.; ZHANG, T. Fog and iot: An overview of research opportunities. **IEEE Internet of Things Journal**, v. 3, n. 6, p. 854–864, 2016.

CHIRAMMAL, H. D.; MUKHEDKAR, P.; VETTATHU, A. **Mastering KVM virtualization: dive in to the cutting edge techniques of Linux KVM virtualization, and build the virtualization solutions your datacentre demands**. Birmingham Mumbai: Packt Publishing, 2016. (Community experience distilled). OCLC: 965647188. ISBN 9781784399054.

COLITTI, W.; STEENHAUT, K.; CARO, N. D. Integrating wireless sensor networks with the web. **Extending the Internet to Low power and Lossy Networks (IP+ SN 2011)**, 2011.

COMPASTIé, M.; BADONNEL, R.; FESTOR, O.; HE, R. From virtualization security issues to cloud protection opportunities: An in-depth analysis of system virtualization models. **Computers & Security**, v. 97, p. 101905, 2020. ISSN 0167-4048. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0167404820301814>.

CORSARO, A.; BALDONI, G. fogØ5: Unifying the computing, networking and storage fabrics end-to-end. In: **2018 3rd Cloudification of the Internet of Things (CIoT)**. [S.l.: s.n.], 2018. p. 1–8.

CUNHA, H. G. V. O. da; MOREIRA, R.; SILVA, F. de O. A comparative study between containerization and full-virtualization of virtualized everything functions in edge computing. In: BAROLLI, L.; WOUNGANG, I.; ENOKIDO, T. (Ed.). **Advanced Information Networking and Applications**. Cham: Springer International Publishing, 2021. p. 771–782. ISBN 978-3-030-75075-6.

Cziva, R.; Pezaros, D. P. Container network functions: Bringing nfv to the network edge. **IEEE Communications Magazine**, v. 55, n. 6, p. 24–31, 2017.

DAHMEN-LHUISSIER, S. **ETSI - Our group Network Functions Virtualisation (NFV)**. 2021. Disponível em: <https://www.etsi.org/committee/1427-nfv>.

DAKIC, V.; CHIRAMMAL, H. D.; MUKHEDKAR, P.; VETTATHU, A. **Mastering KVM Virtualization: Design expert data center virtualization solutions with the power of Linux KVM**. [S.l.]: Packt Publishing Ltd, 2020.

DAVIES, S.; BROADHURST, P. et al. **WebSphere MQ V6 Fundamentals**. [S.l.]: IBM Redbooks, 2005.

DEMIRCI, S.; SAGIROGLU, S.; DEMIRCI, M. Energy-efficient virtual security function placement in nfv-enabled networks. **Sustainable Computing: Informatics and Systems**, v. 30, p. 100494, 2021. ISSN 2210-5379. Disponível em: <https://www.sciencedirect.com/science/article/pii/S2210537920302171>.

DJORDJEVIC, B.; TIMCENKO, V.; KRALJEVIC, N.; MACEK, N. File system performance comparison in full hardware virtualization with ESXi, KVM, hyper-v and xen hypervisors. **Advances in Electrical and Computer Engineering**, Universitatea Stefan cel Mare din Suceava, v. 21, n. 1, p. 11–20, 2021. Disponível em: <https://doi.org/10.4316/aece.2021.01002>.

DOLUI, K.; DATTA, S. K. Comparison of edge computing implementations: Fog computing, cloudlet and mobile edge computing. In: **2017 Global Internet of Things Summit (GIoTS)**. [S.l.: s.n.], 2017. p. 1–6.

DOYLE, F.; COSGROVE, J. Steps towards digitization of manufacturing in an sme environment. **Procedia Manufacturing**, v. 38, p. 540–547, 2019. ISSN 2351-9789. 29th International Conference on Flexible Automation and Intelligent Manufacturing (FAIM 2019), June 24-28, 2019, Limerick, Ireland, Beyond Industry 4.0: Industrial Advances, Engineering Education and Intelligent Manufacturing. Disponível em: <https://www.sciencedirect.com/science/article/pii/S235197892030069X>.

DURUMERIC, Z.; MA, Z.; SPRINGALL, D.; BARNES, R.; SULLIVAN, N.; BURSZTEIN, E.; BAILEY, M.; HALDERMAN, J. A.; PAXSON, V. The security impact of HTTPS interception. In: **Proceedings 2017 Network and Distributed System Security Symposium**. Internet Society, 2017. Disponível em: <https://doi.org/10.14722/ndss.2017.23456>.

EVANS, K. **CNCF to host OpenMetrics in the Sandbox**. 2018. Disponível em: <https://www.cncf.io/blog/2018/08/10/cncf-to-host-openmetrics-in-the-sandbox/>.

FARAHZADI, A.; SHAMS, P.; REZAZADEH, J.; FARAHBAKHSH, R. Middleware technologies for cloud of things: a survey. **Digital Communications and Networks**, v. 4, n. 3, p. 176–188, 2018. ISSN 2352-8648. Disponível em: <https://www.sciencedirect.com/science/article/pii/S2352864817301268>.

FEDERICO, A. D. rev.ng. In: **Proceedings of the Fifth Workshop on Cryptography and Security in Computing Systems**. ACM, 2018. Disponível em: <https://doi.org/10.1145/3178291.3178297>.

FERRARI, P.; SISINNI, E.; BRANDãO, D.; ROCHA, M. Evaluation of communication latency in industrial iot applications. In: **2017 IEEE International Workshop on Measurement and Networking (M N)**. [S.l.: s.n.], 2017. p. 1–6.

GARCÍA-ROIS, J.; FONDO-FERREIRO, P.; GIL-CASTIÑEIRA, F.; GONZÁLEZ-CASTAÑO, F. J.; CANDAL-VENTUREIRA, D. Evaluating management and orchestration impact on closed-loop orchestration delay. **Software: Practice and Experience**, Wiley Online Library, v. 51, n. 2, p. 193–212, 2021.

GAWALI, S. K.; DESHMUKH, M. K. Energy autonomy in iot technologies. **Energy Procedia**, v. 156, p. 222–226, 2019. ISSN 1876-6102. 5th International Conference on Power and Energy Systems Engineering (CPESE 2018). Disponível em: <https://www.sciencedirect.com/science/article/pii/S1876610218310920>.

GEDIA, D.; PERIGO, L. Performance evaluation of sdn-vnf in virtual machine and container. In: **2018 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN)**. [S.l.: s.n.], 2018. p. 1–7.

GIUST, F.; COSTA-PEREZ, X.; REZNIK, A. Multi-access edge computing: An overview of etsi mec isg. **IEEE 5G Tech Focus**, v. 1, n. 4, p. 4, 2017.

GIUST, F.; VERIN, G.; ANTEVSKI, K.; CHOU, J.; FANG, Y.; FEATHERSTONE, W.; FONTES, F.; FRYDMAN, D.; LI, A.; MANZALINI, A. et al. Mec deployments in 4g and evolution towards 5g. **ETSI White paper**, v. 24, n. 2018, p. 1–24, 2018.

GRUENER, S.; KOZIOLEK, H.; RüCKERT, J. Towards resilient iot messaging: An experience report analyzing mqtt brokers. In: **2021 IEEE 18th International Conference on Software Architecture (ICSA)**. [S.l.: s.n.], 2021. p. 69–79.

GUBBI, J.; BUYYA, R.; MARUSIC, S.; PALANISWAMI, M. Internet of things (iot): A vision, architectural elements, and future directions. **Future Generation Computer Systems**, v. 29, n. 7, p. 1645–1660, 2013. ISSN 0167-739X. Including Special sections: Cyber-enabled Distributed Computing for Ubiquitous Cloud and Network Services & Cloud Computing and Scientific Applications — Big Data, Scalable Analytics, and Beyond. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0167739X13000241>.

GUPTA, B.; QUAMARA, M. An overview of internet of things (iot): Architectural aspects, challenges, and protocols. **Concurrency and Computation: Practice and Experience**, v. 32, n. 21, p. e4946, 2020. E4946 CPE-18-0159.R1. Disponível em: <https://onlinelibrary.wiley.com/doi/abs/10.1002/cpe.4946>.

HALFACREE, G.; UPTON, E. **Raspberry Pi user guide**. [S.l.]: John Wiley & Sons, 2012.

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

HUANG, T.; YANG, W.; WU, J.; MA, J.; ZHANG, X.; ZHANG, D. A survey on green 6g network: Architecture and technologies. **IEEE Access**, v. 7, p. 175758–175768, 2019.

IGLESIAS-URKIA, M.; ORIVE, A.; URBIETA, A.; CASADO-MANSILLA, D. Analysis of CoAP implementations for industrial internet of things: a survey. **Journal of Ambient Intelligence and Humanized Computing**, Springer Science and Business Media LLC, v. 10, n. 7, p. 2505–2518, mar. 2018. Disponível em: <https://doi.org/10.1007/s12652-018-0729-z>.

JIA, M.; YIN, Z.; LI, D.; GUO, Q.; GU, X. Toward improved offloading efficiency of data transmission in the iot-cloud by leveraging secure truncating ofdm. **IEEE Internet of Things Journal**, v. 6, n. 3, p. 4252–4261, 2019.

KADIYALA, K. P.; COBB, J. A. Inter-as traffic engineering with sdn. In: **2017 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN)**. [S.l.: s.n.], 2017. p. 1–7.

KARAKUS, M.; DURRESI, A. A survey: Control plane scalability issues and approaches in software-defined networking (sdn). **Computer Networks**, Elsevier, v. 112, p. 279–293, 2017.

KHOSHKHOLGHI, M. A.; DERAHMAN, M. N.; ABDULLAH, A.; SUBRAMANIAM, S.; OTHMAN, M. Energy-efficient algorithms for dynamic virtual machine consolidation in cloud data centers. **IEEE Access**, v. 5, p. 10709–10722, 2017.

KIYANOVSKI, A.; TSAFRIR, D. **The Real Difference Between Emulation and Paravirtualization of High-Throughput I/O Devices**. Tese (Doutorado) — Computer Science Department, Technion, 2017.

KORABLEVA, O.; KALIMULLINA, O.; KURBANOVA, E. Building the monitoring systems for complex distributed systems: Problems and solutions. In: **Proceedings of the 19th International Conference on Enterprise Information Systems**.

SCITEPRESS - Science and Technology Publications, 2017. Disponível em: <https://doi.org/10.5220/0006271002210228>.

KOVACS, A. Comparison of different linux containers. In: **2017 40th International Conference on Telecommunications and Signal Processing (TSP)**. [S.l.: s.n.], 2017. p. 47–51.

KROTOV, V. The internet of things and new business opportunities. **Business Horizons**, v. 60, n. 6, p. 831–841, 2017. ISSN 0007-6813. THE GENERATIVE POTENTIAL OF EMERGING TECHNOLOGY. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0007681317301076>.

KUKLINSKI, S.; TOMASZEWSKI, L. Dasmo: A scalable approach to network slices management and orchestration. In: **NOMS 2018 - 2018 IEEE/IFIP Network Operations and Management Symposium**. [S.l.: s.n.], 2018. p. 1–6.

KURTANOVIC, Z.; MAALEJ, W. Automatically classifying functional and non-functional requirements using supervised machine learning. In: **2017 IEEE 25th International Requirements Engineering Conference (RE)**. [S.l.: s.n.], 2017. p. 490–495.

LAI, Y.-C.; ALI, A.; HOSSAIN, M. S.; LIN, Y.-D. Performance modeling and analysis of tcp and udp flows over software defined networks. **Journal of Network and Computer Applications**, v. 130, p. 76–88, 2019. ISSN 1084-8045. Disponível em: <https://www.sciencedirect.com/science/article/pii/S1084804519300189>.

LAN, T.; HAN, Q.; FAN, H.; LAN, J. Fpga-based packets processing acceleration platform for vnf. In: IEEE. **2017 8th IEEE International Conference on Software Engineering and Service Science (ICSESS)**. [S.l.], 2017. p. 314–317.

LARMO, A.; RATILAINEN, A.; SAARINEN, J. Impact of coap and mqtt on nb-iot system performance. **Sensors**, v. 19, n. 1, 2019. ISSN 1424-8220. Disponível em: <https://www.mdpi.com/1424-8220/19/1/7>.

LEA, P. **Internet of Things for architects: architecting IoT solutions by implementing sensors, communication infrastructure, edge computing, analytics, and security**. Birmingham, UK: Packt Publishing, 2018. ISBN 9781788470599.

LI, M.; PANG, J.; YUE, F.; LIU, F.; WANG, J.; TAN, J. Enhancing dynamic binary translation in mobile computing by leveraging polyhedral optimization. **Wireless Communications and Mobile Computing**, Hindawi Limited, v. 2021, p. 1–12, abr. 2021. Disponível em: <https://doi.org/10.1155/2021/6611867>.

LI, X.; GARCIA-SAAVEDRA, A.; COSTA-PEREZ, X.; BERNARDOS, C. J.; GUIMARÃES, C.; ANTEVSKI, K.; MANGUES-BAFALLUY, J.; BARANDA, J.; ZEYDAN, E.; CORUJO, D. et al. 5growth: An end-to-end service platform for automated deployment and management of vertical services over 5g networks. **IEEE Communications Magazine**, IEEE, v. 59, n. 3, p. 84–90, 2021.

Li, Z.; Kihl, M.; Lu, Q.; Andersson, J. A. Performance overhead comparison between hypervisor and container based virtualization. In: **2017 IEEE 31st International**

Conference on Advanced Information Networking and Applications (AINA). [S.l.: s.n.], 2017. p. 955–962.

LI, Z.; KIHL, M.; LU, Q.; ANDERSSON, J. A. Performance overhead comparison between hypervisor and container based virtualization. In: IEEE. **2017 IEEE 31st International Conference on Advanced Information Networking and Applications (AINA)**. [S.l.], 2017. p. 955–962.

LIU, G.; RAMAKRISHNAN, K. K.; SCHLANSKER, M.; TOURRILHES, J.; WOOD, T. Design challenges for high performance, scalable nfv interconnects. In: **Proceedings of the Workshop on Kernel-Bypass Networks**. New York, NY, USA: Association for Computing Machinery, 2017. (KBNets '17), p. 49–54. ISBN 9781450350532. Disponível em: <https://doi.org/10.1145/3098583.3098592>.

LOKE, S. W.; NAPIER, K.; ALALI, A.; FERNANDO, N.; RAHAYU, W. Mobile computations with surrounding devices: Proximity sensing and multilayered work stealing. **ACM Trans. Embed. Comput. Syst.**, Association for Computing Machinery, New York, NY, USA, v. 14, n. 2, fev. 2015. ISSN 1539-9087. Disponível em: <https://doi.org/10.1145/2656214>.

LUIZELLI, M. C.; RAZ, D.; SA'AR, Y. Optimizing nfv chain deployment through minimizing the cost of virtual switching. In: **IEEE INFOCOM 2018 - IEEE Conference on Computer Communications**. [S.l.: s.n.], 2018. p. 2150–2158.

LV, Z.; XIU, W. Interaction of edge-cloud computing based on sdn and nfv for next generation iot. **IEEE Internet of Things Journal**, v. 7, n. 7, p. 5706–5712, 2020.

MANJULA, T.; SREENIVASULU, U.; HUSSAIN, S. J. A dynamic raspberry pi sense hat multimodality alerting system by using aws iot. **Indian Journal of Science and Technology**, v. 9, p. 39, 2016.

MANOHAR, H. L.; ASIR, T. R. G. Data consumption pattern of mqtt protocol for iot applications. In: VENKATARAMANI, G. P.; SANKARANARAYANAN, K.; MUKHERJEE, S.; ARPUTHARAJ, K.; NARAYANAN, S. S. (Ed.). **Smart Secure Systems – IoT and Analytics Perspective**. Singapore: Springer Singapore, 2018. p. 12–22. ISBN 978-981-10-7635-0.

MATHEW, D.; JOSE, B. A. Performance analysis of virtualized embedded computing systems. In: **2017 7th International Symposium on Embedded Computing and System Design (ISED)**. [S.l.: s.n.], 2017. p. 1–5.

MAVRIDIS, I.; KARATZA, H. Performance and overhead study of containers running on top of virtual machines. In: IEEE. **2017 IEEE 19th Conference on Business Informatics (CBI)**. [S.l.], 2017. v. 2, p. 32–38.

MECHTRI, M.; GHRIBI, C.; SOUALAH, O.; ZEGHLACHE, D. Nfv orchestration framework addressing sfc challenges. **IEEE Communications Magazine**, v. 55, n. 6, p. 16–23, 2017.

Mena, M. P.; Papageorgiou, A.; Ochoa-Aday, L.; Siddiqui, S.; Baldoni, G. Enhancing the performance of 5g slicing operations via multi-tier orchestration. In: **2020 23rd Conference on Innovation in Clouds, Internet and Networks and Workshops (ICIN)**. [S.l.: s.n.], 2020. p. 131–138.

MIJUMBI, R.; SERRAT, J.; GORRICHO, J.-L.; LATRÉ, S.; CHARALAMBIDES, M.; LOPEZ, D. Management and orchestration challenges in network functions virtualization. **IEEE Communications Magazine**, IEEE, v. 54, n. 1, p. 98–105, 2016.

MIRKHANZADEH, B.; SHAKERI, A.; SHAO, C.; RAZO, M.; TACCA, M.; GALIMBERTI, G. M.; MARTINELLI, G.; CARDANI, M.; FUMAGALLI, A. An sdn-enabled multi-layer protection and restoration mechanism. **Optical Switching and Networking**, v. 30, p. 23–32, 2018. ISSN 1573-4277. Disponível em: <https://www.sciencedirect.com/science/article/pii/S157342771730228X>.

MONIR, M. F.; PAN, D. Application and assessment of click modular firewall vs pox firewall in sdn/nfv framework. In: **2020 IEEE REGION 10 CONFERENCE (TENCON)**. [S.l.: s.n.], 2020. p. 991–996.

MONTERO, R.; AGRAZ, F.; PAGÈS, A.; SPADARO, S. End-to-end network slicing in support of latency-sensitive 5g services. In: TZANAKAKI, A.; VARVARIGOS, M.; MUÑOZ, R.; NEJABATI, R.; YOSHIKANE, N.; ANASTASOPOULOS, M.; MARQUEZ-BARJA, J. (Ed.). **Optical Network Design and Modeling**. Cham: Springer International Publishing, 2020. p. 51–61. ISBN 978-3-030-38085-4.

MORLEY, J.; WIDDICKS, K.; HAZAS, M. Digitalisation, energy and data demand: The impact of internet traffic on overall and peak electricity consumption. **Energy Research & Social Science**, v. 38, p. 128–137, 2018. ISSN 2214-6296. Disponível em: <https://www.sciencedirect.com/science/article/pii/S2214629618301051>.

NAIK, N. Choice of effective messaging protocols for iot systems: Mqtt, coap, amqp and http. In: **2017 IEEE International Systems Engineering Symposium (ISSE)**. [S.l.: s.n.], 2017. p. 1–7.

NAIK, S.; MARAL, V. Cyber security — iot. In: **2017 2nd IEEE International Conference on Recent Trends in Electronics, Information Communication Technology (RTEICT)**. [S.l.: s.n.], 2017. p. 764–767.

NALEPA, B.; GWIAZDA, A.; BANAS, W. Study of communication between driver sbRIO9632 and raspberry pi 3. **IOP Conference Series: Materials Science and Engineering**, IOP Publishing, v. 400, p. 052006, sep 2018. Disponível em: <https://doi.org/10.1088/1757-899x/400/5/052006>.

PAN, J.; LIU, Y.; WANG, J.; HESTER, A. **Key Enabling Technologies for Secure and Scalable Future Fog-IoT Architecture: A Survey**. 2018.

PANG, H.; TAN, K.-L. Authenticating query results in edge computing. In: **Proceedings. 20th International Conference on Data Engineering**. [S.l.: s.n.], 2004. p. 560–571.

PATTARANANTAKUL, M.; HE, R.; MEDDAHI, A.; ZHANG, Z. Secmano: Towards network functions virtualization (nfv) based security management and orchestration. In: IEEE. **2016 IEEE Trustcom/BigDataSE/ISPA**. [S.l.], 2016. p. 598–605.

PEREIRA, R. I.; DUPONT, I. M.; CARVALHO, P. C.; JUCá, S. C. Iot embedded linux system based on raspberry pi applied to real-time cloud monitoring of a decentralized photovoltaic plant. **Measurement**, v. 114, p. 286–297, 2018. ISSN 0263-2241. Disponível em: <https://www.sciencedirect.com/science/article/pii/S026322411730605X>.

PEREIRA, S.; KARIA, D. Ai use cases in operational support system and business support system. In: **2018 3rd International Conference on Communication and Electronics Systems (ICCES)**. [S.l.: s.n.], 2018. p. 15–20.

PHAM, Q.-V.; FANG, F.; HA, V. N.; PIRAN, M. J.; LE, M.; LE, L. B.; HWANG, W.-J.; DING, Z. A survey of multi-access edge computing in 5g and beyond: Fundamentals, technology integration, and state-of-the-art. **IEEE Access**, v. 8, p. 116974–117017, 2020.

POURGHASEMIAN, M.; ABEDI, M. R.; SALARHOSSEINI, S.; MOKARI, N.; JAVAN, M. R.; JORSWIECK, E. A. Ai-based and mobility-aware energy efficient resource allocation and trajectory design for nfv enabled aerial networks. **arXiv preprint arXiv:2105.10282**, 2021.

QI, D.; SHEN, S.; WANG, G. Towards an efficient vnf placement in network function virtualization. **Computer Communications**, v. 138, p. 81–89, 2019. ISSN 0140-3664. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0140366418308247>.

RAHMAN, D.; AMNUR, H.; RAHMAYUNI, I. Monitoring server dengan prometheus dan grafana serta notifikasi telegram. **JITSI : Jurnal Ilmiah Teknologi Sistem Informasi**, Politeknik Negeri Padang, v. 1, n. 4, p. 133–138, dez. 2020. Disponível em: <https://doi.org/10.30630/jitsi.1.4.19>.

RAHO, M.; SPYRIDAKIS, A.; PAOLINO, M.; RAHO, D. Kvm, xen and docker: A performance analysis for arm based nfv and cloud computing. In: **2015 IEEE 3rd Workshop on Advances in Information, Electronic and Electrical Engineering (AIEEE)**. [S.l.: s.n.], 2015. p. 1–8.

Ramalho, F.; Neto, A. Virtualization at the network edge: A performance comparison. In: **2016 IEEE 17th International Symposium on A World of Wireless, Mobile and Multimedia Networks (WoWMoM)**. [S.l.: s.n.], 2016. p. 1–6.

RATHOD, D. PERFORMANCE EVALUATION OF RESTFUL WEB SERVICES AND SOAP / WSDL WEB SERVICES. **International Journal of Advanced Research in Computer Science**, IJARCS International Journal of Advanced Research in Computer Science, v. 8, n. 7, p. 415–420, ago. 2017. Disponível em: <https://doi.org/10.26483/ijarcs.v8i7.4349>.

RAY, P. P.; KUMAR, N. Sdn/nfv architectures for edge-cloud oriented iot: A systematic review. **Computer Communications**, v. 169, p. 129–153, 2021. ISSN 0140-3664. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0140366421000396>.

REZENDE, A. G. P.; MARQUES, G.; FIALHO, R.; PASQUINI, R. Arcabouço para orquestração osmótica de cloud slices. In: **Anais do I Workshop de Teoria, Tecnologias e Aplicações de Slicing para Infraestruturas Softwarizadas**. Porto Alegre, RS, Brasil: SBC, 2019. p. 28–41. Disponível em: <https://sol.sbc.org.br/index.php/wslice/article/view/7720>.

RICHARDS, V. M.; MOREIRA, R.; SILVA, F. de O. Enabling the management and orchestration of virtual networking functions on the edge. In: **CLOSER**. [S.l.: s.n.], 2020. p. 338–346.

Rigazzi, G.; Kainulainen, J. P.; Turyagyenda, C.; Mourad, A.; Ahn, J. An edge and fog computing platform for effective deployment of 360 video applications. In: **2019 IEEE Wireless Communications and Networking Conference Workshop (WCNCW)**. [S.l.: s.n.], 2019. p. 1–6.

Riggio, R.; Khan, S. N.; Subramanya, T.; Yahia, I. G. B.; Lopez, D. Lightmano: Converging nfv and sdn at the edges of the network. In: **NOMS 2018 - 2018 IEEE/IFIP Network Operations and Management Symposium**. [S.l.: s.n.], 2018. p. 1–9.

RODRIGUES, G. D. C.; CALHEIROS, R. N.; GUIMARAES, V. T.; SANTOS, G. L. d.; CARVALHO, M. B. de; GRANVILLE, L. Z.; TAROUCO, L. M. R.; BUYYA, R. Monitoring of cloud computing environments: Concepts, solutions, trends, and future directions. In: **Proceedings of the 31st Annual ACM Symposium on Applied Computing**. New York, NY, USA: Association for Computing Machinery, 2016. (SAC '16), p. 378–383. ISBN 9781450337397. Disponível em: <https://doi.org/10.1145/2851613.2851619>.

RUPANI, A.; WHIG, P.; SUJEDIYA, G.; VYAS, P. A robust technique for image processing based on interfacing of raspberry-pi and fpga using iot. In: **2017 International Conference on Computer, Communications and Electronics (Comptelix)**. [S.l.: s.n.], 2017. p. 350–353.

SABHARWAL, N.; PANDEY, P. Working with prometheus query language (promql). In: _____. **Monitoring Microservices and Containerized Applications: Deployment, Configuration, and Best Practices for Prometheus and Alert Manager**. Berkeley, CA: Apress, 2020. p. 141–167. ISBN 978-1-4842-6216-0. Disponível em: <https://doi.org/10.1007/978-1-4842-6216-0_5>.

SAJJAD, H. P.; DANNISWARA, K.; AL-SHISHTAWY, A.; VLASSOV, V. Spanedge: Towards unifying stream processing over central and near-the-edge data centers. In: **2016 IEEE/ACM Symposium on Edge Computing (SEC)**. [S.l.: s.n.], 2016. p. 168–178.

SALHOFER, P. Evaluating the FIWARE platform. In: **Proceedings of the 51st Hawaii International Conference on System Sciences**. Hawaii International Conference on System Sciences, 2018. Disponível em: <https://doi.org/10.24251/hicss.2018.726>.

SANCHEZ-AGUERO, V.; VIDAL, I.; VALERA, F.; NOGALES, B.; MENDES, L. L.; DIAS, W. D.; FERREIRA, A. C. Deploying an nfv-based experimentation scenario for 5g solutions in underserved areas. **Sensors**, Multidisciplinary Digital Publishing Institute, v. 21, n. 5, p. 1897, 2021.

SATYANARAYANAN, M.; BAHL, P.; CACERES, R.; DAVIES, N. The case for vm-based cloudlets in mobile computing. **IEEE Pervasive Computing**, v. 8, n. 4, p. 14–23, 2009.

SHI, W.; CAO, J.; ZHANG, Q.; LI, Y.; XU, L. Edge computing: Vision and challenges. **IEEE Internet of Things Journal**, v. 3, p. 637–646, 2016.

SILVA, A. P.; TRANORIS, C.; DENAZIS, S.; SARGENTO, S.; PEREIRA, J.; LUíS, M.; MOREIRA, R.; SILVA, F.; VIDAL, I.; NOGALES, B.; NEJABATI, R.; SIMEONIDOU, D. 5ginfire: An end-to-end open5g vertical network function ecosystem. In: . [s.n.], 2019. v. 93, p. 101895. ISSN 1570-8705. Disponível em: <http://www.sciencedirect.com/science/article/pii/S1570870518309387>.

SONI, D.; MAKWANA, A. A survey on mqtt: a protocol of internet of things (iot). In: **International Conference On Telecommunication, Power Analysis And Computing Techniques (ICTPACT-2017)**. [S.l.: s.n.], 2017. v. 20.

STANDARD, O. Mqtt version 3.1. 1. **URL http://docs. oasis-open. org/mqtt/mqtt/v3**, v. 1, 2014.

SUKHIJA, N.; BAUTISTA, E. Towards a framework for monitoring and analyzing high performance computing environments using kubernetes and prometheus. In: **2019 IEEE SmartWorld, Ubiquitous Intelligence Computing, Advanced Trusted Computing, Scalable Computing Communications, Cloud Big Data Computing, Internet of People and Smart City Innovation (SmartWorld/SCALCOM/UIC/ATC/CBDCom/IOP/SCI)**. [S.l.: s.n.], 2019. p. 257–262.

TALEB, T.; SAMDANIS, K.; MADA, B.; FLINCK, H.; DUTTA, S.; SABELLA, D. On multi-access edge computing: A survey of the emerging 5g network edge cloud architecture and orchestration. **IEEE Communications Surveys Tutorials**, v. 19, n. 3, p. 1657–1681, 2017.

Tambe, S. D.; Mandge, Y.; Antony Franklin, A. Performance study of multi-access edge computing deployment in a virtualized environment. In: **2020 IEEE 3rd 5G World Forum (5GWF)**. [S.l.: s.n.], 2020. p. 424–429.

TIBURSKI, R. T.; MORATELLI, C. R.; JOHANN, S. F.; MATOS, E. de; HESSEL, F. A lightweight virtualization model to enable edge computing in deeply embedded systems. **Software: Practice and Experience**, 2021. Disponível em: <https://onlinelibrary.wiley.com/doi/abs/10.1002/spe.2968>.

TURNBULL, J. **Monitoring with Prometheus**. [S.l.]: Turnbull Press, 2018.

TUSA, F.; CLAYMAN, S.; VALOCCHI, D.; GALIS, A. Multi-domain orchestration for the deployment and management of services on a slice enabled nfvi. In: **2018 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN)**. [S.l.: s.n.], 2018. p. 1–5.

UPTON, E.; HALFACREE, G. **Meet the Raspberry Pi**. [S.l.]: John Wiley & Sons, 2012.

VALANTASIS, A.; MAKRIS, N.; ZARAFETAS, C.; KORAKIS, T. Experimental evaluation of orchestration software for virtual network functions. In: **2021 IEEE Wireless Communications and Networking Conference (WCNC)**. [S.l.: s.n.], 2021. p. 1–6.

VILALTA, R.; LERMA, A. M. López-de; LÓPEZ, V.; MRÓWKA, K.; SZWEDOWSKI, R.; NEIDLINGER, S.; FELIX, A.; STEVKOVSKI, Z.; TANCEVSKI, L.; SINGH, A. et al. Transport api extensions for the interconnection of multiple nfv infrastructure points

of presence. In: OPTICAL SOCIETY OF AMERICA. **Optical Fiber Communication Conference**. [S.l.], 2019. p. W1G–2.

WANG, H.; FAPOJUWO, A. O. A survey of enabling technologies of low power and long range machine-to-machine communications. **IEEE Communications Surveys Tutorials**, v. 19, n. 4, p. 2621–2639, 2017.

WATADA, J.; ROY, A.; KADIKAR, R.; PHAM, H.; XU, B. Emerging trends, techniques and open issues of containerization: A review. **IEEE Access**, v. 7, p. 152443–152472, 2019.

WIRAWAN, I. M.; WAHYONO, I. D.; IDFI, G.; KUSUMO, G. R. Iot communication system using publish-subscribe. In: **2018 International Seminar on Application for Technology of Information and Communication**. [S.l.: s.n.], 2018. p. 61–65.

WU, F.; QIU, C.; WU, T.; YUCE, M. R. Edge-based hybrid system implementation for long-range safety and healthcare iot applications. **IEEE Internet of Things Journal**, p. 1–1, 2021.

YASSEIN, M. B.; SHATNAWI, M. Q.; ALJWARNEH, S.; AL-HATMI, R. Internet of things: Survey and open issues of mqtt protocol. In: **2017 International Conference on Engineering MIS (ICEMIS)**. [S.l.: s.n.], 2017. p. 1–6.

YONGGUO, J.; QIANG, L.; CHANGSHUAI, Q.; JIAN, S.; QIANQIAN, L. Message-oriented middleware: A review. In: **2019 5th International Conference on Big Data Computing and Communications (BIGCOM)**. [S.l.: s.n.], 2019. p. 88–97.

YOUSEFPOUR, A.; FUNG, C.; NGUYEN, T.; KADIYALA, K.; JALALI, F.; NIAKANLAHIJI, A.; KONG, J.; JUE, J. P. All one needs to know about fog computing and related edge computing paradigms: A complete survey. **Journal of Systems Architecture**, v. 98, p. 289–330, 2019. ISSN 1383-7621. Disponível em: <https://www.sciencedirect.com/science/article/pii/S1383762118306349>.

YUN, J.; PARK, K.-W.; KOO, D.; SHIN, Y. Lightweight and seamless memory randomization for mission-critical services in a cloud platform. **Energies**, v. 13, n. 6, 2020. ISSN 1996-1073. Disponível em: <https://www.mdpi.com/1996-1073/13/6/1332>.

ZHANG, Q.; LIU, L.; PU, C.; DOU, Q.; WU, L.; ZHOU, W. A comparative study of containers and virtual machines in big data environment. In: **2018 IEEE 11th International Conference on Cloud Computing (CLOUD)**. [S.l.: s.n.], 2018. p. 178–185.

ZHONG, X.; LIANG, Y. Raspberry pi: An effective vehicle in teaching the internet of things in computer science and engineering. **Electronics**, v. 5, n. 3, 2016. ISSN 2079-9292. Disponível em: <https://www.mdpi.com/2079-9292/5/3/56>.

ZHOU, Z.; TAN, L.; GU, B.; ZHANG, Y.; WU, J. Bandwidth slicing in software-defined 5g: A stackelberg game approach. **IEEE Vehicular Technology Magazine**, v. 13, n. 2, p. 102–109, 2018.

ÖSTBERG, P.-O.; BYRNE, J.; CASARI, P.; EARDLEY, P.; ANTA, A. F.; FORSMAN, J.; KENNEDY, J.; DUC, T. L.; MARIñO, M. N.; LOOMBA, R.; PEñA, M. L.; VEIGA, J. L.; LYNN, T.; MANCUSO, V.; SVOROBEJ, S.; TORNEUS, A.; WESNER,

S.; WILLIS, P.; DOMASCHKA, J. Reliable capacity provisioning for distributed cloud/edge/fog computing applications. In: **2017 European Conference on Networks and Communications (EuCNC)**. [S.l.: s.n.], 2017. p. 1–6.