

UNIVERSIDADE FEDERAL FLUMINENSE

ANSELMO LUIZ ÉDEN BATTISTI

**V-PRISM: AN EDGE-BASED ARCHITECTURE
TO VIRTUALIZE MULTIMEDIA SENSORS IN
THE INTERNET OF MEDIA THINGS**

NITERÓI

2020

UNIVERSIDADE FEDERAL FLUMINENSE

ANSELMO LUIZ ÉDEN BATTISTI

**V-PRISM: AN EDGE-BASED ARCHITECTURE
TO VIRTUALIZE MULTIMEDIA SENSORS IN
THE INTERNET OF MEDIA THINGS**

Master Thesis presented to the Programa de
Pós-Graduação em Computação of the Uni-
versidade Federal Fluminense as requirement
to obtain the Degree of Master in Computa-
tion. Area: Computer Systems

Advisor:

DÉBORA CHRISTINA MUCHALUAT SAADE

Co-Advisor:

FLÁVIA COIMBRA DELICATO

NITERÓI

2020

Ficha catalográfica automática - SDC/BEE
Gerada com informações fornecidas pelo autor

B336v Battisti, Anselmo Luiz Édén
V-PRISM: AN EDGE-BASED ARCHITECTURE TO VIRTUALIZE MULTIMEDIA
SENSORS IN THE INTERNET OF MEDIA THINGS / Anselmo Luiz Édén
Battisti ; Débora Christina Muchaluat-Saade, orientadora ;
Flavia Coimbra Delicato, coorientadora. Niterói, 2020.
108 p. : il.

Dissertação (mestrado)-Universidade Federal Fluminense,
Niterói, 2020.

DOI: <http://dx.doi.org/10.22409/PGC.2020.m.04524565922>

1. Internet of Things. 2. Internet of Media Things. 3.
Virtual Multimedia Sensor. 4. Edge Computing. 5. Produção
intelectual. I. Muchaluat-Saade, Débora Christina,
orientadora. II. Delicato, Flavia Coimbra, coorientadora. III.
Universidade Federal Fluminense. Instituto de Computação.
IV. Título.

CDD -

Anselmo Luiz Éden Battisti

V-PRISM: An Edge-Based Architecture to Virtualize Multimedia Sensors in the
Internet of Media Things

Master Thesis presented to the Programa de
Pós-Graduação em Computação of the Uni-
versidade Federal Fluminense as requirement
to obtain the Degree of Master in Comput-
ing. Area: Computer Systems

Approved in August 2020

APPROVED BY



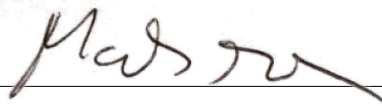
Prof.ª Débora Christina Muchaluat Saade - Advisor, UFF



Prof.ª Flávia Coimbra Delicato - Co-Advisor, UFF



Prof. Célio Vinicius Neves de Albuquerque, UFF



Prof. Markus Endler, PUC-RIO

Niterói

2020

Acknowledgements

I want to thank God for giving me the strength to reach another life goal.

I am grateful for the unconditional support of my dear parents, Fátima and Janir, my loved partner Marielle, my sister Monalisa, my brother Abraão, my favourite niece Bianca and my brother-in-law André.

I want to thank my advisors, Professor Débora and professor Flávia for all the time dedicated to the project, and also for all the patience they had with me!

I also grateful Professor Célio for the guidance and suggestions on how to proceed at the beginning of the master's degree.

I also thank my friend Fernando Hallberg for the support given in this and other projects. I also thank my colleagues Bruno, Nilson and Sidney, for taking the time to participate in one of the experiments in this dissertation.

Finally, I thank all my friends and family for the affection and friendship. I am also grateful for the financial support provided by the Fundação de Amparo à Pesquisa do Estado de São Paulo (FAPESP). Last but not least, I thank everyone at the UFF Computing Institute for welcoming me so well.

Resumo

Os sensores multimídia tornaram-se recentemente uma fonte de dados significativa na Internet das Coisas (IoT - *Internet of Things*), dando origem à Internet das Coisas Multimídia (IoMT - *Internet of Media Things*). Aplicações multimídia geralmente são sensíveis à latência e, por conta disso, o processamento de dados na nuvem nem sempre é adequado. Uma estratégia para minimizar o atraso é processar os fluxos multimídia mais perto das fontes de dados, explorando os recursos na borda da rede. Seguindo essa estratégia, esta dissertação propõe uma arquitetura, chamada V-PRISM, para virtualizar e gerenciar sensores multimídia com componentes implantados e executados em múltiplos nós de borda. A arquitetura fornece componentes para orquestração, alocação de recursos, monitoramento de ambiente, intermediação de mensagens, compartilhamento de fluxo multimídia, virtualização de dispositivos multimídia, gerenciamento de solicitações, além de outras funções. A entidade que processa um fluxo multimídia é denominada *Virtual Multimedia Sensor* (VMS). Eles são uma camada de abstração entre a aplicação IoMT e dispositivos físicos que produzem o fluxo multimídia. Essa estratégia pode reduzir a complexidade existente pela heterogeneidade nos ambientes de IoT. Como múltiplos nós de borda podem compor o ambiente V-PRISM, fornecemos um componente para alocação dinâmica de VMSs que pode ser estendido para executar diferentes tipos de algoritmos de alocação de recursos. V-PRISM foi validado através da implementação de uma prova de conceito chamada ALFA. Ela fornece os componentes lógicos descritos na arquitetura e vários tipos de VMSs. Os experimentos mostraram que a adoção do V-PRISM pode reduzir o consumo dos recursos nos dispositivos IoT, no tráfego de rede e atraso fim-a-fim das aplicações, além de aumentar o ROI (*Return On Investment*) para provedores de infraestrutura IoT. Também foi realizado um estudo informal com desenvolvedores, mostrando que a adoção do V-PRISM pode trazer também benefícios para o desenvolvimento de VMSs. Outra contribuição deste trabalho foi o desenvolvimento de uma categorização hierárquica de VMS fundamentada nos recursos e nas funções fornecidas por cada tipo de VMS.

Palavras-Chaves: Sensor Virtual Multimídia, Internet das Coisas, Internet das Coisas Multimídia, Computação na Borda da Rede.

Abstract

Multimedia sensors have recently become a significant data source in the Internet of Things (IoT), giving rise to the Internet of Media Things (IoMT). Since multimedia applications are usually latency-sensitive, data processing in the cloud is not always suitable. A strategy to minimize delay is to process the multimedia streams closer to the data sources, exploiting the resources at the edge of the network. Following this strategy, we propose V-PRISM, an architecture to virtualize and manage multimedia sensors with components deployed and executed in multiple edge nodes. The architecture provides components for orchestration, resource allocation, environment monitor, message broker, multimedia stream share, multimedia device virtualization, request management, besides other functions. The entity that processes the multimedia stream is called Virtual Multimedia Sensor (VMS). VMSs are an abstraction layer between IoMT applications and physical multimedia devices that produce the multimedia stream. This strategy can reduce the complexity due to the heterogeneity in IoT environments. As multiple edge nodes can compose V-PRISM environment, we provide a dynamic VMS allocation component used to automatic allocate VMS in edge nodes. This component can be extended to run different types of resource allocation algorithms. We validated V-PRISM through an implementation named ALFA, a proof-of-concept (PoC) that provides most components described in the architecture and multiple VMS types. The experiments show that the adoption of V-PRISM can reduce resource consumption of IoT devices, network traffic, and end-to-end delay while increasing the ROI (Return On Investment) for infrastructure providers. We also conducted an informal study with developers, which shows that the adoption of V-PRISM can bring benefits to the development of VMSs. Another contribution of this work is a hierarchical categorization of VMS based on features and functions they provide.

Keywords: Virtual Multimedia Sensors, Internet of Things, Internet of Media Things, Edge Computing.

List of Figures

2.1	Smart cities and IoMT [39].	21
2.2	Cloud of Things approach X Edge computing approach.	23
3.1	MEC system reference architecture [29]	37
4.1	V-PRISM three-tier architecture overview	41
4.2	Description of an environment where V-PRISM can be deployed	41
4.3	VMS Hierarchical Categorization	42
4.4	Traditional and virtual sensor approach in surveillance systems	44
4.5	Example of security threat in legacy IoT environment	45
4.6	VMS converting an audio stream codec format	45
4.7	VMS sound selector	46
4.8	<i>Aggregator</i> VMS of producing a video mosaic	47
4.9	Gun shoot detection a <i>Detector</i> VMS	47
4.10	V-PRISM Logic Components Architecture	49
4.11	Relationship between VD and VMS	52
4.12	VRM state diagram	53
4.13	USB Webcam attached to a Raspberry	59
4.14	Stream Share component operation	60
4.15	VMS creation sequence diagram	61
4.16	Interaction between CMB and Virtual Devices	63
5.1	ALFA Deployment Diagram	65
5.2	Web application module to manage edge nodes	67
5.3	Docker Overview	68

5.4	Overlay Docker Network ¹	68
5.5	Script to install the Docker Images from VMS and VD in the edge node . .	70
5.6	VMS and VD running in Docker containers	71
5.7	Web Interface Application	72
5.8	Web application bind function	73
5.9	MQTT client displaying the topic used to bind VMS and VD	74
5.10	VMS Video Mosaic	77
6.1	V-PRISM analyzed from different perspectives	78
6.2	Design of the experiment about resource usage in IoT Device and IoT Network	80
6.3	IoMT device CPU consumption	82
6.4	IoMT device network bandwidth consumption	82
6.5	IoMT device battery consumption	83
6.6	Experiment design of audio processing in edge and cloud nodes	86
6.7	Example of Video frame with the QR Code	88
6.8	Experiment design to obtain metrics for total latency	89
6.9	<i>First Frame Detection</i> (FFD)	89
6.10	<i>Frame Detection Difference</i> (FDD)	90
6.11	Fault-tolerant scenario	95
6.12	ROI increasing scenario	95
6.13	Execution of on-demand multimedia stream pipeline	96

List of Tables

2.1	Comparison between IoT and IoMT [94]	19
2.2	Definition of Edge Computing	24
2.3	Definition of Virtual Sensor	27
3.1	Comparison between virtual things approach	34
3.2	Comparison between Sensor Virtualization Architectures	38
6.1	Goals definition	79
6.2	Questions for G1 goal	79
6.3	Summary of the metrics used to answer the questions of G1 goal.	80
6.4	Questions for G2 goal	84
6.5	Summary of the metrics used to answer the questions of G2 goal.	85
6.6	Interval between successive noise detection events.	87
6.7	FRL, QFD and QDE metrics over 450 QR Code frames.	88
6.8	Questions used for addressing G3 goal	91
6.9	Summary of the metrics used to answer the questions of G3 goal.	91
6.10	New VMS type description	91
6.11	Form to collect the perception of the developers about V-PRISM	93
6.12	Answer snippets that help to understand metric ETU	93
6.13	Answer snippets that help to understand metric VEF	94

List of Acronyms

API	: Application Programming Interface;
ATAM	: Architecture Tradeoff Analysis Method;
BCO	: Battery Consumed;
CCO	: CPU Consumed;
CMB	: Control Message Broker;
CoT	: Cloud of Things;
DTN	: Detection Time Noise;
EC	: Edge Computing;
EF	: Edge Function;
ENM	: Edge Node Manager;
ESO	: European Standards Organization;
ETSI	: European Telecommunications Standards Institute;
ETU	: Easy to Use;
FDM	: Frame Detection Difference;
FFD	: First Frame Detection;
FRL	: Frame Loss;
GQM	: Goal Question Metric;
IIoT	: Industrial Internet of Thing;
IX	: Internet Exchange Point;
IoMT	: Internet of Media Things;
IoT	: Internet of Things;
JSON	: JavaScript Object Notation;
LXC	: Linux Containers;
MBT	: Megabits Transferred;
MEC	: Multi-access Edge Computing;
MQTT	: MQ Telemetry Transport;
MSaaS	: Multimedia Sensing as a Service;
OS	: Operating System;
PoC	: Proof-of-Concept;

QDE	: Quantity Data Extracted;
QFD	: Quantity Frame Detected;
QoS	: Quality of Service;
RAM	: Resource Allocation Manager;
RAN	: Radio Access Network;
REST	: Representational State Transfer;
ROI	: Return On Investment;
RTSP	: Real-Time Streaming Protocol;
S2T	: String to Text;
SBC	: Single-board Computer;
SS	: Stream Sharer;
SVA	: Sensor Virtualization Architectures;
V-PRISM	: Virtual Programmable IoT Sensor for Multimedia;
VD	: Virtual Device;
VDM	: VD Manager;
VEF	: V-PRISM Effectiveness;
VMS	: Virtual Multimedia Sensor;
VN	: Virtual Node;
VNF-FG	: Virtual Node Function Forwarding Graph;
VRM	: VMS Request Manager;
VS	: Virtual Sensor;
WAV	: WAVEform audio format;
WMSN	: Wireless Multimedia Sensor Network;
WSAN	: Wireless Sensor and Actuator Network;
WSN	: Wireless Sensor Network.

Contents

1	Introduction	14
1.1	Research Questions and Goals	16
1.2	Main Contributions	17
1.3	Organization	17
2	Background	19
2.1	Internet of Media Things and its Applications	19
2.2	Edge Computing and Multimedia Streams	22
2.3	Containers in Edge Computing	25
2.4	Virtual Sensors	27
3	Related Work	29
3.1	Traditional and Multimedia Sensor Virtualization	29
3.2	Sensor Virtualization Architectures	34
4	V-PRISM Proposal	39
4.1	The three-tier architecture	39
4.2	VMS Categories	42
4.2.1	Replicator	43
4.2.2	Improver	44
4.2.3	Converter	45
4.2.4	Selector	45
4.2.5	Aggregator	46

4.2.6	Detector	47
4.2.7	Transformer	48
4.3	V-PRISM Logic Components	48
4.3.1	Virtual Multimedia Sensor	49
4.3.2	Virtual Device	50
4.3.3	VMS Registry & VMS Request Manager	52
4.3.4	Maestro	53
4.3.5	Resource Allocation Manager	54
4.3.5.1	Resource Allocation Heuristic Model	55
4.3.6	Virtualization Engine	56
4.3.7	Edge Node Manager	57
4.3.8	Environment Monitor & VN Monitor	57
4.3.9	VMS Manager	58
4.3.10	VD Manager	58
4.3.11	Stream Sharer	60
4.4	V-PRISM Operation	60
4.4.1	V-PRISM Deployment	60
4.4.2	Initialization of a VMS	61
4.4.3	Multimedia Stream Sharer	62
4.4.4	Control Message Broker	62
5	ALFA: An Implementation of V-PRISM	64
5.1	ALFA Main Technologies	64
5.1.1	Edge Nodes	66
5.1.2	Docker	67
5.1.3	VD and VMS in Containers	69
5.1.4	Main Used APIs	71

5.1.5	Web Interface Application	72
5.1.6	Resource Allocation Management	74
5.2	Implemented Virtual Devices	75
5.3	Implemented Virtual Multimedia Sensors	76
6	Evaluation	78
6.1	Resource usage in IoT Device and IoT Network	79
6.2	Comparison Between Edge and Cloud	84
6.2.1	Noise Detector Experiment	85
6.2.2	Video Processing Experiment	87
6.3	Development of New VMS Types	91
6.4	Other Examples of V-PRISM Use Cases	94
7	Conclusion	97
	References	101

Chapter 1

Introduction

With the widespread of the Internet of Things (IoT) and its integration with cloud computing, a new paradigm called Cloud of Things (CoT), or Cloud-assisted IoT, has recently emerged [15], which exploits the synergy between IoT and the cloud. In this scenario, ambients, people, and objects are continuous sources of data generation that are consumed by applications that receive processed data streams through the cloud. The cloud provides a vast amount of processing and storage capabilities for the data generated by IoT devices while abstracting their heterogeneity. By offering sensing and actuation as a service, cloud providers broaden their portfolio of services for users and applications.

One of the enabling technologies of the CoT paradigm is virtualization. It refers to the process of building a logical abstraction of hardware and/or software features. Virtualization is at the core of cloud computing. It allows hiding from clients the variety of types of infrastructures, platforms and data available at the back-end, promoting the decoupling between entities that produce and consume resources and facilitating application delivery. It has also been used in other contexts, such as communication networks [4], and more recently in sensors and sensor networks [36]. In wireless sensor network systems, virtualization allows the creation of virtual sensor nodes, which abstract physical sensor nodes and are responsible for providing services to the application and end users. The mapping of physical sensors to their virtual counterparts is done through a virtualization model. Several virtualization models are described in the literature specifically designed for wireless sensors and sensor networks [41]. Such models are tailored for sensors with reduced processing, memory and battery capacities, and equipped with wireless interfaces for communication.

In the context of our work, among the various types of sensors that compose the IoT infrastructure, and therefore, the CoT, multimedia devices have increasingly stood out.

In a report produced by Cisco [10], it is estimated that by 2022 about 80% of the Internet bandwidth will consist of multimedia streams. The relevance of this type of device has given rise to the concept of Internet of Media Things (IoMT)¹ [6], or even Multimedia Internet of Things (M-IoT) [59]. In this work, we will adopt the term IoMT also adopted in the pattern ISO/IEC 23093-1:2020².

In the IoMT, sensors are cameras and microphones with limited processing and storage capacity, which connect to heterogeneous devices and diverse applications. Traditional sensors, such as temperature, pressure, and light detection, typically generate discrete data. On the other hand, multimedia sensors produce continuous, massive data streams with nontrivial structure and temporal significance. Such features make the processing of multimedia streams more complex than traditional sensor data. Moreover, there is high heterogeneity in the communication protocols and data formats supported by multimedia devices, adding an extra level of complexity in the acquisition, processing, and consumption of data.

The particular features of multimedia streams in the context of IoMT make the use of existing sensor virtualization models unsuitable. It becomes necessary to design virtualization models tailored for multimedia sensors. Considering the high heterogeneity of multimedia devices and the specific requirements of multimedia stream processing, the design of models and mechanisms to abstract multimedia sensors will enable the development of innovative services in several relevant areas such as health, security [60], education and entertainment.

In a typical CoT system, sensor-generated multimedia streams are virtualized in the cloud and delivered on-demand to applications. Such an approach implies that the massive amount of data generated needs to be transferred from the devices to the cloud, thus demanding a considerable amount of bandwidth. Moreover, cloud computing incurs high latency for data exchange between cloud servers and devices. Therefore, the cloud-based IoT model may not be able to meet the strict time requirements of multimedia applications. A promising strategy that has recently gained momentum to decrease the latency for applications is to migrate (part of) processing from the cloud to the edge of the network, placing it closer to the sensing devices (data sources). Such strategy is exploited by recent paradigms of Edge [91] and Fog computing [58].

In this work, we use the terms edge and fog interchangeably. We adopt the definition of

¹IoMT is also called Internet of Multimedia Things.

²<https://www.iso.org/news/ref2449.html>

Edge Computing (EC) provided by [35] where it is a horizontal, system-level architecture that distributes computing, storage, control and networking functions closer to the users along a cloud-to-thing continuum. The use of EC brings benefits as the decreasing delay and bandwidth consumption at the network core, better use of available resources, and in some cases, increasing data security/privacy. Multimedia applications benefit from running in low-latency environments, so designing an architecture for virtualization of multimedia sensors at the edge is a promising approach.

The inclusion of elements at the edge of the network in a CoT environment creates a new computing layer. Therefore, we consider the CoT as a three-tiered ecosystem, encompassing the cloud, the edge and the IoT device tiers.

1.1 Research Questions and Goals

Based on the challenges discussed in the previous section, we defined the following research questions to be addressed in this work:

RQ1: *Can a multimedia sensor virtualization architecture enable a single multimedia device to provide multimedia streams for different applications?*

RQ2: *Can a multimedia sensor virtualization architecture reduce CPU usage and battery consumption in IoT devices?*

RQ3: *Can a multimedia sensor virtualization architecture reduce bandwidth usage in the IoT network?*

RQ4: *Can an edge-based multimedia sensor virtualization architecture reduce the delivery time of a multimedia stream to an IoMT application, compared to using a virtual multimedia sensor hosted in the cloud?*

Based on this research questions, the general goal of this work is proposing an architecture that enables the virtualization of multimedia sensors in edge computing environments. The architecture components are responsible for processing multimedia streams produced by physical multimedia devices. The stream processing is performed by entities named virtual multimedia sensors (VMS). The architecture will follow a three-tier approach encompassing the cloud, the edge, and the things. All of the architecture components are deployed in the edge tier.

Besides proposing the logic components of the architecture and their operation, we implemented a prototype named ALFA that worked as Proof-of-Concept (PoC). In ALFA, we developed and validated the logical components, besides using different technologies to create different parts of the architecture. Furthermore, we developed a set of VMSs and virtual devices to evaluate our approach in real environments.

1.2 Main Contributions

This work provides the following contributions:

- Proposal of an architecture to manage VMSs in edge computing environments;
- Definition of a classification for VMSs based on the functionalities provided and resources consumed;
- Definition of templates for facilitating the creation of new VMS types and Virtual Device Types.

1.3 Organization

The remainder of this work is structured as follows. In Chapter 2, we discuss the theoretical background that represents state-of-the-art about Internet of Media Things (IoMT) and its applications. We also discuss the adoption of technologies like multimedia processing and container virtualization in the edge computing environments. Finally, we define the virtual sensor paradigm and its applications.

In Chapter 3, we present other projects that are related to our proposed approach. Firstly, we analyze various cases where the authors use the virtual sensor paradigm, and include strategies in traditional and multimedia sensors. After that, we examine multiple sensor virtualization architectures already proposed and compare them with our proposed approach.

In Chapter 4, we describe our proposed architecture. Firstly we present the three-tier architecture and its implications. After that, we propose a VMS categorization. Finally, we also explain the logic components and the operation of our architecture. Besides that, we also present a resource allocation algorithm for virtual sensor placement in edge nodes.

In Chapter 5, we present ALFA, an implementation as a Proof-of-Concept (PoC) of our proposal. ALFA is a functional implementation of the logic components and provides all the mechanisms necessary to virtualize multimedia sensors in edge environments. Another relevant characteristic is that the PoC source code is available in GitHub under GPL license.

In Chapter 6, we present evaluation of our proposal. We conducted experiments to analyze if the adoption of our approach improves the resource usage in IoT Device and IoT Network. We also examine the benefits that our approach brings in contrast with the Cloud of Things approach for multimedia stream processing. Finally, we investigate if the proposed architecture can be used for the development of new VMSs and if it can improve the return on investment in the IoMT environment.

Finally, in Chapter 7, we conclude discussing our main contributions, limitations and future work.

Chapter 2

Background

This chapter presents the main concepts and technologies used in this work. We discuss the Internet of Media Things in Section 2.1, edge computing and multimedia streams in Section 2.2, containers in edge computing in Section 2.3 and finally the concept of virtual sensors in Section 2.4.

2.1 Internet of Media Things and its Applications

Multimedia sensors, such as cameras and microphones, are becoming a significant source of data on the IoT. The proliferation of this type of device in IoT environments has given rise to a new subset of the IoT called Internet of Media Things (IoMT). Beyond the traditional IoT challenges, IoMT has specific issues that must be addressed to enable its adoption on a large scale. Table 2.1 lists some fundamental differences between IoT and IoMT.

Table 2.1: Comparison between IoT and IoMT [94]

IoT Scenario	IoMT Scenario
Linear Data	Bulky Data
Low Processing	High Processing
Low Storage	Massive Storage
Low Bandwidth	High Bandwidth
Delay Tolerant	Delay Sensitive
Low Power Consumption	High Power Consumption
Simple Data Encoding	Complex Media Encoding

In the IoMT, the data collected, processed, transported, and consumed are multimedia streams. The nature of a multimedia stream is different from the data produced by

traditional sensors like temperature and humidity sensors. Traditional sensors produce discrete data, and the data structure is simple. In contrast, multimedia sensors produce continuous data, and its data structure is complex. Moreover, the bulky nature of the multimedia stream, in contrast to the IoT limited network bandwidth, increases the challenges to satisfy the application QoS requirements [59]. It means that new techniques, standards, and frameworks must be developed to deal with these new challenges.

It is relevant to highlight that, besides IoMT, other approaches deal with multimedia streams too. One of them is Wireless Sensor Network (WSN). A WSN [69] can be defined as a network of small embedded devices, called sensors, which use wireless communication following an ad hoc configuration. A Wireless Multimedia Sensor Network (WMSN) is a particular case of WSN where sensors are multimedia devices. In [6], the authors define that the main limitation of WSN and WMSN is that usually the deployment scenario is typically a fixed architecture with restrictive mobility, a pre-defined set of multimedia devices can be adopted, and a pre-defined set of functionalities are known. These strategies are inefficient and lead to redundant deployments when new applications are needed [41].

One of the first definitions of IoMT was presented in [6]. They define IoMT as *the global network of interconnected multimedia things which are uniquely identifiable and addressable to acquire sensed multimedia data or trigger actions as well as having capability to interact and communicate with other multimedia and not multimedia devices and services, with or without direct human intervention*. This is a broad definition, encompassing many study fields.

There are many fields where IoMT will cause significant impact. Some of them are security, healthcare, mobility and supply chain [59]. In particular, smart cities environments will be enriched with innovative services based on IoMT, and these services will improve daily citizen lives. In Figure 2.1, we can see a smart city scenario where a vast number of heterogeneous multimedia actuators (public display, audio players, video players) and sensors are connected to provide services to a variety of applications. However, despite the significant potential of IoMT, it is essential to remind that the integration of multimedia services and devices into the IoT remains a challenge. This challenge was mainly raised by limited capabilities of IoT devices and distinct characteristics of multimedia applications, such as latency constraints [39].



Figure 2.1: Smart cities and IoMT [39].

With the availability of multimedia data streams, service providers are developing new types of services. Some areas that are changing drastically are security [60], road monitoring [21], environment monitoring [66], etc. Many possible IoMT applications can be seen in Figure 2.1. These applications are connecting multimedia things with each other and with other smart things. For instance, an IoT motion sensor can discover and directly interact with available IoT cameras and trigger surveillance of the area, as presented in [24]. The multimedia stream from a camera can be processed by a neural network to recognize faces to identify unauthorized access, and it can generate a message alert to the security staff. The possibilities are vast.

Another innovative use case of the integration between multimedia devices and IoT devices was presented in [84]. In that work, the authors show the use of drones to detect and clean up graffiti in buildings and road signals. In this scenario, the deployment of IoMT applications faces heterogeneity devices challenges. Drones from different vendors and with different sensors and functionalities can be used for this task. Techniques to overcome this scenario must be developed. Another relevant aspect presented in that work is the use of machine learning in restricted devices, like drones for example.

Differently from the IoMT scenario presented before, we have a typical case of a WMSN at Niterói/RJ city. In 2015, a surveillance system with 350 cameras was deployed in the city. Technical details can be found in the public edict No. 001/2014¹. Despite the innovation, the approach adopted is not friendly for sharing multimedia data to different applications, since new services cannot easily use the already deployed camera infrastructure.

The advance of IoT brings new light to a dynamic scenario where multimedia sensors

¹<http://www.niteroi.rj.gov.br/licitacao/sma/2014/cp-001-14.pdf>

are part of our daily life and whose multimedia data can be used unpredictably by IoT developers. The deployment of multimedia sensors is not an easy task. This difficulty is raising a new type of service called multimedia sensing as a service (MSaaS) [83]. In this scenario, infrastructure providers offer multimedia streams as services to IoMT applications. This new data availability enables a fast growth of this type of applications. Thus, the creation of an architecture that virtualizes multimedia devices can be seen as a potential trend.

In IoMT environments, some trade-offs must be addressed. For example, to minimize the size of a multimedia stream, the device must run a more powerful encoding method, which will consume more energy and CPU [6]. The tuning of all the variables presented in the IoMT stack to obtain the QoS level required by the IoMT is a complex task.

Multimedia streams are usually complex and bulky. Their syntax and semantics are not obvious and they are tolerant for packet loss. Also, several compression algorithms can be applied to multimedia streams, so the amount of data in the stream can be reduced without losing its semantics. Multimedia streams can be combined and new information can be extracted. Besides, flows can be handled in different environments, such as edge devices, edge nodes, public clouds and private clouds as presented in [92].

Specific features of multimedia streams in the context of IoMT make the use of already existing traditional sensor virtualization models not suitable. There is a need to design models tailored for multimedia sensors. We claim that models and mechanisms to abstract the complexity of multimedia sensors will enable the development of innovative services in relevant areas such as healthcare, security [60], education and entertainment.

As previously mentioned, IoMT applications are typically latency-sensitive. It means that the adoption of the cloud to process multimedia streams sometimes is not viable because the Internet provides best effort service and streams may suffer high latency and jitter. In the next section, we will discuss the adoption of the edge-computing paradigm to deal with multimedia streams.

2.2 Edge Computing and Multimedia Streams

The Cloud of Things (CoT) paradigm implies that multimedia streams devices are virtualized in the cloud, and the processed stream is delivered on-demand to applications, as depicted in Figure 2.2(a). This strategy has some drawbacks. For example, the massive amount of data generated needs to be transferred from the devices to the cloud. Hence,

the CoT models cannot reach the strict computing time requirements of multimedia applications. A new strategy emerged recently propose to migrate part of the processing from the cloud to the edge of the network, placing the processing closer to multimedia devices, as shown in Figure 2.2(b). Such strategy is exploited by recent paradigms of Edge [91] [81] and Fog computing [58]. Although there are some differences between fog and edge computing, in this work we adopt that term fog computing and edge computing as synonymous.

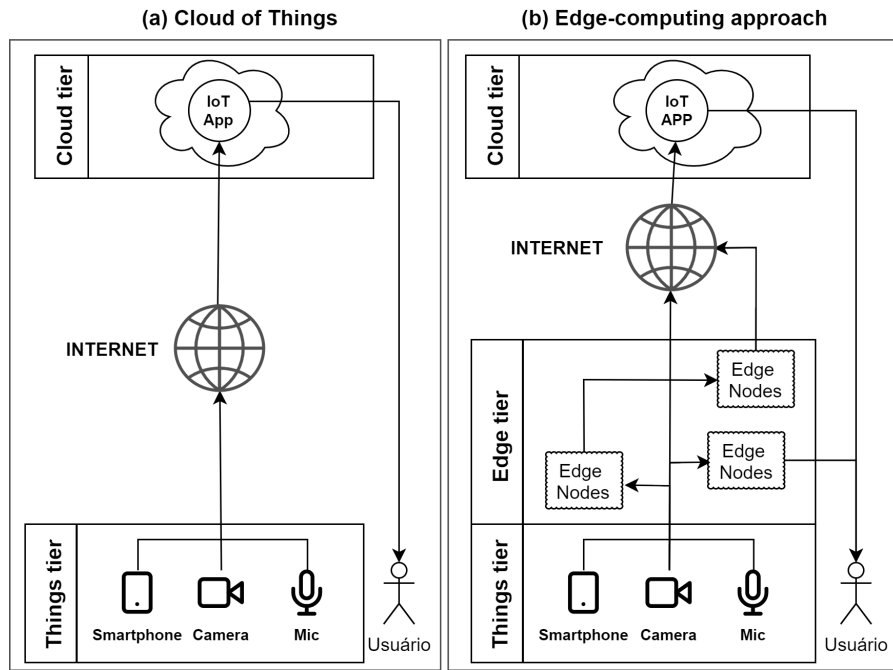


Figure 2.2: Cloud of Things approach X Edge computing approach.

In this section, the relationship between edge computing (EC) and multimedia streams processing will be discussed. We will define EC and present the main motivations to its adoption in our work. Table 2.2 presents some definitions of EC according to different authors.

In this work, we adopt the definition provided by OpenFog Consortium in [35], "*edge computing is a horizontal, system-level architecture that distributes computing, storage, control and networking functions closer to the users along a cloud-to-thing continuum*". The edge computing approach is depicted in Figure 2.2(b). EC addresses a range of issues, such as low-latency requirements, geolocalization and heterogeneity of IoT devices, data security and privacy. These issues can be addressed in different approaches for enabling EC, namely, Cloudlet [75] and Multi-access Edge Computing (MEC) [65].

The Cloudlet approach is defined by [75] as a trusted, resource-rich computer or

Table 2.2: Definition of Edge Computing

Author	Definition
Pang et al. [62]	Is the strategy that pushes business logic and data processing from corporate data centers out to proxy servers at the “edge” of the network.
Shi et al.[79]	Any computing and network resources along the path between data sources and cloud data centers.
Satyanarayanan et al. [74]	Computing and storage nodes placed at the Internet’s edge in close proximity to mobile devices or sensors.
OpenFog Consortium [35]	A horizontal, system-level architecture that distributes computing, storage, control and networking functions closer to the users along a cloud-to-thing continuum.
Morabito et al. [54]	Edge layer owns the crucial role of bridging and interfacing the central cloud with IoT. Essentially, an edge element in this layer can be characterized by a small to medium size computing entity that aims to provide extra computing, storage, and networking resources to the applications deployed across IoT devices, edge and cloud.

cluster of computers that is well-connected to the Internet and available for use by nearby mobile devices. In that, strategy the content or application that is needed by the user is preloaded in a small nearby datacenter transparently. The user acts as a thin client connected a virtual machine.

The MEC approach was initially called Mobile Edge Computing. It was defined in [65] as a new platform that provides information technologies and cloud-computing capabilities within the Radio Access Network (RAN) near mobile subscribers. But in 2016, the European Telecommunications Standards Institute - Industry Specification Group (ETSI- ISG) changed "mobile" to "Multi-access" expanding the definition of MEC to aggregate heterogeneous network devices including WiFi and fixed access technologies [55]. To the 5G growth, the deployment of a massive amount of base stations and access points will be needed. MEC will harvest the enormous idle computation and storage resources available at network edges [51].

The distribution of multimedia stream processing among different edge nodes was studied in [7]. They proposed an architecture to perform anomaly detection at the edge. The edge device that collects data from the microphone performed a transformation over an audio stream and delivered it to an IoT gateway that runs a machine learning program that detects an anomaly in an office environment.

The use of EC and distribution of the multimedia stream processing among different

edge nodes introduce new challenges to IoMT application development and deployment. A traditional way to deal with these challenges is the use of virtualization. However, traditional virtualization methods are intensive resources consumers, and the EC environment is resource-constrained. Thus a new approach to lightweight virtualization called *containerization* has been adopted. The lightweight virtualization technology is usually based on a group of processes that still appears to have its dedicated system, but it is running in an isolated environment. This group is typically named a "container". All containers run on top of the same kernel, reducing the resource consumption in the host machine that runs the containers.

2.3 Containers in Edge Computing

Edge computing is a new trend, and in combination with the cloud, it can improve QoS for latency-sensitive applications [8]. On the other hand, develop and manage an IoMT application that is executed in the edge is not an easy task because there is a plethora of edge node device types. An edge node is the computational unit in an EC environment. Each edge node can vary in memory, CPU / GPU and storage capabilities, besides architecture, operating system and many other individual characteristics. Some types of edge nodes are:

- **Single-board Computer (SBC):** It is a complete computer, with limited resources [53] that is built over a single electronic board. It contains components like microprocessor, input/output, memory and sometimes storage. An SBC typically provides fanless, low-power computing and a low profile architecture.
- **IoT Gateway:** It serves as the connectivity gateway between IoT devices and the cloud, allowing the data collected by IoT devices to be uploaded to the cloud [81]. Typically it is a single machine, and it is placed near the ambient where the sensors are deployed.
- **Micro Datacenter:** It is a highly virtualized platform, which provides computation, storage, and networking services between IoT end nodes and clouds [1]. Typically it is composed of some machines, and it is placed inside some local telecommunication station, like an Internet Exchange Point (IX).

One strategy to reduce the complexity of hardware and software heterogeneity is using virtualization [42]. It is possible to categorize virtualization strategies in heavyweight and

lightweight ones. In a heavyweight virtualization environment [42] a *hypervisor* system is typically used to emulate virtual hardware on top of physical hardware and, on top of that virtual structures, an OS is installed and end-user application can be executed. In lightweight virtualization, processes isolation is usually implemented at the OS level, thus avoiding the virtualization of hardware and drivers [54].

The idea of using virtualization in EC is not new. In 2009, the paper [75] studied the possibility of using heavyweight virtualization in EC. Two things negatively impacted the results reported in the paper. The first one was that the EC nodes were not powerful enough to run the software that they tested (Gimp). The second one was that the *hypervisor* used was VirtualBox, and it is a huge resource consumer. The advances in EC node's capabilities and the development of lightweight virtualization have recently enabled the adoption of virtualization in EC.

The adoption of container technology, which is considered lightweight virtualization, is overgrowing. The Gartner Company predicts that, by 2023, 70% of organizations will be running three or more containerized applications in production². It has been adopted because of advantages of application development, deployment and management.

One management aspect was studied in [2], was that as the container approach skips the operation system boot phase, it speedups between 1x (no improvement) and 60x the time for the service to become available, depending on the container type.

Another study was presented in [18]. They analyzed the overhead introduced by container virtualization when multiple concurrent containerized micro-services are executed in parallel within the same edge node device in order to optimize both virtual sensing and actuating resources. The experiments show that the introduced overhead is acceptable, considering the apparent advantages brought by the adoption of container virtualization in terms of resource partitioning.

The studies mentioned previously show that the adoption of the containerization approach in EC can make it a viable solution to host processes that perform actions like video cutting, speech detection or pattern recognition over multimedia streams. Those processes can be encapsulated in virtual entities such as virtual sensors. In the next section, we discuss the virtual sensor strategy.

²<https://www.stackrox.com/post/2020/03/6-container-adoption-trends-of-2020/>

2.4 Virtual Sensors

The idea of creating virtual sensors is not recent. Table 2.3 presents some definitions about what the literature defines as a virtual sensor. The adoption of virtual sensors is emerging again as a viable solution to solve problems related to IoMT environments.

Table 2.3: Definition of Virtual Sensor

Author	Definition
Gat et al. [31] (1990)	Virtual sensor is a mathematical function defined over the values of the physical sensors, and it may require resource scheduling.
Hardy et al. [33] (1999)	Virtual sensor is an algorithm pre-defined and fixed, and it abstracts general purpose sensing functionalities which are useful in their own right and may also be used as building blocks for higher and more complex sensing abstractions.
Kabadayi et al. [38] (2006)	Virtual sensor is a software that provides indirect measurements of abstract conditions combining sensed data from a group of heterogeneous physical sensors.
Madria et al.[49] (2014)	A virtual sensor is an emulation of a physical sensor that obtains its data from underlying physical sensors, providing a customized view to users using distribution and location transparency.

Table 2.3 presents studies that points reasons for the adoption of virtual sensors. Some of them are:

- **Physical sensors heterogeneity:** in the same physical environment, sensors from different vendors may coexist. It means that applications need to hide away the idiosyncrasies, and variations of data that are collected and processed. The heterogeneity also increases the complexity to build and maintain complex systems that gather data from different physical sensors. The virtual sensor approach promotes decoupling of lower-level interfacing;
- **Reuse the physical sensor to increase the ROI:** A virtual sensor can be used to produce new services using the same infrastructure that already exists in the organization;
- **Value Added Information:** The data provided by a virtual sensor can be enriched with data that does not is generated directly by the physical sensor;
- **Increase Fault-tolerance:** a monitored environment typically has more than one single device providing the same data. In a factory, multiple devices measure the

same phenomenon, for example, a chamber pressure. A virtual sensor can aggregate all this data and deliver to the IoT application only one data. In this scenario, if some of the physical devices fail, the IoT application will continue to work.

But, despite the advantages of virtual sensors, there are some issues in the adoption of a virtual sensor approach. Some of them are:

- **Time constraints:** typically dedicated hardware is faster than emulation. Even if the host has the resources for running the virtual entity, it is essential to highlight that the resources are shared among the process running in the host. For the case of multimedia applications, they usually demand a GPU, and many edge devices do not have this resource.
- **Communication:** virtual sensors usually do not run in the same physical device that collects the data. Thus, the data needs to travel from the physical device to the host machine that runs the virtual entity. The network bandwidth must be compatible with the data volume produced by the device. Multimedia devices produce a massive data flow, and this is a challenge to virtualize IoMT devices.

The adoption of virtual sensors has positive and negative aspects as we mentioned. This strategy can be adopted in the IoMT environment too. In this scenario, a virtual sensor that emulates a multimedia device is called Virtual Multimedia Sensor (VMS). A VMS can be used in many situations to feed IoMT application with multimedia streams. This approach can also improve many application aspects, for example, security and privacy. In [77], the authors discuss significant issues in many web surveillance cameras, so hiding information about physical devices can improve the aspects mentioned previously.

The combination of EC and the VMS approach enables the rising of a new plethora of IoMT applications. It can reduce the total latency due to use of EC and overcome the edge heterogeneity with a container lightweight virtualization approach. Another advantage of the use of containers is that the development of the VMS can be done using the technology that best fits with the VMS and IoMT application requirements.

In this chapter, we presented the background for understanding this work. We defined IoMT, edge computing, virtual sensors, and the characteristics that enable the use of containers in edge environments. In Chapter 3, related work will be discussed.

Chapter 3

Related Work

The great heterogeneity of devices and multimedia vendors brings an extra effort to develop, deploy and manage IoT multimedia applications (IoMT applications). On the other hand, the adoption of an architecture/framework may reduce that effort significantly.

As previously commented, in the Cloud of Things (CoT) paradigm [15], the cloud acts as an intermediate layer between physical data producer and applications. Traditional cloud computing is strongly based on resource virtualization, to make it easier for multiple users to share a single physical resource or application. Cloud virtualization also helps managing the workload, transforming traditional computing and making it more scalable, cost-effective and efficient.

In the CoT, the strategy of virtualization has been adopted to reduce management complexity generated by sensor heterogeneity, since it helps to abstract specific features of physical sensors, thus promoting interoperability among devices. Virtualization also contributes to decrease the sensor infrastructure maintenance costs and increase the service availability.

In this chapter, we describe and compare studies directly related to our proposal. We analyze the following aspects: (i) traditional and multimedia sensor virtualization and; (ii) architectures for sensor virtualization.

3.1 Traditional and Multimedia Sensor Virtualization

In [20], the authors propose programming abstraction simplifying the development of decentralized WSN applications using virtual sensors. In this novel, the data acquired by multiple sensors can be collected, processed and aggregated, and then perceived as

the reading of a single virtual sensor. Another work that follows the same approach is [33]. The authors define a structure/framework that must be followed by the developers for designing and constructing virtual sensors, avoiding low-level implementation details. The adoption of a specific language can reduce the effectiveness of the approach once the developers must learn the new language and remake all work done in other languages. Our proposed architecture, on the other hand, allows the developers to choose the programming language that better fits the requirements of the virtual sensor being built.

An automatic allocation method for virtual sensors in the cloud was proposed in [46]. The authors argue that the provisioning process is a crucial task since it is responsible for selecting physical sensors that will be used to create virtual sensors. The approach includes two algorithms: an adaptive clustering algorithm based on similarity where the physical sensor nodes are clustered according to their measurements similarity, and only a subset of nodes from each cluster is selected to compose a virtual sensor, and ant colony optimization for sensor selection based on similarity. Results from experiments show that by using the proposed approach, the overall energy consumption of sensor nodes was reduced, prolonging the lifetime of the sensor cloud. However, they do not investigate multimedia sensors and do not evaluate the relationship between energy and bandwidth in edge nodes.

In [71], the authors exploit computing capabilities of the Wireless Sensor and Actuator Network (WSAN). They claim that, by running localized and collaborative algorithms, inside the nodes, leverage the cloud of sensors paradigm to make the best use of the cloud and physical WSAN environments. The proposed strategy also supports one-to-one, one-to-many, and many-to-one associations between real-world IoT devices and virtual devices in the cloud. However, the authors defined the cloud to host virtual sensors and actuators. For latency-sensitive applications, this approach is not ideal. In our approach, we also use the idea of the multiple types of relationships between devices and virtual sensors but we host virtual multimedia sensors in the edge nodes, exploiting the low-latency characteristic of this environment.

A proposal that adopts a lightweight virtualization model that distributes data processing in the edge tier, encompassing the specific requirements of IoT applications was presented in [5]. The distributed resource management process provides each edge node with decision-making capabilities, engaging neighboring edge nodes to allocate or provision on-demand virtual nodes. However, the architecture does not support applications requesting different types of data in a single request, and the data type cannot be a

multimedia stream.

In [56], the authors propose an architecture that automates the provisioning of applications with components spanning cloud and fog nodes. It also enables discover of existing cloud and fog nodes and generates the application graph. In that architecture, each application is defined as a Virtual Node Function Forwarding Graph (VNF-FG). In their proposal, an application is a set of VNFs chained in a specific order to solve a particular problem. The architecture provides a set of tools that enable the deployment of complex applications. However, the bind between the VNF is defined in initialization time. That behavior implies that, for restarting a VNF, all applications must be restarted. In our approach, we provide a dynamic binding between virtual sensors and virtual devices; thus and we can change in execution time the components of the stream processing pipeline.

Smart transportation safety envisions improving public safety through a significant paradigm shift for police authority responses on crimes toward a pro-active one. The framework proposed in [60] was tailored to deal with event detection over multimedia stream, specifically for smart transportation safety. It is a 3-tier architecture named Fog-FISVER. The first tier refers to an in-vehicle fog node featuring mechanisms for fetching local sensor data and implementing local-level crime analytics. The second tier is running at the fog computing infrastructure, performing high-performance crime event analytics. The third tier refers to mobile applications running on mobile devices at incident response police vehicles, which in turn report a crime to the end-user (the police agent) within a short latency. The proposed approach allows only one edge node in the monitored environment, therefore there is no need to implement a resource allocation method. Typically edge environments have many edge nodes in the same place, thus, that approach can be applied in a small set of scenarios. In our proposal, many edge nodes can be used to host virtual sensors.

In [61], the authors present a container-based edge node approach for enabling edge nodes to run applications (virtual sensors) in containers, as well as the orchestration of container deployment. Their architecture includes a development layer to provision and manage applications over edge nodes. An important aspect is that the edge environment is a cluster, based on a homogeneous set of edge nodes. However, the typical case is that edge nodes are heterogeneous devices. Because of that, proposing a homogeneous deployment scenario limits that approach adoption. Our approach does not impose limits to the type of edge nodes.

An architecture to deal with edge node functions placement in roadside infrastructure

was proposed in [21]. They presented an orchestration framework that enables edge-cloud collaborative computing for road context assessment. The framework can create on-demand task execution pipelines spanning multiple, potentially resource-constrained edge nodes with the smart IoT infrastructure support. They use the term Edge Function (EF) to define an atomic function that embeds a small piece of the application logic that can be executed standalone. That EF can be chained together to execute a complete task, for example, accident detection. The description of the application is based on a directed acyclic graph, the same way as in [27].

Node-RED¹ is programming environment that enable the development for event-driven applications with a minimum amount of code writing. It can be used for creating virtual sensor environments. In [19], the authors used Node-RED to define an architecture for smart home application virtualization. The aim of their approach is to allow development of more complex applications that can be used in different sensor networks. The virtual sensors were used for identification of temporal relationship patterns in user behavior using recurrent neural networks. As a limitation imposed by Node-RED, the virtual sensor cannot manipulate multimedia streams like audio and video, and the virtual sensors can only be executed inside the Node-RED platform.

The authors of [38] presented a method for data aggregation/processing in sensor networks. They argued that the separation of the specification of the sensing task from the sensing behavior allows a programmer to describe the behavior of a virtual sensor, without having to specify the details of how it should be built. Role separation is important in IoT once the environment is heterogeneous and dynamic. The idea of task segmentation is also used in our approach, where virtual multimedia sensors can receive streams from a variety of physical multimedia devices via virtual devices that encapsulate the complexity of handling physical devices.

In [70], the authors propose a framework that provides resources for monitoring, collecting, processing, storing data, and interfaces for providing data to other applications and/or systems in WSN environments. OSIRIS uses a set of abstractions to offer flexibility for the creation of various monitoring systems and to decouple network physical sensors from data consuming applications. The authors validated OSIRIS building a thermal monitoring system for datacenters. Their approach does not handle multimedia devices and the authors do not formally defined where VMS will be deployed. However, by analyzing the examples provided in the paper, we can infer that it is done in one edge

¹<https://nodered.org/>

node.

In [87], the authors propose a novel video analytics architecture based on edge computing, where a large amount of video data does not need to be uploaded through the core network and processed in the cloud. Instead, computing takes place partially or entirely in edge nodes. It reduces the possibility of link congestion and failures, increases the performance and shortens the response time. The main focus of the authors was presenting an indoor video detection technique and not the architecture itself, thus components were poorly described. Nevertheless, the results presented show that the edge adoption to process multimedia streams brings benefits in terms of latency and bandwidth consumption.

The comparison between the features proposed in those studies is shown in Table 3.1. Our proposal, named **V-PRISM** is included in the last row of Table 3.1. The ✓ indicates that the paper covers the topic analyzed in that column. Before presenting the table, we will describe what each column of this table means:

- **Virtual Sensor:** we analyze if the proposed approach deals with virtual sensors;
- **Virtual Actuator:** we analyze if the proposed approach deals with virtual actuators;
- **Cloud Deployment:** we analyze if the proposed approach enables the deployment of virtual sensors or actuators in the cloud nodes;
- **Edge Deployment:** we analyze if the proposed approach enables the deployment of virtual sensors or actuators in the edge nodes;
- **Device Deployment:** we analyze if the proposed approach enables the deployment of the virtual sensors/actuators in the device;
- **Resource Allocation:** we analyze if the proposed approach provides any resource allocation method/strategy/algorithm;
- **Data Sharing:** we analyze if the proposed approach provides any method to share data processed by the virtual sensor between the components;
- **Multimedia Stream:** we analyze if the proposed approach can handle multimedia streams;
- **Dynamic Multimedia Bind:** we analyze if the proposed approach can handle dynamic binding between the device (physical or virtual) and the virtual sensor.

Table 3.1: Comparison between virtual things approach

Papers	Virtual Sensor	Virtual Actuator	Cloud Deployment	Edge Deployment	Device Deployment	Resource Allocation	Data Sharing	Multimedia Stream	Dynamic Multimedia Bind
Ciciriello et al.[20]	✓	✓	-	-	✓	-	-	-	-
Lemos et al. [46]	✓	-	✓	-	-	✓	✓	-	-
Santos et al. [71]	✓	✓	✓	-	-	-	✓	-	-
Alves et al. [5]	✓	✓	✓	✓	-	✓	✓	-	-
Mouradian et al. [56]	✓	-	✓	✓	-	✓	✓	✓	-
Neto et al. [60]	✓	-	-	✓	-	-	-	✓	-
Pahl et al. [61]	✓	-	-	✓	-	-	-	-	-
Donassolo et al. [27]	✓	-	-	✓	-	✓	✓	-	-
Cozzolino et al. [21]	✓	-	-	✓	-	✓	-	✓	-
Hardy et al. [33]	✓	✓	-	-	✓	-	-	-	-
Chenaru et al. [19]	✓	-	-	✓	-	-	✓	✓	✓
Kabadayi et al. [38]	✓	-	-	-	✓	-	-	-	-
Santos et al. [70]	✓	-	-	✓	-	-	✓	-	-
Xie et al. [87]	✓	-	-	✓	-	-	-	✓	-
V-PRISM	✓	-	-	✓	-	✓	✓	✓	✓

In this section, we discussed approaches to handle virtual sensors. In the next section, we will analyze sensor virtualization architectures. Some studies presented in this section will also be discussed in the next section from an architectural perspective.

3.2 Sensor Virtualization Architectures

In order to select related work about Sensor Virtualization Architectures (SVA), we adopted the software architecture definition provided by [22], where an architecture consists of (a) a partitioning strategy and (b) a coordination strategy. The partitioning strategy leads to dividing the entire system in discrete, non-overlapping parts or components. The coordination strategy leads to explicitly defined interfaces between those parts.

Extending the software architecture definition provided by [22], we define an SVA as a set of software components and patterns that enable the development and management of virtual sensors (VS) that are source of data to applications. In this section, we show some VSA already proposed to manage virtual sensor environments. We did not restrict our search for architectures that manage multimedia virtual sensor because they are in a limited number.

There are many approaches to deal with virtual sensors (VS). In [43], the authors propose a scalable virtual sensor framework that supports building a logical dataflow (LDF) by visualizing physical sensors or custom virtual sensors. A web-based virtual sensor editor (VSE) was implemented on the top of the framework to simplify the creation and configuration of the LDF. Each virtual sensor is composed of a set of linked mathematical functions provided by the MathJS library². Each VS is a process running inside a cluster server. There is only one cluster, and it is elastic to adapt to the unexpected change in terms of the number of concurrent users, this adaption is done by adding or removing a node to/from the cluster. The authors do not discuss the strategy adopted for VS allocation. Besides that, as the VS is composed only by purely predefined mathematical functions, the capability of new VS Types are limited to scalar data.

In [27], the authors propose FITOR, an orchestration system for IoT applications in the fog environment. This solution builds a realistic fog environment while offering efficient orchestration mechanisms. FITOR relies on an optimized fog service provisioning strategy that aims at minimizing the provisioning cost of IoT applications while meeting their requirements. The architectural model assumes that the edge nodes must be a Docker host machine, and each running container must have a set of services to collect data from the container. The strategy adopted to collect statistic data was not optimized, increasing the resources used in an already limited resource environment. In addition, the approach was not specifically tailored to deal with multimedia edge node environments. Besides that, they did not specify which resource allocation method was used.

A new sensor cloud architecture for IoT based on fog computing was proposed in [85]. The architecture preprocesses raw sensor data on fog node and provides temporary storage of the results, controls and manages diverse types of sensors in IoT devices through the virtualization of physical sensors, and ultimately provides dynamic, on-demand, elastic and standardized Sensing-as-a-Service to end-users. However, the proposed architecture does not allow the inclusion of multimedia sensors or the remote management of the VS.

²<https://mathjs.org/>

Two similar approaches were proposed in [93] and [37]. Both studies propose a web-based authoring tool for creating virtual sensors. In these tools, a VS is created by aggregating the data provided by physical sensors, and, the data-flow is chained to generate VS for complex event detection. The VS Type was defined using JSON and Javascript. When the execution of a VS starts, the VS Type definitions (JSON and Javascript files) are loaded from a database. After that, the loaded script is evaluated by a Javascript Engine. The Javascript Engine runs in the server where the architecture is executed. It implies that there is only one node for running the VSs. Besides that, the architecture defined that the devices must generate scalar data.

Another relevant work was discussed in [70]. The authors proposed a framework to virtualize sensors of multiple types. Besides that, they developed a communication protocol called OMCP (OSIRIS Module Communication Protocol), which allows that external applications manage virtual sensors. It is important to highlight that that study was the only framework analyzed that formally discussed and implemented such a type of interaction. Unfortunately, the authors did not discuss aspects of virtual sensor deployment, an essential issue for sensor virtualization.

Besides academic work, many industry initiatives use the virtual sensor approach to deal with IoT devices. In [23], the authors compare the *Amazon AWS Greengrass* and *Azure IoT Edge*. Both are commercial platforms to virtualize sensors. They enable the manipulation of scalar data and some types of multimedia streams like audio and static images. Both platforms distribute virtual sensor in the edge and cloud. The authors stated that the performances of Greengrass and Azure Edge are similar in many test cases, but they found that Azure Edge exhibits higher end-to-end latency due to the batch-based processing adopted in multimedia scenario. That higher latency scenario represents a problem for latency-sensitive applications. Besides that, those platforms have only commercial licenses, and they are proprietary software.

Another organization that is heavily working on the standardization of edge computing architectures is the European Telecommunications Standards Institute (ETSI). ETSI defines standards to deal with telecommunications, broadcasting and other electronic communication networks and services. In [29], they provided a framework and reference architecture for Multi-access Edge Computing (MEC). The document describes a MEC system that enables MEC applications to run efficiently and seamlessly in a multi-access network. This is a general guide to develop architectures to a more specific domain. Our architecture follows some of the standards described in [29]. In particular, the idea of

an orchestrator, as we can see in Figure 3.1, that shows a complete view about physical devices and software components of the environment and defines in which edge node the application must be started or when the application must to be relocated.

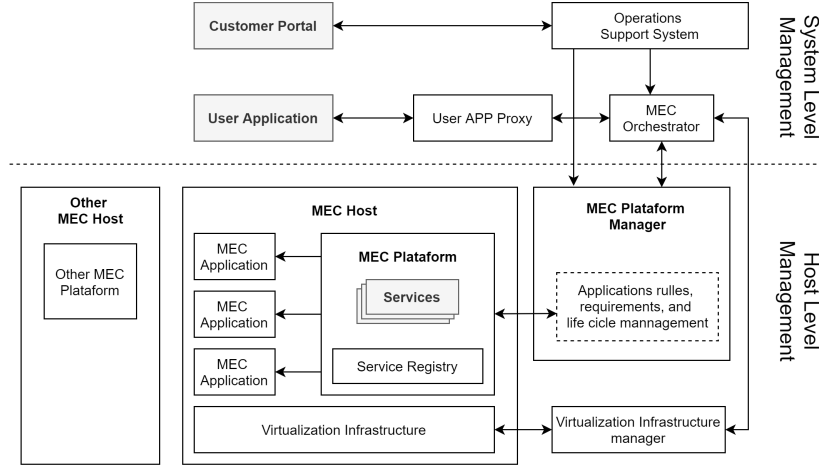


Figure 3.1: MEC system reference architecture [29]

A relevant feature depicted in Figure 3.1 is that the infrastructure can handle many MEC Hosts (edge nodes) and, each host can execute different MEC applications types (virtual multimedia sensors). In this way, different application providers could deliver their applications to be executed inside the MEC infrastructure. Some factors that enable this level of interoperability are the application virtualization and the existence of a Service Registry component.

In contrast with the previous architectures, our proposed architecture, V-PRISM, adopts a light virtualization approach, allowing the execution of multiple virtual multimedia sensors in multiples heterogeneous edge nodes. External applications can manage VMS through an API, the VMS allocation can be automated by multiple resource allocation algorithms, and these algorithms are selected by the IoT infrastructure owner. There is a web interface to manage V-PRISM components, and programmers can develop and deploy new VS types following the guidelines provided by our architecture.

The comparison between some VSA is depicted in Table 3.2. The last row of the table represents our proposal architecture. Each column of the table means:

- **Node Management:** We analyze if the SVA allows node management. A node is a computational device where one or more VS will be executed.
- **VS Type Setup:** We analyze if the SVA is capable of define which kind of VS Type

can be executed in each node of the infrastructure (either edge or cloud), meaning that a single node can run different VS Types.

- **API for VS Management:** We analyze if the SVA allows remote applications to manage a VS. Typically this is done via a REST API.
- **Resource Allocation:** We analyze if the SVA provides any method/mechanism for dynamic resource allocation.
- **Management Interface:** We analyze if the SVA has any interface (graphic or via command line) for managing its components.
- **Light Virtualization:** We analyze if the SVA uses any light virtualization method for virtualizing its components.
- **Component Deployment:** We analyze where a VS is deployed. The deployment can be done in the cloud (C), edge (E) or a hybrid approach (H).

Table 3.2: Comparison between Sensor Virtualization Architectures

Papers	Node Management	VS Type Setup	API for VS Creation	Resource Allocation	Management Interface	Light Virtualization	Component Deployment	Multimedia Stream
Kim-Hung et al. [43]	-	✓	-	-	✓	-	E	-
Donassolo et al. [27]	✓	✓	-	✓	✓	✓	E	-
Wei et al. [85]	✓	✓	-	✓	✓	✓	E	-
Zhang et al. [93]	-	✓	-	-	✓	-	C	-
Jeong et al. [37]	-	✓	-	-	✓	-	C	-
Santos et al. [70]	-	✓	✓	-	-	-	E	-
ETSI et al. [29]	✓	✓	✓	✓	✓	✓	E	-
V-PRISM	✓	✓	✓	✓	✓	✓	E	✓

In this chapter, we presented existing work and projects that were developed in fields related to our proposal. We studied cases when the virtual sensor paradigm was adopted. Besides that, architectures for virtualizing multimedia sensors were compared with our approach. In the next chapter, we will discuss the concepts and components of V-PRISM.

Chapter 4

V-PRISM Proposal

This chapter presents V-PRISM, our proposed architecture to virtualize and manage virtual multimedia sensors (VMS). Initially, we detail the three-tier architecture considered in our work in Section 4.1. In the following, we present a categorization for VMSs in Section 4.2. V-PRISM logic components are described in Section 4.3, and the V-PRISM operation is detailed in Section 4.4.

4.1 The three-tier architecture

In this section, V-PRISM architecture is presented. According to Roy Fielding in his REST dissertation, “A software architecture is an abstraction of the run-time elements of a software system during some phase of its operation.” Following this vision, the elements that make up V-PRISM will be described in the next subsections, from a logical point of view. That is, the structure (set of elements) and the behavior (operation) of these elements when they interact to perform the expected functions will be detailed. These elements are software components. However, such components must be deployed in computational nodes that build up an IoMT system. For the design of V-PRISM, we consider that IoTM systems follow a three-tier architecture encompassing: (i) the cloud, (ii) edge, and (iii) things tiers, as proposed in [48]. Figure 4.1 shows an overview depicting how each entity considered in our proposal is deployed in each tier. The main goals and composition of each tier are:

- The **Cloud** is the top level of the architecture. The elements that integrate this tier are machines hosted in big data centers, with powerful CPU, memory, storage, and network capabilities. Usually, IoMT applications are hosted in this tier. The cloud

can provide almost unlimited resources for applications, but the processing nodes in this tier are far from IoMT devices. This distance implies an increase of the total latency.

- The **Edge** tier is the intermediate layer between the things and the cloud. The elements that make up this tier are resource constrained machines hosted in small data centers (cloudlets), or single-board computers like Raspberry deployed directly inside the monitored environment. The resources at the edge are limited, in contrast to the cloud, although they are closer to the physical devices. Thus this tier can provide services with lower latency in comparison to the cloud.
- The **Thing** tier is where multimedia devices are hosted. The things are typically heterogeneous and produce a massive amount of data. They usually have limited resources available to process and store the generated data. The goal of this tier is to produce data streams. Moreover, devices in this tier may contain, in addition to sensors, some processing capacity, although limited. Therefore, simple processing tasks can be performed in this tier.

V-PRISM architecture encompasses the following types of elements: Virtual Multimedia Sensors (VMS), Virtual Devices (VD), and V-PRISM Manager components. All these elements are deployed in edge nodes, as presented in Figure 4.1. A VD receives/collects multimedia streams from a physical device. After that, one or more VMSs will request the multimedia stream to perform some operation over the data. Finally, the VMS output will be sent to other VMSs or IoMT applications. We consider IoMT applications as any type of external V-PRISM entity that consumes the output of a VMS. Typically IoMT applications are hosted in the cloud, but they can also be hosted in the edge and things tier. V-PRISM can handle all these deployment configurations for IoMT applications (at the cloud, edge or at the things tiers).

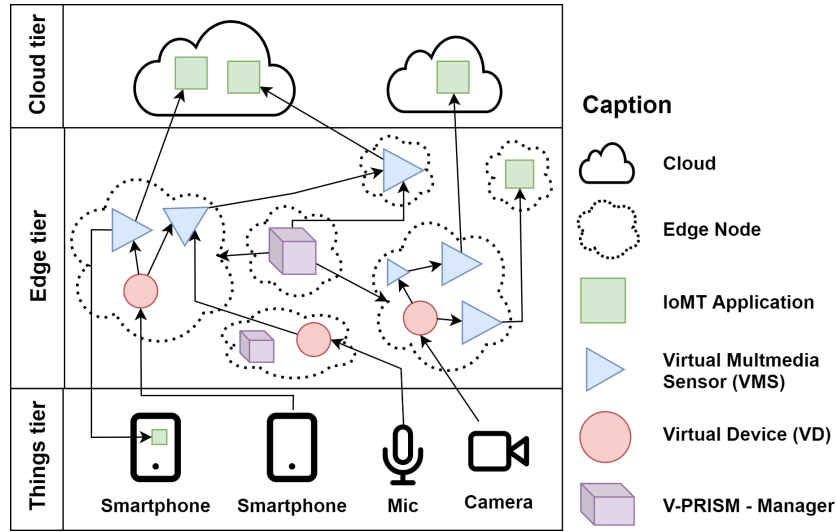


Figure 4.1: V-PRISM three-tier architecture overview

Figure 4.2 depicts an example of VDs and VMSs in a smart building application. The left side of the figure shows the monitored places in an organization called *Corporate X*. The right side of the picture presents the virtual multimedia sensors. The monitored places in this example are *Room A*, *Hall B* and *Backyard*. Each named place represents a physical space in a building where the *Corporate X* organization operates. Physical devices are deployed in each of these places. For example, in Room A there is a Full HD Camera & Microphone.

Each physical device can be a source of multimedia data to many VMSs. In Figure 4.2, the data generated by the *Full HD Camera & Microphone* is providing multimedia streams to three different VMSs. This approach can maximize the usage of the organization resources since an already deployed device can provide multimedia streams to new opportunistic services, as presented in [47].

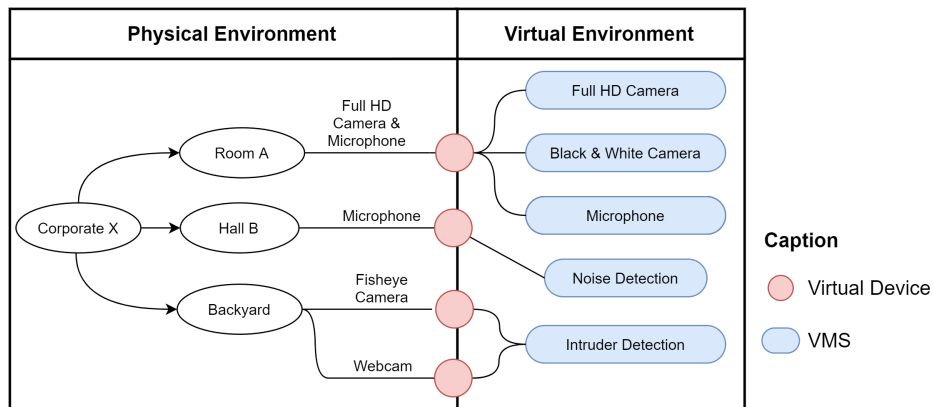


Figure 4.2: Description of an environment where V-PRISM can be deployed

The three-tier architecture is an approach designed to explore the integration of things/edge/cloud tier. In V-PRISM we adopt this strategy once each tier have its own responsibilities. In the next section, we will present a proposal of VMS categorization.

4.2 VMS Categories

To facilitate understanding and organizing the different types of VMS, we propose a categorization according to the multimedia stream abstraction provided by the VMS type. We coined two new terms *VMS types* and *VD Types*. For better comprehension, an analogy with Object Orientation is that a VMS type is a class and a VMS is an object. A VMS is the instantiation of a VMS Type. Besides helping to organize the various possible types of VMS that can be built on V-PRISM, these categories also potentially serve for the following two purposes:

- (a) the proposed categories group the VMS by the provided functionalities (from the simplest to the most complex ones), so they also help to guide the amount of resources that will be needed to instantiate each type. More complex nodes will require more infrastructure resources for their deployment and execution. Therefore, a minimum amount of resources required for each VMS category can be stipulated and a resource allocation algorithm can use this information for its decision making. Only edge nodes able to provide the minimum resources of a given category would be candidates for the implementation of nodes of the respective type;
- (b) provide templates for each VMS Type, which facilitates the work of the developers.

To the best of our knowledge, no previous work proposed a taxonomy tailored for categorizing VMS. Figure 4.3 depicts the proposed hierarchical categorization.

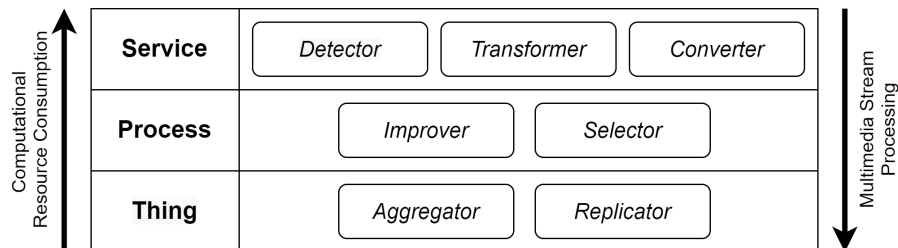


Figure 4.3: VMS Hierarchical Categorization

The categorization presented in Figure 4.3 was inspired on the information classification proposed in [16]. The author coined three concepts to define information abstraction levels. The first one is that information is a **thing** (information-as-thing) when the information is only data. The second one is **knowledge** (Information-as-knowledge) when an entity makes some reasoning over the information, and the third one is **process** (Information-as-process) when the information consumed by some entity promotes some change in the entity that receives the information. Therefore, inspired by this work, our proposed organization is also composed of three main categories defined as follows. VMS as a **Thing**, when the VMS only forwards the data collected by the device. VMS as a **Process**, when the VMS, besides forwarding the data, adds new features to the virtual device. VMS as a **Service**, when the VMS provides a high-level abstraction feature over the multimedia stream.

As depicted in Figure 4.3, the computational complexity and resource consumption tend to grow when the abstraction level of the VMS category rises. On the other hand, the level of multimedia stream processing reduces as the category level decreases. It indicates that VMSs defined in different categories have different requirements. Thus this hierarchical categorization can be used as a parameter to resource allocation algorithms once a VMS cannot run inside every edge node available.

In order to produce a comprehensive organization, we studied the already proposed categorizations for virtual sensors presented in [49], [14] [70], [43] and [32]. Therefore, we propose the following second-level categories: replicator, improver, converter, selector, aggregator, detector and transformer. It is important to note that this is not an exhaustive categorization, since with the advancements in IoT environments, new devices can enable innovative VMS types, forcing the creation of new categories to cover them.

Although this proposal of categorization, depicted in Figure 4.3, is one of the contributions of this dissertation, to explore the benefits of its use is future work. In addition, the relationship between VMS categories and resource allocation algorithms will also be addressed in future work. In the next sections, we will describe each VMS category and present some use cases to illustrate them.

4.2.1 Replicator

In a *Replicator* VMS, all characteristics and behavior of a physical multimedia device are replicated in a virtual twin entity. *Replicators* are the elementary type of VMS. A VMS in this category does not make any change in the original multimedia stream.

Replicator VMSs are mainly used to reduce resource consumption in the physical device and in the IoT network. A VMS in this category also consumes few resources in the edge node because they are only data forwarders. As a consequence, they can be executed in low-capacity edge nodes enabling its adoption in legacy deployed IoT environment.

One use case of *Replicator* VMS is in surveillance systems. Typically the video stream produced by the surveillance camera is collected and delivered to one or more IoMT applications, as we can see in Figure 4.4(a). Using a *Replicator* VMS, Figure 4.4(b), the data is collected from the physical camera only once, and the stream is delivered to many different IoMT applications by the VMS.

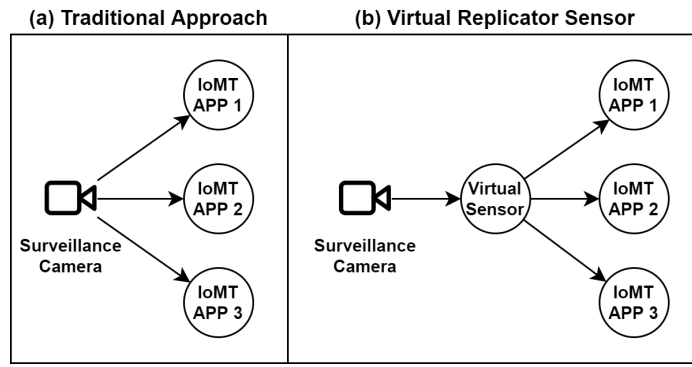


Figure 4.4: Traditional and virtual sensor approach in surveillance systems

4.2.2 Improver

Many IoMT devices have restricted capabilities. They sometimes cannot perform QoS tasks or providing reliable data security. An *Improver* VMS can provide to IoMT applications features that do not exist in the physical multimedia device.

A frequent situation in IoT environments is the existence of devices deployed without a firmware update, which allows security threats. It happens because typically an IoMT system can run for years, and during this time, vendors can stop making improvements in the device's firmware. In these situations, an *Improver* VMS can be used to increase lifetime for legacy multimedia devices already deployed in the environment.

Security is one of the most concerning issues in the IoT world nowadays. Many papers have been published talking about techniques and strategies to improve IoT security [34]. In particular, IP cameras have significant potential for security issues, particularly when they get old. This scenario is represented in Figure 4.5(a). As vendors stop producing new drivers to IP cameras, they became vulnerable to hacking attacks. One strategy to

continue using the vulnerable device is placing it behind a VMS, and exposing the VMS IP and not the camera IP, as in Figure 4.5(b).

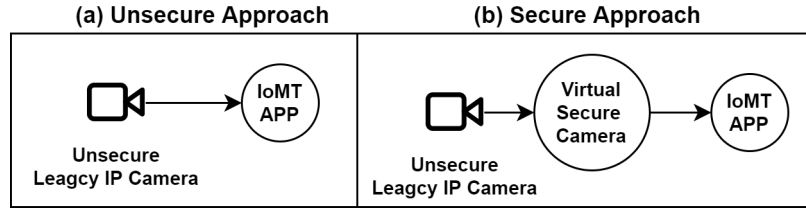


Figure 4.5: Example of security threat in legacy IoT environment

4.2.3 Converter

One of the most prominent characteristics of IoMT is device heterogeneity. It is common that in the same environment, devices produced by different vendors can be used together to perform some task. In many situations, the type of multimedia stream produced by some multimedia sensor is incompatible with the IoMT application requirements.

A *Converter* VMS can be used when the multimedia stream produced by the device needs to be changed to meet the IoMT application requirements. For example, if an old microphone can produce only a WAV stream, but the IoMT application requires an OGG audio stream, as we can see in Figure 4.6, a *Converter* VMS can convert the multimedia stream to a different codec format.

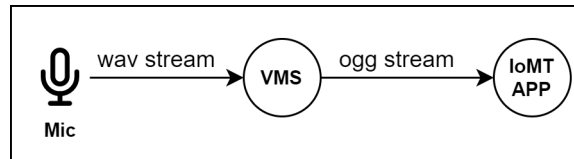


Figure 4.6: VMS converting an audio stream codec format

4.2.4 Selector

The *Selector* VMS category is composed by VMS types that receive multiple multimedia streams as input and then choose only one stream as output. All the multimedia input streams must be of the same type. The multimedia stream is not modified, only forwarded to the IoMT application.

Each *Selector* VMS has at least one selection function. The selection function can use a simple QoS device stream analytics or a complex multimedia pattern recognition. For

pattern recognition, the VMS can implement artificial intelligent algorithms as proposed in [44].

One example of a *Selector* VMS is a *Sound Selector* VMS. Figure 4.7 depicts the scenario where multiples microphones are placed in the same meeting room. The VMS will send to the IoMT application the sound of the microphone with the best quality at a given time.

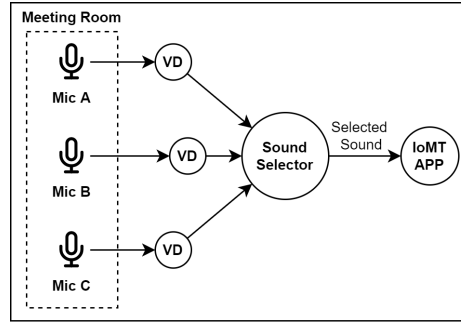


Figure 4.7: VMS sound selector

4.2.5 Aggregator

An *Aggregator* VMS receives multiple multimedia streams and sends a single combination of them to the IoMT application. There are two classes of *Aggregator* VMS. The first one will aggregate multimedia streams of the same type (for example, two or more video streams), and the second one will aggregate multimedia streams of different types (for example, audio streams and video streams).

One example of an *Aggregator* VMS is a video mosaic, depicted in Figure 4.8. A video mosaic application receives video streams from many different cameras and combine all streams in one single output stream. In this scenario the VMS will aggregate all the streams and will send to the application. Another example is a VMS that receives a video stream from one camera and an audio stream from a microphone and combine both streams before delivering it to the IoMT application.

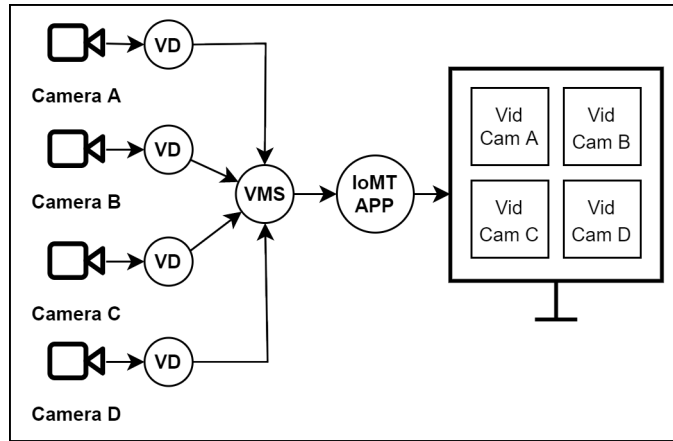


Figure 4.8: *Aggregator* VMS of producing a video mosaic

4.2.6 Detector

The *Detector* VMS category encompasses VMSs that perform event detection in a multimedia stream. Artificial Intelligence techniques can be used in these VMS [44]. It is essential to mention that the VMS runs in an edge node, usually with low capabilities; thus, the algorithm selected to execute the detection needs to be compatible with the resources available in the edge node.

A *Detector* VMS receives a single multimedia stream as input. *Detector* VMS can delivery to IoMT applications a different type of stream. For example, the VMS that receives a video stream can trigger an API endpoint if a monitored event occurs.

It is possible to develop a *Detector* VMS to perform complex event detection. In [3], the authors presented a complex real-time event detection framework for resource-limited multimedia sensor networks. The same type of strategy can be applied to a VMS running inside an edge node.

Figure 4.9 presents a use case of an audio stream as a multimedia source for a *Detector* VMS. If a gunshot noise is detected, a message is sent to the police department. This proactive monitor can be used in smart cities to improve citizens security.

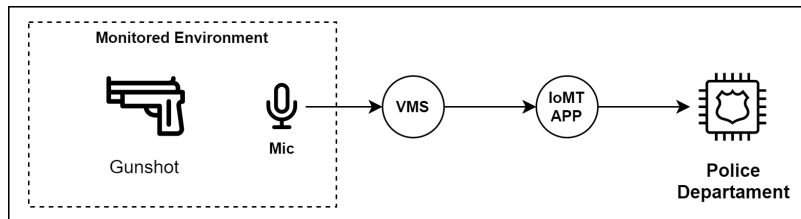


Figure 4.9: Gun shoot detection a *Detector* VMS

4.2.7 Transformer

A *Transformer* VMS changes the nature of the stream. It is used when the multimedia stream needs to be transformed into another type of data. An example of transformation is when video streams must be converted into a file or an audio stream must be converted into text.

A car license plate detection VMS is an example of a *Transformer* VMS. Typically plate detection algorithms receive a video stream and extract the numbers and letters from the plate. The IoMT application receives only the text extracted from the video. This approach can save resources from the network environment once only the recognized string is transferred over the network and not the car plate image.

In the next sections, we present the V-PRISM logic components. The components and interactions between them will be described in detail.

4.3 V-PRISM Logic Components

In recent years, there has been a lot of effort to specify a general-purpose architecture to run and manage applications in the edge [60] [56] [61] [5]. Our architecture follows a general propose meta specifications presented in [29] by ETSI. ETSI is a European Standards Organization (ESO). As a general specification, it leave domain detail definitions to more specific architectures. Because of that, we propose V-PRISM, an architecture explicitly tailored to deal with the challenges of IoMT in edge environments.

The V-PRISM logic components are depicted in Figure 4.10, They are responsible for processing multimedia streams produced by multimedia devices and consumed by IoMT applications. IoMT applications are typically deployed in the cloud, and physical multimedia devices are typically placed in the things tier. Stream processing is performed by the set of virtual multimedia sensors (VMS) provided and hosted by the architecture. The VMS and all V-PRISM components can be deployed in multiples edge nodes, in a distributed fashion.

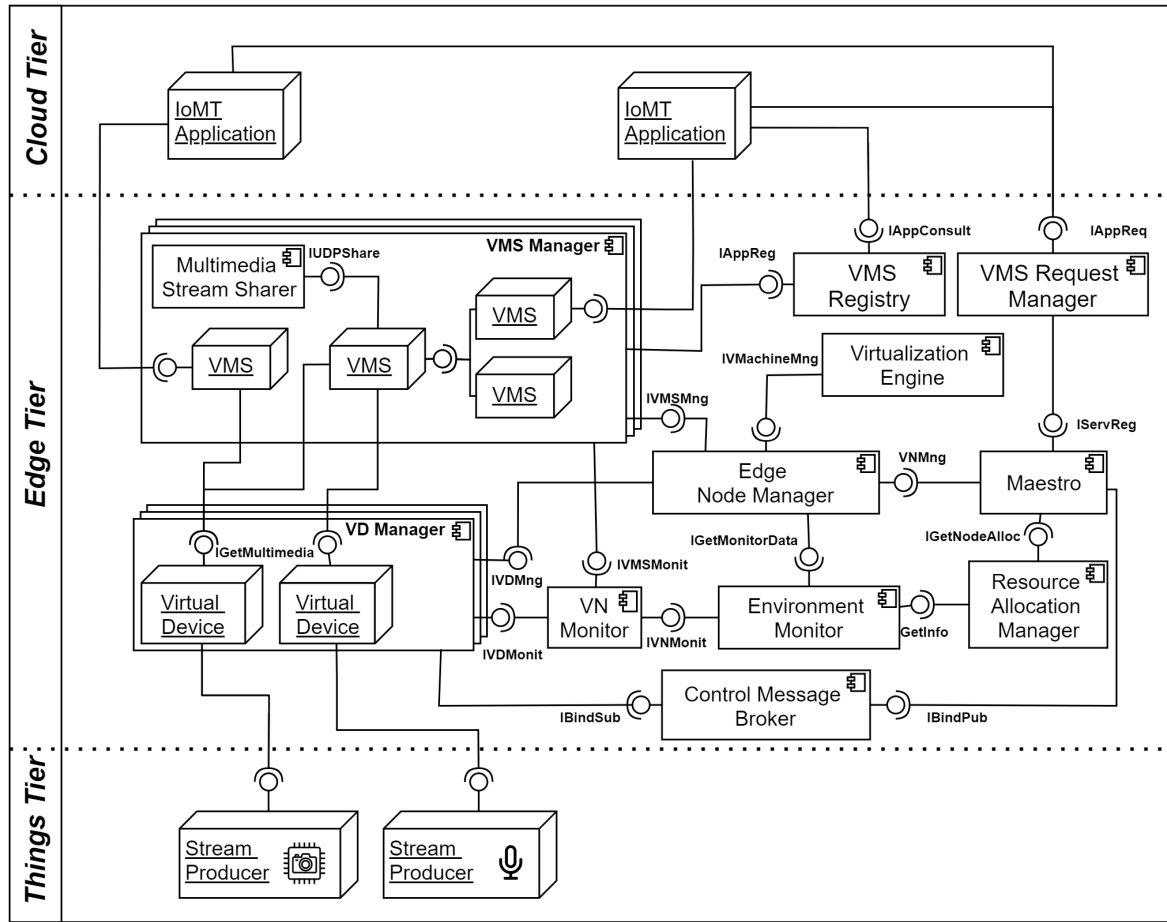


Figure 4.10: V-PRISM Logic Components Architecture

4.3.1 Virtual Multimedia Sensor

A Virtual Multimedia Sensor (VMS) is the architectural component responsible for providing multimedia stream processing. Therefore, a VMS component performs some process in a multimedia stream produced by physical devices (media things). Examples of processing are the conversion of video formats, multimedia feature extraction; aggregation of new characteristics to already existing multimedia streams, to name a few. For each specific type of multimedia stream processing, there will be one VMS Type. We will define VMS Types in the following sections.

The data processed by a VMS can be used by two types of entities. The first one is the IoMT application that requested the corresponding stream. The second one is another VMS. Therefore, the VMSs can be chained, where the output of a given VMS is used as input for the next VMS. Chaining VMSs allows reusing the output of a stream processing while creating complex processing flows to accommodate application requirements.

Each VMS has one or more input ports, connected to a virtual device (VD) or to another VMS, and one output port. It can be noticed in Figure 4.10 that the connection between a VMS and an application is unidirectional. This fact implies that the application does not have a direct communication channel with the VMS. If the application needs to send a control message to a VMS, this must be done using the Service Registry component.

A VMS is defined as a tuple $VMS = \{P, I, S, D, C, SH, E\}$ where $P = \{par_1, \dots, par_n\}$ is a set of configuration parameters $par = \{name : value\}$ that will define how that specific VMS will process the multimedia stream, such as saturation level, noise level, quality of the video, etc.; $I = \{en_1, \dots, en_n\}$ is a set composed of VD or VMS entities providers of input streams. S is the software that performs the respective multimedia stream processing, this value can be the container ID if the software is running inside a Docker environment, the PID if its running on LXC (Linux Containers), or any other type of software identification; D is an $(IP, PORT)$ tuple that defines the first address used to send the output result (it can identify an IoMT application or another VMS); C refers to the category of VMS, as described in Section 4.2, each implementation of V-PRISM can choose the best approach to identify the category, it can be the category name, a pointer reference, or an ID; SH is a boolean value that defines if the processed output stream can be shared with other VMSs or IoMT applications; E is an optional parameter that defines in some cases in which edge node the VMS must be executed.

VMSs are independent of the physical device. It means that, during execution time, the physical device that provides multimedia stream to the virtual device can be changed without any VMS downtime or reconfiguration.

4.3.2 Virtual Device

A Virtual Device (VD) is the architectural component responsible for forwarding a multimedia stream produced by a multimedia device to a given VMS. Each VD is connected to one physical device. If a given smart object has more than one device, e.g. a camera and microphone, and the output stream is multiplexed, then there will be only one VD connected to that smart object. Otherwise, there will be one VD to each device (sensor).

Each VD is programmed to connect to a specific device type (e.g. USB camera) and to send a specific type of multimedia stream, e.g. H.264 video. This approach makes the architecture flexible for incorporating different VD types. The VD output type can not be changed in execution time.

The VD is data stream agnostic since it has no knowledge of the content being carried. This feature decreases the complexity for creating new VD types. A same VD can send data to multiple VMSs.

The relationship between the physical device cannot be changed after the start of the VD. It must be defined using V-PRISM stream descriptors during the setup process and can not be changed after the VD starts running. The VD type must be compatible with the type of data stream and the communication protocol available at the physical device. A VD operates as a driver, interpreting specific data formats and protocols. In contrast, it is possible to have many binds between a VD and VMSs. VD and VMS have a weak relationship since it can be established and changed during execution time.

A VD is defined as a tuple $VD = \{P, C, S, T, E\}$, where $P = \{par_1, \dots, par_n\}$ is a set of parameters $par = \{name : value\}$ that identify the device, such as a path for the device, the device IP address, etc.; $C = \{cnf_1, \dots, cnf_n\}$ is a set of the device configurations $cnf = \{name : value\}$, each device has different configurations, for example, video resolution or audio quality; S is the software that performs multimedia stream collecting and forwarding, this value can be the container ID if the software is running inside a Docker environment, the PID if its running on LXC, or any other type of software identification. T is the VD data type, it defines the type of the VD output, which can be audio or video; E is an optional parameter that defines in which edge node the VD must be started, the edge node can be identified in many ways depending on the implementation infrastructure, in a Docker environment it can be the IP:PORT where the Docker API is listening.

The parameter E is optional, and only needs to be assigned when there is some restriction for deploying VDs. For example, to start a VD for a USB camera, it must be started in the edge node where the USB camera is connected.

Figure 4.11 depicts how the relationship between VMS and VD works. First, the multimedia stream is collected/received by the VD, the stream can be sent to many VMSs and, a VMS can handle a multimedia stream from another VMS. Lastly, an IoMT application will receive the processed stream.

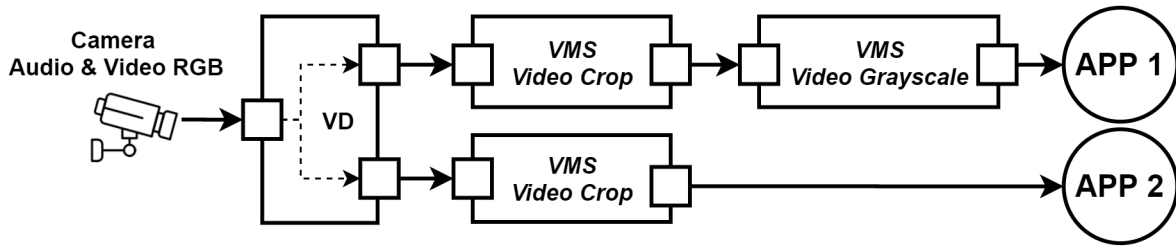


Figure 4.11: Relationship between VD and VMS

4.3.3 VMS Registry & VMS Request Manager

The VMS Registry component is used to manage and provide a descriptor list about the availability of VMS types in each edge node. The IoMT application will query this list before making the VMS creation requests. The descriptor list can be fully retrieved or queried by specific parameters such as location, VMS categories, and VMS types. For example, an IoMT application can request the list of all VMSs that perform voice recognition in a specific location.

As already mentioned, the VMS component processes a multimedia stream and delivers the result to an IoMT application or another VMS. However, the IoMT application is not allowed to directly execute internal functions for instantiating a VMS. The VMS Request Manager (VRM) component is responsible for receiving requests made by the IoMT application to create or destroy a VMS. It is through the VRM that external demands created by IoMT applications reach other V-PRISM components. This component uses a message-driven communication. Such approach allows heterogeneous systems to communicate in a loosely coupled way, and it fits the dynamic nature of IoT and CoT systems. The communication protocol used between these entities is not in the scope of this work.

The main task of the VRM is depicted in Figure 4.12. This component is responsible for managing the admission of new VMS creation requests. The admission is ruled by business and technical aspects. Each V-PRISM implementation will define its own set of rules to deal with the admission of new VMS creation requests.

In Figure 4.12, the first activity "App is Allowed?" will manage all the business questions, for example, here they can verify if that specific IoMT application is allowed by the infrastructure provider to create the requested VMS type. The second activity "VMS is Allowed?" deals with a technical question, for example, here the VRM will verify if there is any running edge node with the VMS types requested by the IoMT

application.

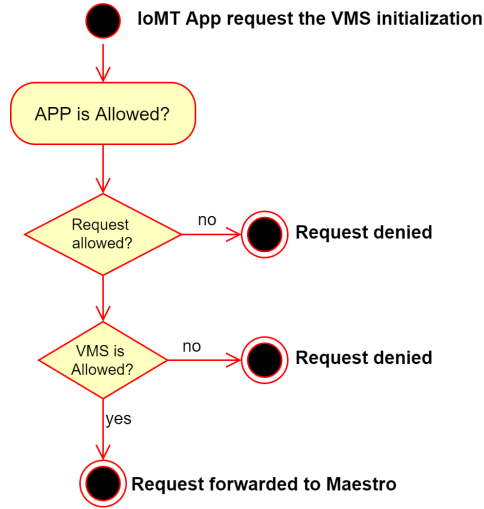


Figure 4.12: VRM state diagram

There are many ways to describe service requests. In [27], the authors use a customized JSON string pattern to specify the characteristics of the requested service. This approach is quite simple, and dealing with a complex application is a problem. In contrast, [56] proposed a sophisticated approach where a directed graph is used to define the sequential steps of the multiples virtual entities that compose the application and, that graph is used by the orchestrator to derive the chaining execution plan. Nevertheless, the definition of the best approach is out of the scope of our current work, and it will be up to the developers to choose it.

4.3.4 Maestro

Maestro maintains an overall view of the system based on deployed edge nodes, the available resources, and network topology. Another responsibility is triggering application reallocation as needed. Maestro was designed based on the specifications defined for the Multi-access edge orchestrator component proposed in [29].

Maestro is the first component executed after a VMS creation request is admitted by VRM. It is a V-PRISM core component and has two main functions. The first one is to manage the VMS life cycle and the second one is to orchestrate the VMS stream pipeline. Maestro works together with other components to provide these functions. In the VMS life cycle management, Maestro uses the Edge Node Manager component to send commands to create, destroy, or move a VMS from one edge node to another. Maestro consumes

functions provided by the Resource Allocation Manager component to determine in which edge node a VMS will be created.

To orchestrate the VMS stream pipeline, Maestro uses the VMS Manager and VD Manager components. The VD Manager will receive messages to bind a VD to a VMS. Furthermore, the VMS Manager will receive commands to attach the VMS output as an input to another VMS.

4.3.5 Resource Allocation Manager

The edge environment, in contrast with the cloud, is resource-constrained. The amount of CPU, memory and storage is limited. Thus resources must be well used to maximize the number of IoMT applications served and improve the delivered QoS [34]. One of the key features to achieve this goal is using proper resource allocation methods.

The Resource Allocation Manager (RAM) is the component that defines in which edge node a VMS must be instantiated. There are two class of information used by RAM to perform this task:

- **Environment data:** they are gathered in the edge environment, edge nodes and network. Examples of this type of data are, the candidate edge nodes, number of VMSs running in each candidate edge node, the delay between edge node and the IoMT application host, CPU usage level, free memory amount, battery level, resources history usage, etc.;
- **Application data:** they are gathered from the IoMT application environment. QoS requirements and stream configurations parameters are some data examples.

Different types of resource allocation algorithms require different parameters. In this version of V-PRISM, for the sake of simplicity, we are considering that the RAM component is capable of delivering all the parameters required by each algorithm, but in future versions, we will introduce a context manager monitor. V-PRISM architecture is capable of running different resource allocation algorithms types, the IoT infrastructure needs to install each possible resource algorithms inside the V-PRISM environment.

The Maestro component has configurable parameters that will define which algorithm will be used to perform the edge node selection process. One possible example is, if the number of available edge nodes is fewer than a certain threshold, Maestro can choose a

$O(n^3)$ algorithm, otherwise, the algorithm must be $O(n \log n)$. This example demonstrates the flexibility level available in V-PRISM.

In the next section, we discuss a Resource Allocation Heuristic Model proposed to select which edge node should be chosen to run a new VMS.

4.3.5.1 Resource Allocation Heuristic Model

Resource allocation (RA) is a process in charge of executing the tasks to allocate resources necessary to meet the workload of one or several applications [25]. In V-PRISM, the RAM component is responsible for executing the RA process. The component has the main task of allocating VMS in some edge nodes in order to meet application requests.

In this section, we propose a heuristic algorithm to be implemented as complement of the RAM component. As this algorithm is not the focus of our work, we implement a naive approach only to validate V-PRISM operation, since the resource allocation is a key part of it. However, it is essential to mention that more sophisticated algorithms proposed in other studies like [90], [89], [8], and [5] could be implemented as well. The problem solved by the algorithm is how to select the best edge node to run a new VMS. Such a problem is an NP-Hard problem because it involves placement decisions [89]. We will use a heuristic method since this is a common way to solve this type of problem. We desire to choose the edge node that will meet the IoMT application requirements to maximize utility regarding response time (latency) and consumed bandwidth. We define some premises to the algorithm development:

- Each edge node has its own VMS type repository;
- Each edge node can have different VMS types;
- The IoMT application requests a single VMS type each time;
- The information about the edge node such as memory, CPU and bandwidth usage is already available before the algorithm starts;
- The resource availability of the edge node should always be checked before starting a new VMS.

IoMT applications are latency-sensitive [8]. One strategy to minimize the latency is to run the VMS in the edge node that has the lowest latency to the IoMT application host.

Algorithm 1 shows this approach. Many metrics can be adopted in a resource allocation algorithm. We choose latency because in IoMT applications it is a critical factor [8].

Algorithm 1 Resource Allocation Algorithm Based on Network Latency

Initialization: *run()*

Function *resourceAllocation(req)*

// Get all edge node candidates.

// It must be online, with VMS type deployed and with sufficient resources available.

 edgeNodesCandidates \leftarrow **call** Maestro.getEdgeNodes(req);

 selected \leftarrow NULL;

while (*en in edgeNodesCandidates*) **do**

if (*selected == NULL or en.latencyToApp < selected.latencyToApp*) **then**
 | selected = en;

end

end

return selected;

Function *run()*

while *true* **do**

 event \leftarrow **call** ServiceRegistry.receiveEvent();

if (*event.type == "VMS_CREATION"*) **then**
 | resourceAllocation(event);

end

end

In the next sections, we will describe the Virtualization Engine components and its role in the V-PRISM architecture.

4.3.6 Virtualization Engine

The VMS and VD are the main components of V-PRISM. This components can be deployed in multiples and heterogeneous edge nodes. Each edge node can run various components at the same time.

Each component can be developed using the technology that better fits its requirements, for example, a VMS that detects movement can be developed in python using OpenCV. In contrast, a VD that collects data from a microphone can be developed using C language. This means that for V-PRISM the VMS is agnostic about technology adopted in the component development. To overcome the heterogeneity of hardware,

software and network communication standards, each component will run inside its private virtual environment based on virtualization. The Virtualization Engine uses the low level API provided by the virtualization platform used to virtualize the components of V-PRISM.

The Virtual Engine component will be responsible for providing interfaces for managing the virtualization of components. A responsibility of this component is to monitor the status of each virtual machine. We advocate that the virtualization system component should use a lightweight approach, like containerization [54]. This approach better fits the edge resource constraints environment.

Another responsibility taken by the virtualization engine component is the communication mechanism between components. It must abstract the heterogeneity of the network environment. In this manner, virtual entities can focus only on their main task.

4.3.7 Edge Node Manager

The Edge Node Manager (ENM) is the component responsible for managing all edge nodes that can run V-PRISM components. In V-PRISM an edge node can be characterized as a small to medium size computing entity that aims to provide computing, storage, and networking resources to the applications deployed across IoT devices, edge and cloud [53]. As we described in Section 2.3, there are three types of edge nodes: single-board computer, IoT gateways and micro datacenters. V-PRISM can be executed in any of them.

The ENM is a key component of V-PRISM. In the MEC Architecture [29], ENM has the same functionalities described in the MEC Platform Manager. ENM is responsible for the following functions:

- Receiving virtualized resource fault reports;
- Performance measurements from the virtualization infrastructure;
- Providing elements for managing V-PRISM core components;
- Preparing the virtualization infrastructure to run a VMS and VD container image.

4.3.8 Environment Monitor & VN Monitor

To orchestrate infrastructure resources, V-PRISM needs to collect statistic data about the environment. The Environment Monitor component gathers data from the edge nodes.

It acts as a statistic data monitor aggregator.

The Virtual Node (VN) Monitor collects statistics data from the virtual entities (VMS and VD). The RAM is the main consumer of these data. The freshness of the data depends on the capabilities of the devices in the edge environment. Typically these data are not collected in real-time because the resource allocation algorithm component is time constrained.

4.3.9 VMS Manager

The VMS Manager component is responsible for providing interfaces to create, destroy, and manage VMSs. Moreover, it keeps track of VMS types available in each edge node. Each edge node has the capabilities to run a subset of all VMS types available. A VMS type is implemented by a software that performs some processing in a multimedia stream.

The definition of which edge node can run each VMS type needs to take into account technological and business aspects. More complex VMS types will require edge nodes with, for example, a GPU or also with abundant storage, so, an edge node without these capabilities can not be part of edge nodes candidate set for the instantiation of a VMS type. On the other hand, even if the edge node has the required resource, it can be exclusively allocated to maintain only specific VMS types to provide some QoS agreement. Thus this definition necessarily must be done by a human or semi-automated process.

The VMS Manager component must be executed in each edge of V-PRISM that will be used to run a VMS. In Figure 4.10 is depicted that this component provides metadata about each VMS running, inside the edge node, to the *VN Monitor*. This data will be used, for example, by the *Resource Allocation Manager* during the instantiation of a new VMS.

Another feature provided by this component is a fault-tolerant subsystem. If one VMS stops working, the VMS manager has mechanisms to recreate it. Besides, the component *Multimedia Stream Sharer* is part of the VMS Manager. This strategy improves the data security of the architecture.

4.3.10 VD Manager

The VD Manager (VDM) component is responsible for providing interfaces to create, destroy, and manage the virtual devices. Each VD stores physical sensor metadata, like

the format of the generated stream, the physical location of the sensor, its data-sampling rate, etc. The attributes stored about each sensor can be distinct and depend on the physical nature of the multimedia sensor.

The VD Manager component must be executed in each edge of V-PRISM that will be used to run a VD. In Figure 4.10 is depicted that this component provides metadata about each VD running, inside the edge node, to the *VN Monitor*. This data will be used, for example, by the *Environment Monitor* to notify the infrastructure owner about any outage of resources inside a VD instance.

Another feature of VDM is to manage the VD types. A VD type is implemented by a software that is able to connect to the physical device. For example, if the physical device provides an RTSP interface, so there will be a VD type that implements this protocol.

A VD has an additional requirement in contrast to VMS. Even if the edge node has the VD type installed, it can only be started if the edge node can access the physical device. For example, the VD of a USB Camera only can be deployed in the edge node where the camera is attached. In contrast, an RTSP virtual device can be deployed in any edge node. The edge node placement of a VD is typically defined manually by the infrastructure manager once the physical device must be configured to be part of V-PRISM.

In Figure 4.13, we can see a USB Webcam and a Raspberry acting as an edge node. When the VD that collects multimedia stream from the physical device is started, it must be executed in the Raspberry edge node where the USB camera is connected. It is not a limitation of the architecture but a consequence of how this type of device can be accessed.



Figure 4.13: USB Webcam attached to a Raspberry ¹

¹<https://tinyurl.com/y9pfcxx3>

4.3.11 Stream Sharer

The Stream Sharer (SS) component provides a function that enables the VMS to send a multimedia stream to more than one destination. By default, the VMS has many input ports but only one output port. This model allows the VMS to focus solely on the main task that is multimedia processing and not in the process of sharing the result stream.

In Figure 4.14, the SS component is depicted. It has one input port and many output ports. Each output port is linked to an IoMT application or a VMS. SS dynamically defines the number of output ports. The SS component only shares the multimedia stream.

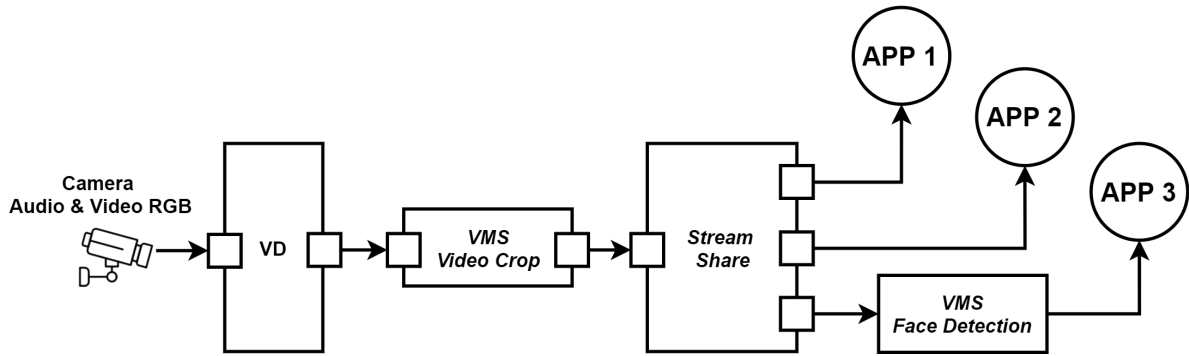


Figure 4.14: Stream Share component operation

4.4 V-PRISM Operation

In this section, we present some important aspects of V-PRISM. The discussion about component deployment will be done in Section 4.4.1, the initialization of a VMS in Section 4.4.2, how the multimedia stream share works in Section 4.4.3, and finally, the Control Message Broker (CMB) will be analyzed in Section 4.4.4.

4.4.1 V-PRISM Deployment

V-PRISM is composed of many distinct components. The components can be deployed in different edge nodes. Every edge node that compounds V-PRISM must have at least the Virtualization Engine component, and one or more VMS types or VD types.

Before ingress in a V-PRISM environment, the edge node must be registered in the *Edge Node Manager* component. The edge node must also have a unique identifier (ID) that will be used to address the control messages sent to it. Typically the virtual engine

adopted in the development already provides some ID, in Docker, for example, the ID is composed by a string with 26 characters.

An edge node can host virtual entities like VMSs and VDs. The software of the virtual entity must be deployed and available at the edge node. Real-time software deployment increases virtual entity startup delay.

It is mandatory some network communication between all edge nodes that compound V-PRISM environment. For improving the V-PRISM security, we advocate that all edge nodes must be part of the same private network, the network can be physical either virtual network. The only components that must have a direct connection to the Internet is the Service Registry and VMSs that send data to an IoMT application hosted in the cloud.

Devices produce multimedia stream. Each device is attached to only one virtual device. Before a VMS uses the multimedia stream provided by a VD, the parameters for VD configurations must be informed, and the virtual device must be started.

4.4.2 Initialization of a VMS

The initialization of a VMS has two phases. The first is the admission phase, where the component VMS Request Manager (VRM) defines if the IoMT application request is valid or not. After that, the second phase is started and it consists in a call to Maestro to start the creation of the VMS. In Figure 4.15, we show a sequence diagram of the second phase.

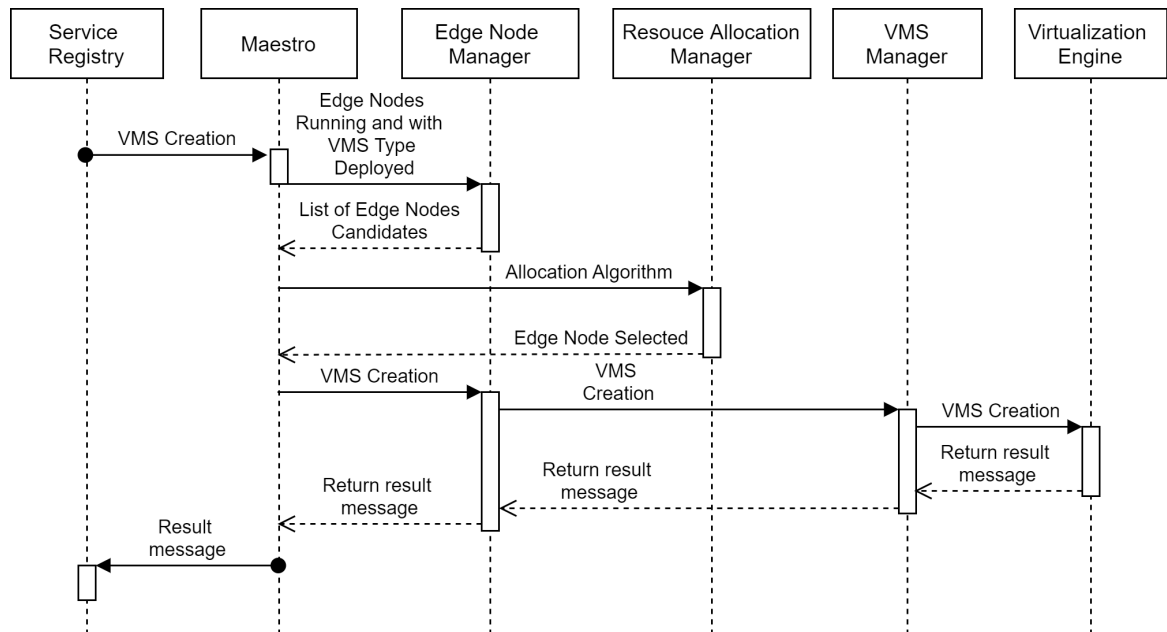


Figure 4.15: VMS creation sequence diagram

First, the Service Registry component sends Maestro the request to create a new VMS. Maestro requests the Edge Node Manager component to retrieve the list of all edge nodes that are running and have already deployed the corresponding VMS type. Finally, the list of candidate edge nodes and the information provided by the IoMT application is sent to the Resource Allocation Manager. It will choose, based on a resource allocation algorithm, one edge node to run the VMS, and that information will be sent to VMS Manager that will call the Virtualization Engine to create the VMS.

4.4.3 Multimedia Stream Sharer

By default, a VMS can only send a multimedia stream to a single destination. This strategy allows the development of a VMS to be focused on multimedia stream processing. The Stream Sharer component is used to share the stream resultant of a VMS processing to multiple destinations.

When a VMS is created, by default, the stream processed by it will be sent directly to the IoMT application that requested it, or to another VMS. In situations where the VMS can share the multimedia stream with more than one entity, this parameter must be explicitly informed to the VMS Manager when the request for the VMS creation. This pattern allows the manager to create an approach to avoid problems with data privacy aspects concerns generated by a VMS that is sharing data inadvertently.

In Figure 4.14, when the VMS Video Crop was created, the *IoMT APP 1* explicitly defined that the result of this VMS could be shared. When the *IoMT APP 2* requests a new VMS with the same specifications of the VMS already created that can share content, the VMS Manager only attaches a new output port to the SS component that was attached to the VMS previously created.

4.4.4 Control Message Broker

The Control Message Broker (CMB) is responsible for providing communication between Maestro and VD. It is essential to mention that VMSs and VDs run inside their private virtualized environment. Thus, direct communication between them increases the coupling of the components. The use of sockets is an example of coupling increasing. We advocate that a better approach to performing the communication with VD is using a publish-subscriber pattern, where each VD subscribes to a specific topic, and Maestro publishes data in this topic when it needs to interact with the VD.

In Figure 4.16, we illustrate the interaction between CMB and virtual devices. In this example, a single security camera is serving two different VMSs. Each VMS is executing a different task and providing multimedia services using the same physical infrastructure.

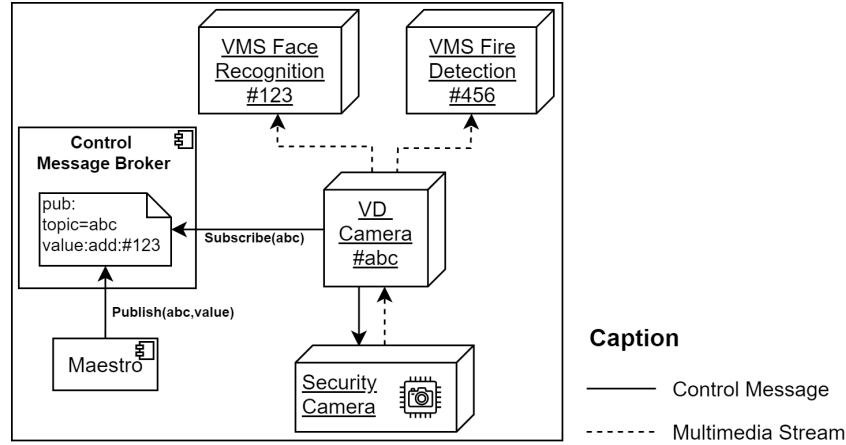


Figure 4.16: Interaction between CMB and Virtual Devices

Another feature presented in Figure 4.16 is the communication sequence between Maestro and CMB. First, the *VD Camera* subscribes to the topic *abc*, and this process occurs when the VD is started. When some VMS needs the stream from this VD, Maestro publishes in topic *abc* a string value. When the value of a topic changes, all the subscribers will receive the new value, then the VD will parse the message to perform the binding between the VD and VMS 123.

When a VD is instantiated, the multimedia data received from the physical device is not delivered to any VMS. Maestro will send a message to the VD via CMB when some VMS is initialized and requests the multimedia stream, or when the VMS does not need the stream anymore. The virtual device is not destroyed automatically, even if no VMS is using the multimedia stream it provides.

In this chapter, we presented V-PRISM, our proposed architecture to virtualize and manage virtual multimedia sensors. We detailed the three-tier architecture in Section 4.1. We depicted our proposed categorization for virtual multimedia sensors in Section 4.2, V-PRISM logic components were discussed in Section 4.3, and finally in Section 4.4 we presented V-PRISM operation. In Chapter 5, we will present ALFA, the proof-of-concept of V-PRISM.

Chapter 5

ALFA: An Implementation of V-PRISM

In this chapter, we will present a V-PRISM implementation as a proof-of-concept (PoC). We named this prototype as ALFA. This implementation instantiates all the components of our three-tier architecture shown in Chapter 4. The source code and additional information like tutorials, manual instructions, and walkthrough videos can be found in GitHub¹.

The rest of this chapter is composed as follows, in Section 5.1 we describe the leading technologies used in the development of ALFA, in Section 5.2 we show some VMS types that we created and, in Section 5.3 we show some virtual devices that were developed. During this chapter, we also show the implementation and details about the software and hardware specifications for running ALFA.

5.1 ALFA Main Technologies

In this section, we will present the leading technologies used to build ALFA. The focal point here is how the interaction between these technologies can be used to generate an environment to host Virtual Multimedia Sensors (VMS) and Virtual Devices (VD) providing essential features to enable IoMT using edge computing to perform multimedia processing. In Figure 5.1, we show ALFA deployment diagram.

The development of a VMS or VD usually requires tools for multimedia stream processing like GStreamer², OpenCV³ or FFmpeg⁴. V-PRISM proposal does not restrict

¹<https://github.com/midiacom/alfa>

²<https://gstreamer.freedesktop.org/>

³<https://opencv.org/>

⁴<https://www.ffmpeg.org/>

which technology developers must use. This is important because it allows the selection of tools that better fits with the individual requirements of each VMS type or VD type. This approach provides flexibility for developers and brings compatibility with already existing tools. In our current implementation, VMSs mainly use GStreamer and OpenCV.

Typical IoT applications are distributed. All components shown in Figure 5.1 can potentially be distributed among different edge nodes. This strategy allows ALFA to be executed in a variety of edge nodes, like a single-board computer [53], IoT gateways [81] or even in micro datacenters [1].

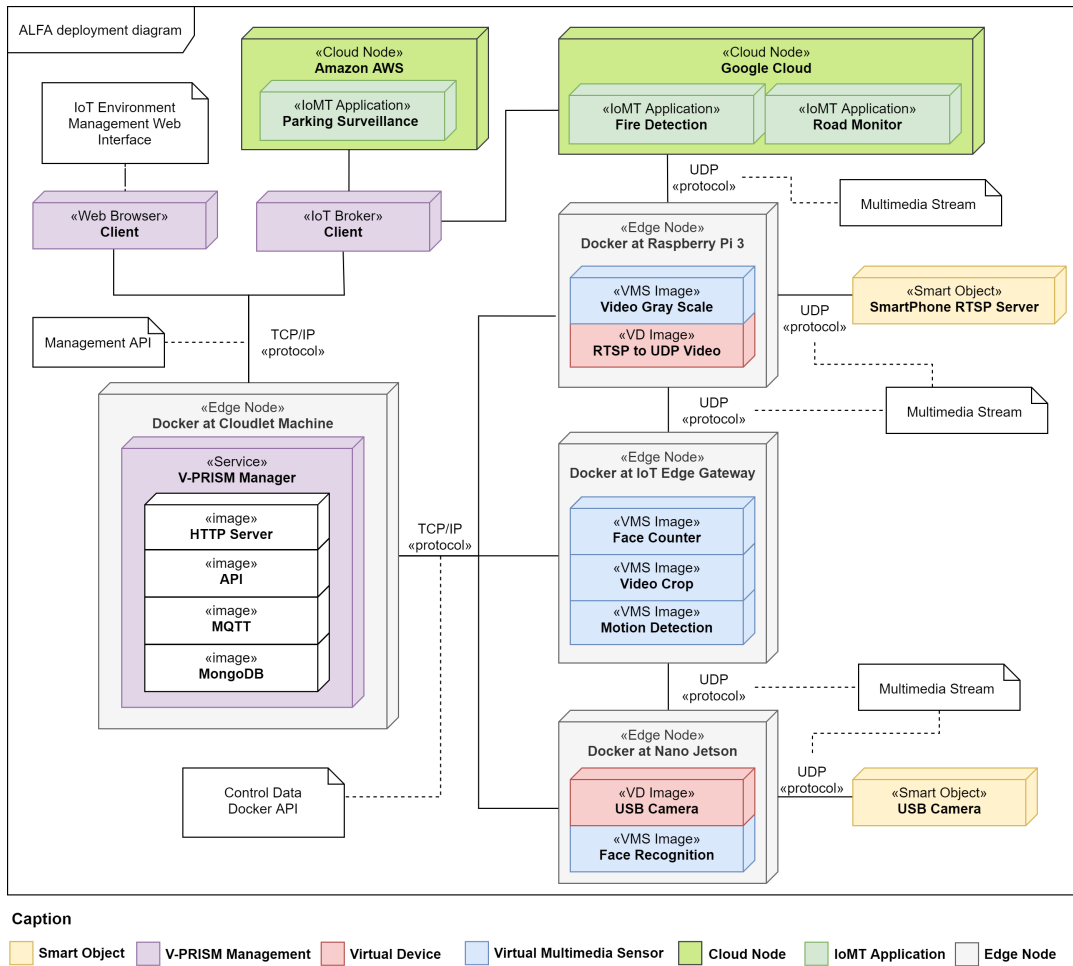


Figure 5.1: ALFA Deployment Diagram

Each VMS or VD is a software instance running inside a Docker container, providing multimedia stream in response to an application or other VMS requests. The VD is the entity responsible for abstracting the computation and communication capabilities of physical nodes [72]. The VMS and VD are concepts based on the idea of microservice,

where each entity is designed to implement one specific task over a multimedia stream.

The communication protocol used to send multimedia data stream between VMS and VD is UDP. In Figure 5.1, the *Smart Object USB Camera* sends a multimedia stream to the VD using UDP. The physical device sends a multimedia stream to the virtual device after the control data message is performed indirectly via publish/subscribe strategy. This approach is used because the components may not be up and running at the moment when the bind message is sent by V-PRISM Manager. Using this strategy, when the components become available, they will receive the message and start the communication.

5.1.1 Edge Nodes

In ALFA implementation, an edge node is a device with storage, memory and CPU, capable of running Docker version 19.0 or higher, and accessible via TCP/IP network. Not every VMS or VD needs to have Internet access, only VMSs that send data to IoMT applications running in the cloud must have Internet connectivity. The component VMS Request Manager must also have an Internet connection. Typically an edge node is a small computer like a Raspberry, IoT gateways or a more robust machine running inside a local micro datacenter. In a few cases, even the IoT device can be used as an edge node. V-PRISM can handle the edge environment heterogeneity.

In an ALFA running environment, we can have many edge nodes. These edge nodes need to work together to execute all the V-PRISM tasks. In Docker, we can have *nodes swarms*, which are groups of machines running Docker and joined into a cluster [26]. All edge nodes in ALFA environment are part of the same Docker swarm. Docker swarms have features like load balancing, scaling, multi-host network and security.

All the edge nodes must be registered to be part of ALFA environment. Figure 5.2 depicts the web application module, which will be presented in Section 5.1.5, used to manage ALFA edge nodes. The status and the number of VMSs in each edge node are data used by the resource allocation algorithm to perform VMS allocation. This module also provides the list of VMS and VD images available at each edge node.

Name	IP	Status	VMS Num.	Actions
Local	localhost	Online	0	[Edit] [Delete] [Refresh] [VN Images]
Raspberry PI	192.168.0.151	Online	0	[Edit] [Delete] [Refresh] [VN Images]
IoT Gateway	192.168.0.152	Offline	0	[Edit] [Delete]
Local Cloudlet 1	192.168.0.153	Offline	0	[Edit] [Delete]

Figure 5.2: Web application module to manage edge nodes

After an edge node ingresses in the Docker swarm, it is necessary to install all the different VMS types and VD types that can run at that edge node. In Figure 5.1, the edge node *Docker Nano Jetson* can run a VD to collect a multimedia stream from a USB camera, and a VMS to perform a *face recognition* task. On the other hand, the *Docker Edge Gateway* can run a VMS for motion detection. Each edge node can run a different set of VMS and VD types.

5.1.2 Docker

ALFA was built over a virtualization paradigm. Docker has tools that enable the use of containers. A container is a standard unit of software that packages up code and all its dependencies, so the application runs quickly and reliably from one computing environment to another [26]. The adoption of containers in a heterogeneous environment, like the edge, can improve the network efficiency, computation and data storage [9].

In ALFA, each Docker container runs a single VMS or VD. Docker handles the containers life cycle. Figure 5.3 depicts the Docker architecture. The Docker API and Docker CLI are tools for managing the Docker infrastructure and containers.

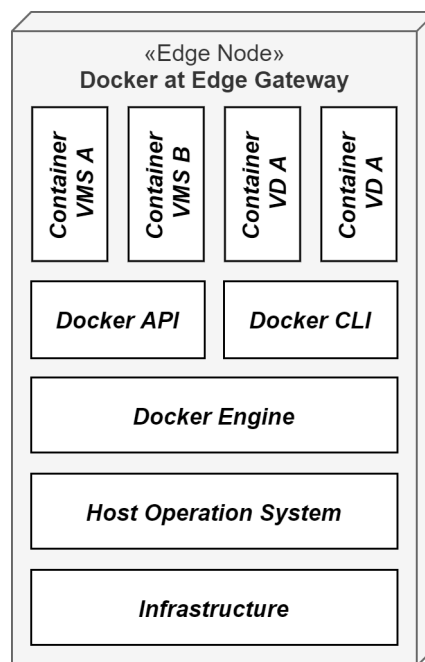
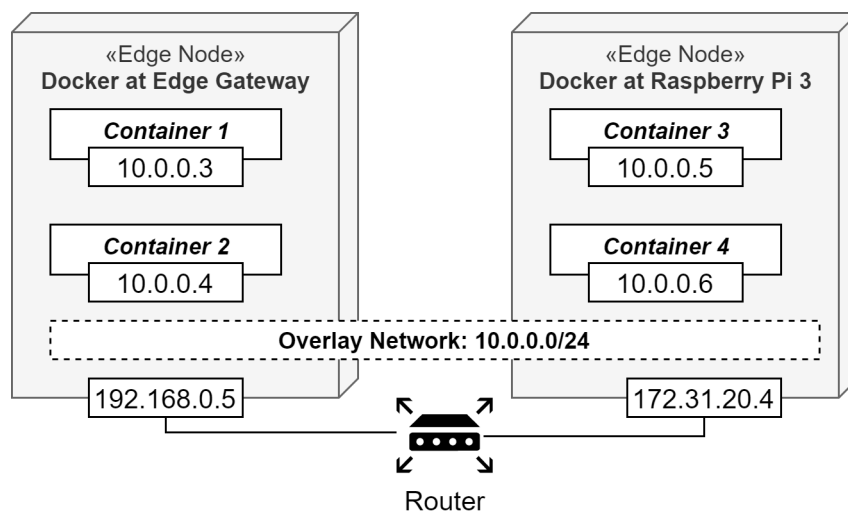


Figure 5.3: Docker Overview

Another useful technology that Docker offers is the network overlay driver. Since each VMS or VD can be deployed in a different edge node and each edge node can be attached to a different network, and geographically distributed, we need to implement a way to enable a container-to-container communication. In Figure 5.4, we can see four containers running in two edge nodes. Despite the edge nodes are attached to different IP subnetworks, the Docker overlay network enables direct communication between the containers as if they were in the same network.

Figure 5.4: Overlay Docker Network⁵

Docker also has a mechanism to collect information about each edge node and the containers running inside it. This information is used by the component resource allocation to manage and distribute the VD over the edge nodes. The information can be accessed via Docker API or Docker CLI.

5.1.3 VD and VMS in Containers

In ALFA, all VMSs and VDs are executed in Docker containers. The flexibility of this approach allows the adoption of any programming language for their development. For example, the *Noise Detector* VMS was written in C, and *Face Counter* was written using Python, both are presented in Section 5.3. Source Code 5.1 shows a version of VMS *Video Crop* written using a tool called `gst-launch`. This VMS receives at port 5000 an input video stream, does a crop operation and sends the resultant video stream to an application that is listening in a destination IP and PORT address.

```

1 # The Docker image with gstreamer installed
2 FROM alfa/docker-gstreamer
3 # Executing the VMS
4 ENTRYPOINT exec gst-launch-1.0 videotestsrc \
5   ! videobalance saturation=0 \
6   ! x264enc ! video/x-h264, stream-format=byte-stream \
7   ! rtph264pay ! udpsink host=$IP port=$PORT

```

Source Code 5.1: Implementation of VMS for video crop

In Source Code 5.1, we used GStreamer library to perform the video stream processing. GStreamer is a powerful tool to manipulate any type of stream. Many VMSs and VDs in ALFA were implemented using this library.

A containerization approach is a lightweight alternative to traditional virtual machines. In [28], the authors define container as a means of providing isolation and resource management to applications. No dependence on hardware emulation provides performance benefits over full virtualization but restricts the number of supported operating systems that can be spawned as guest operating systems. The containerization of VMS and VD in ALFA enables that the same source code can be used in different environments that support Docker containers.

To start a Docker container, first we need to create a container image. A container image is constructed in inside a Dockerfile. The Dockerfile is a text file that embeds all the

⁵<https://youtu.be/nGSNULpHHZc?t=46>

packages that will be installed inside the image. Source Code 5.2 presents the Dockerfile used to start the container that hosts the VMS *Video Crop* presented in Source Code 5.1.

```

1 # Base image from docker
2 FROM alpine:3.10.0
3 # instalation of gstreamer and bash gcc (build-base)
4 RUN apk add --update --no-cache bash build-base gstreamer
    gstreamer-dev gst-plugins-base gst-plugins-good gst-plugins-
    ugly gst-plugins-bad x264-dev x264-libs gst-libav
5 # Copy VMS files to the image
6 COPY ./Makefile ./Makefile
7 COPY ./udp_video_crop.c ./udp_video_crop.c
8 # This script compile the source in the target architecture and
    run the VMS program
9 COPY ./start.sh ./start.sh
10 ENTRYPOINT ["./start.sh"]

```

Source Code 5.2: Dockerfile for the VMS presented in Source Code 5.1.

After the Dockerfile has been created, it must be used to generate the Docker image. Docker CLI provides commands to generate these images. The VMS and VD of ALFA can be installed using a script already available in the installation folder, as shown in Figure 5.5. The VMS or VD must be installed in every edge node where it will be executed.



Figure 5.5: Script to install the Docker Images from VMS and VD in the edge node

Edge nodes are heterogeneous. If the VMS or VD was coded in a compiled language like C, to provide interoperability, the compilation of the code must be done inside the Docker container. For example, an x86 edge node has different libs from an ARM one. Thus the executable file from one architecture does not run in the other. In Source Code 5.2, the *Makefile* and *udp_video_crop.c* are copied to the image, and before the *start.sh* runs the compilation command.

In our proposal, the initialization of containers must not generate overload to the host

system, since it is usually an edge node with low capacity. Docker uses shared libraries from the host machine, which reduces memory consumption and improves the container startup time [57]. Docker cache system allows fast startup of a container from an image that is already compiled.

In Figure 5.6 we can see VMSs and VD in execution inside an edge node. This edge node in particular is also running some ALFA components like *rest-api* and, *web-app*. The *Video Merge* VMS, for example, is running inside the container related to the docker image *alfa/vms/video_merge*. The amount of VMSs that can be executed in an edge node depends on its own resources.

IMAGE	NAMES	COMMAND
alfa/vms/video_merge	tender_babbage	". /start.sh '172.17.0.1 10001'"
alfa/vms/udp_video_black_white	modest_noether	". /start.sh '172.17.0.1 10001'"
alfa/vms/udp_to_udp	unruffled_ramanujan	". /start.sh '192.168.0.150 10026 '"
alfa/device/camera_usb	tender_antonelli	". /start.sh '5ebaba4dd30be70037251cfe /dev/video0'"
alfa_web-app	web-app	"nginx -g 'daemon off;'"
alfa_rest-api	rest-api	"docker-entrypoint.sh npm run dev"
mongo	mongo	"docker-entrypoint.sh mongod"
eclipse-mosquitto	mosquitto	"/docker-entrypoint.sh /usr/sbin/mosquitto -c /mosquitto/config/mosquitto.conf"

Figure 5.6: VMS and VD running in Docker containers

5.1.4 Main Used APIs

An API exposes a set of data and functions to facilitate interactions between programs and allow them to exchange information [52]. In ALFA, two different APIs were used. The first one is the Docker API, used to orchestrate the containers life cycle. The second one was created to enable clients like web clients and IoMT applications that need to interact with ALFA.

The Docker API enables the life cycle of a Docker container to be controlled by external applications. In Source Code 5.3, we can see an example of a NodeJs application initializing a new *Face Counter* VMS. The only entity that uses the Docker API directly is ALFA. External applications will call an ALFA API endpoint to manipulate VMSs.

```

1 var Docker = require('dockerode');
2 var docker = new Docker(edge_node_ip);
3 docker.createContainer({Image: 'alfa/plugin/face_counter'})
4 .then(function(container){return container.start();})

```

Source Code 5.3: Docker API used by ALFA

A typical V-PRISM environment is composed by multiples edge nodes, as we can see in Figure 5.1. When a VMS or a VD is manipulated, the Docker API must be connected to the corresponding host edge node. In Source Code 5.3, the variable *edge_node_ip*

defines in which edge node the VMS or VD will be started.

In ALFA, Docker implements all the functionalities of the virtualization system, namely the orchestration (create, destroy, monitor and configure) of containers that run VMSs via API for processing automation. In [53], authors argue that adopting light virtualization with container technology can bring benefits to IoT. This was our main motivation for adopting Docker, a light virtualization tool. Besides saving processing and memory, starting a Docker container is usually faster compared to a traditional virtual machine [86].

The ALFA REST API provides a set of functions that enable different external components to interact with VMSs. In our implementation, we developed a web application where the IoT infrastructure owner can manage ALFA system components. This approach aims at facilitating the implementation of mobile applications and other types of applications. The web application will be described in the next section.

5.1.5 Web Interface Application

To manage ALFA infrastructure, we developed a web application, depicted in Figure 5.7. The primary user of this interface is the IoT infrastructure owner. In Figure 5.1, the HTTP Server represents the web application service. It can be deployed in any of the edge nodes of the environment. The main modules of the web application are:

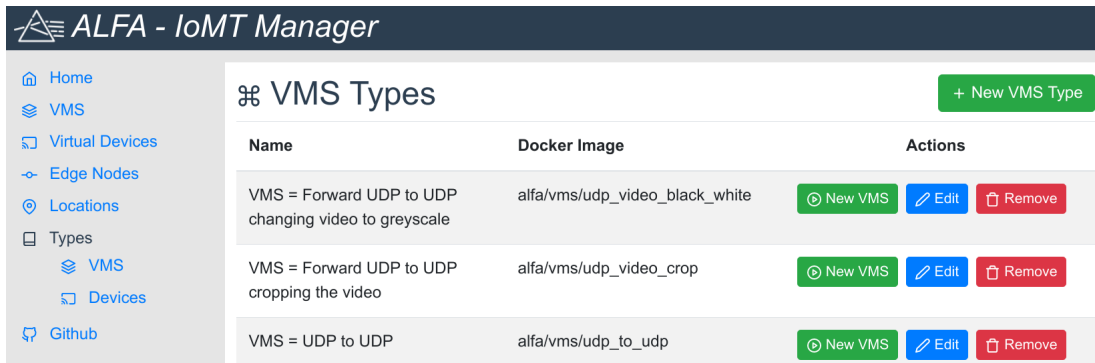


Figure 5.7: Web Interface Application

- **VMS:** It lists all created VMSs inside any edge node and allows VMS management.
- **Virtual Devices:** It lists all the virtual devices created that can be used as source of multimedia data.
- **Edge Nodes:** It manages all edge nodes that will be used to run a VMS or a VD.

- **Locations:** It manages all the places where a physical device can be deployed. A location can be a meeting room, a hall, etc. Many Virtual Devices can be placed in the same location.
- **VMS types:** It manages all the possible VMS types that can be initiated in ALFA deployment.
- **Device Types:** It manages all the possible VD types that can be initiated in ALFA deployment.

As mentioned before, a VD can be a source of a multimedia stream to many VMSs. When we attach a VMS to a VD, we bind it. In Figure 5.8, we show the web application interface used to bind a VMS to a VD.

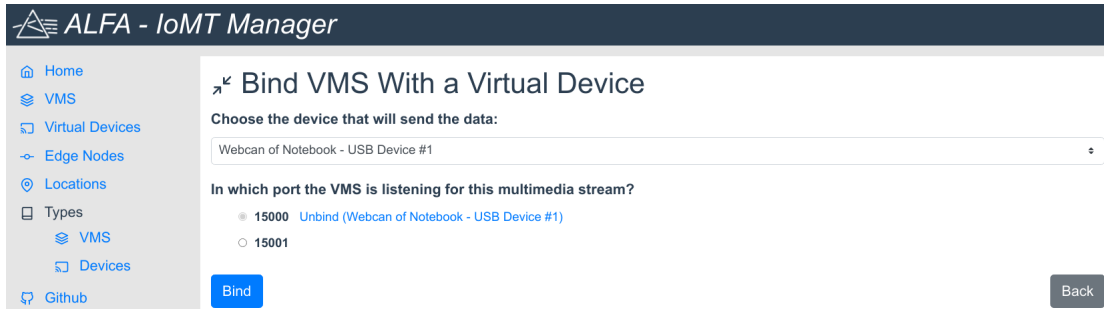


Figure 5.8: Web application bind function

In Figure 5.8, the bind will be between a *Video Merge* and some VD. The *Video Merge* VMS has two input ports, we can see that the port 15000 is already attached with the VD USB Webcam, and the port 15001 can be attached with other VD. Each VMS developer will define in each port the VMS will listen.

VMSs and VDs are separated processes running in different Docker containers. The message that starts the bind between a VMS and a VD arrives by a publish/subscribe service implemented with MQTT⁶. There is an MQTT client inside any VD waiting for messages to start or stop the multimedia transmission to a VMS. In ALFA, as can be seen in Figure 5.6, there is an Eclipse Mosquitto MQTT server running inside a Docker container providing this communication feature. We chose an MQTT server because, as presented in [82], it has lower delay than CoAP[78] messages at lower packet loss rate networks.

⁶<http://mqtt.org/>

In Figure 5.9, we can see an MQTT client connected to the MQTT server used to send bind commands to the VD. We adopt a simple string pattern where each variable is separated with a ";". In the example, the VD with the ID *5ebaba4dd30be70037251cfe* will receive the message *10.0.2.127;15000;15000298415787;A*. This message will be parsed, and the multimedia stream will be sent to the VMS associated white IP 10.0.2.127 and port 15000. A more sophisticated approach like XML could be used, but as already studied in [73], they usually are computationally costly.

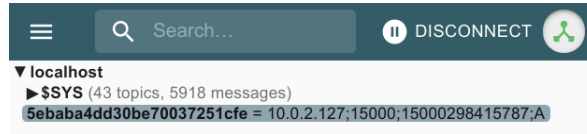


Figure 5.9: MQTT client displaying the topic used to bind VMS and VD

5.1.6 Resource Allocation Management

As presented in Section 4.3.5, there are many strategies to perform resource allocation (RA). ALFA potentially supports different types of algorithms to deal with this task. In Source Code 5.1.6, we show an implementation of a Round-Robin approach for resource allocation. We chose a Round-Robin approach to test ALFA because it is one of the less complex methods [88] and it is capable of running in machines with few resources.

```

1 const docker = require("../util/dockerApi")
2 const ra_rr = {
3   run: async function(payload) {
4     let node = payload.nodes[0]
5     for(let i = 1; i < payload.nodes.length; i++){
6       // Select the edge node with less running VMS
7       if (node.virtualEntityNum > payload.nodes[i].
          virtualEntityNum)
8         node = payload.nodes[i]
9     }
10    return node
11  }
12 }
13 module.exports = ra_rr

```

Source Code 5.4: Round Robin resource allocation algorithm

Every RA algorithm in ALFA implements a run method. This method will receive an object via parameters. This object has a list of all edge nodes where the requested VMS

can run and a list of parameters provided by the IoMT that requests the VMS. These parameters can be used to improve the system QoS (quality of Service).

5.2 Implemented Virtual Devices

In this section, some of virtual devices (VD) developed in our implementation will be presented. Each VD is a Docker image that contains the source code of a software that knows how to collect or receive data from a multimedia device. By presenting those VDs, we will explore some possibilities that V-PRISM brings to IoMT and edge computing.

There are two types of communication between the VD and VMS. The first one is the multimedia stream (received by UDP), and, the second one is the control data (received by TCP), as detailed in Figure 5.1. A multimedia stream flows in only one direction, from a VD to a VMS, and, it can be represented in a variety of application formats like H.264, WAV, etc. On the other hand, control data can flow in both directions, and they are text messages sent to an MQTT server.

The *RTSP to UDP Video* is a VD created to establish a connection with devices that use Real-Time Streaming Protocol (RTSP). RTSP [76] is an application-level protocol, that provides an extensible framework to enable controlled, on-demand delivery of real-time data, such as audio and video. As a widely adopted protocol, data from many IoMT devices can be accessed over RTSP. Because of that, We developed a VD capable of connecting to devices that use this protocol. This VD type collects data from a device that is not physically attached to the edge node where the VD is running, the device uses a WiFi or another wireless infrastructure to transfer the data to the edge node.

The *USB Camera* and *Mic* are VDs that collect data from devices attached to the edge node where the VD container is running. By default, a Docker container is forbidden to grab data from the host machine devices. To allow it and bind a host device (camera, microphone) to a container, when the container is started, it is necessary to explicitly give access and map the external device to an inside path at the Docker container.

For example, in Figure 5.1, the *SmartPhone RTSP Server* and *USB Camera* devices are multimedia devices whose streams are sent via UDP to a VD. The first one uses a wireless connection, and the second one is attached to the edge node. ALFA can handle different types of devices.

5.3 Implemented Virtual Multimedia Sensors

In this section, some of the VMSs developed in ALFA will be presented. Each VMS, as we mentioned previously, is encapsulated in a Docker image that contains the source code to process a multimedia stream and deliver the result to an IoMT application or another VMS. ALFA is flexible, and new VMS types can be developed and installed as needed.

The *Video Greyscale* VMS converts a colored video stream in grayscale. This VMS is useful in situations where the IoMT application runs some Hough transform algorithms, like the *HoughCircle* present in OpenCV. The first step of a Hough transform is to convert the video stream to grayscale, because of that, transferring the colored video will be a resource waste. In our categorization, this VMS is a *Transformer*.

The *Video Crop* VMS cuts part of a video stream. This VMS is useful in situations where the application will need only a fraction of the complete video. For example, a surveillance application can be interested only in the video of the door instead of the video of all the environment. In our categorization this VMS is a *Transformer*.

The *Video QR Code Detection* VMS can be used to detect and extract data from a QR Code inside a video stream. This VMS is useful when a QR Code is used by the user to interact with a remote application using QR Code and camera to execute some task. The QR Code data extracted can be used by the application as a signal to start the execution of another task. In our categorization this VMS is a *Detector*.

The *Noise Detector* VMS can post in an MQTT topic if the sound volume of the environment was higher than a configured threshold. This VMS is useful in the Industrial Internet of Thing (IIoT) to monitor facilities and machines where the noise level can be used to predict a dangerous incident. In our categorization, this VMS is a *Detector*.

The *Face Counter* VMS can post in an MQTT topic the number of faces identified in a video stream. This VMS can be combined with another VMS to count the number of people in an environment. This VMS was created to exemplify the adoption of deep learning for multimedia processing. We used the *Face Recognition* library provided at GitHub⁷. In our categorization, this VMS is a *Detector*.

An example of a *Video Mosaic* VMS used by an IoMT application is depicted in Figure 5.10. This VMS has two video input ports. These ports allow that two VDs send video stream to the VMS. The VMS combines the streams before sending to the IoMT

⁷https://github.com/ageitgey/face_recognition

application. In our categorization, this VMS is an *Aggregator*.



Figure 5.10: VMS Video Mosaic

One of the premises of V-PRISM is the flexibility to incorporate new VD Types. it allows that VMSs and VDs can be built using different techniques. In ALFA, we test these two features, by developing multiple VMSs and VDs for different goals and using different techniques, to validate the approaches proposed in V-PRISM.

In this chapter, we presented our PoC named ALFA that follows the V-PRISM architecture. We implemented the main components of V-PRISM and validated some premises of V-PRISM. In the next chapter, we will discuss some experiments used for the evaluation of our proposal.

Chapter 6

Evaluation

This chapter describes the experiments performed to evaluate V-PRISM architecture and its implementation named ALFA. We developed experiments to analyze our work from different perspectives, as depicted in Figure 6.1. The perspectives are **IoMT Device**, where we analyze the behavior of an IoMT device when it is integrated with V-PRISM; **Placement**, where we compare edge and cloud deployments; **Development**, where we analyze, via an informal study, V-PRISM easy of usage by developers, and **Scenarios**, where we present use cases of V-PRISM adoption.

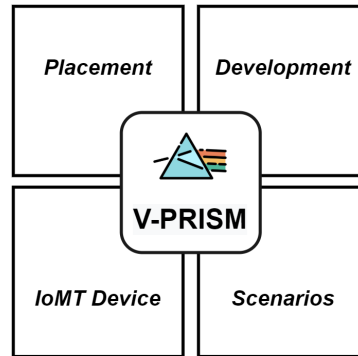


Figure 6.1: V-PRISM analyzed from different perspectives

In order to plan the experiments related to the three first perspectives (IoMT Device, Placement and Development), we adopted the Goal Question Metric (GQM) [11] approach. GQM model is a hierarchical (three levels) structure that, in each level, refines the granularity of what is relevant in order to provide reliable insights on a phenomenon. A **goal** represents which phenomenon should be analyzed, where each goal can be represented as one or more **questions**, and for each question, a set of metrics will be used to answer it. Table 6.1 presents the goals used for evaluating this work.

Table 6.1: Goals definition

Goal	Goal description	Perspective
G1	Analyze if the adoption of V-PRISM in IoMT environments improves the resource consumption in the IoMT device and in the IoMT network in comparison with the traditional approach.	IoMT Device
G2	Analyze if the adoption of V-PRISM to process multimedia streams in the edge improves QoS aspects of IoMT application in comparison with the cloud approach.	Placement
G3	Analyze if the adoption of V-PRISM facilitates the development of VMSs.	Development

6.1 Resource usage in IoT Device and IoT Network

IoT devices typically have limited resources. The amount of CPU, memory, storage, battery [64] and network [63] are scarce. One of the key features to enable IoT is to improve resource usage in the IoT devices. Considering the importance of such requirement, in order to assess how V-PRISM contributes to satisfying it, we have defined goal **G1** as *Analyze if the adoption of V-PRISM in IoMT environment improves its resource consumption in the IoMT device and IoMT network in contrast with the traditional approach.* The traditional approach in this experiment is defined as the configuration where the IoMT application collects data directly from the device. Table 6.2 presents the questions for goal G1 and Table 6.3 presents the metrics used to answer such questions.

Table 6.2: Questions for G1 goal

Question	Question description
Q1	Does the proposed architecture reduce the amount of CPU usage in the IoMT device, compared with a traditional approach?
Q2	Does the proposed architecture reduce the bandwidth usage in the IoT network (the IoT network is the network used by IoT devices to send collected data), compared with a traditional approach?
Q3	Does the proposed architecture reduce the amount of battery usage in the IoMT device, compared with a traditional approach?

Table 6.3: Summary of the metrics used to answer the questions of G1 goal.

Metric	Metric description	Question
CCO	CPU Consumed (CCO) is the total time, in seconds, of CPU usage in the IoT device to capture, process and transfer the data.	Q1
MBT	Megabits Transferred (MBT) represents the total amount of data, measured in Mega bits, transferred from the IoMT device to the entity (VMS or IoMT application) that consumes the multimedia stream.	Q2
BCO	Battery Consumed (BCO) is the percentage of battery that was consumed in the IoMT device during the process of capturing, processing and transferring data.	Q3

To evaluate the effect of V-PRISM adoption in IoMT environments, we conducted the experiments depicted in Figure 6.2. The first one, Figure 6.2(a), was performed in a traditional environment where the IoMT application collects data directly from the IoMT device. The second one, Figure 6.2(b), was performed in a V-PRISM environment where the IoMT application collects data from a VMS.

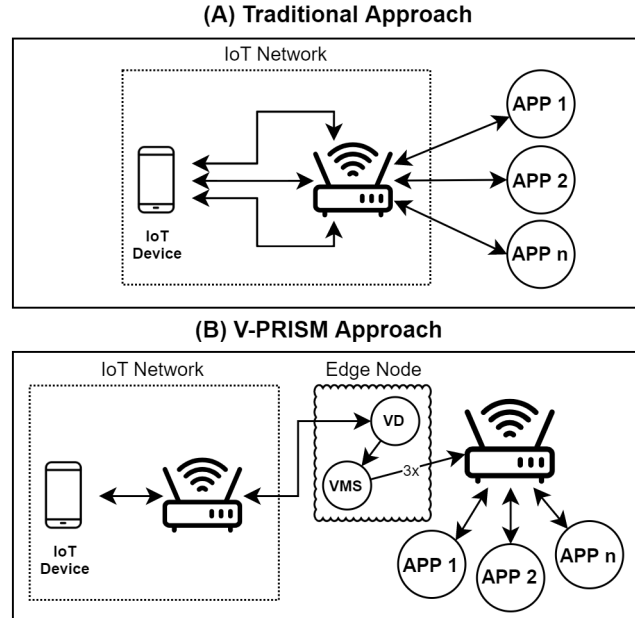


Figure 6.2: Design of the experiment about resource usage in IoT Device and IoT Network

In this experiment, many IoMT applications received a video stream from an IoMT device. The IoMT device was Moto G5 smartphone using an Android 8.1 as operating system. The video was collected and shared using an application called *IP Webcam*¹.

¹<https://play.google.com/store/apps/details?id=com.pas.webcam>

The IoMT device was connected via WiFi to a dedicated access point. The video stream was shared using the RTSP protocol. The video generated by the IoMT device has 640x480 spatial resolution and was converted to H.264 before sending to IoMT application.

The experiment was conducted in rounds. In each round, the number of IoMT application receiving the data was increased. The number of IoMT application in each round was [1, 2, 5, 10, 20]. Each round was executed during five minutes, and, it was performed five times. For each round, we collect the metrics CCO, MBT and BCO, described in Table 6.3. The values presented in the next figures are the mean values of these metrics with 95% confidence interval.

Before running the tests to collect the data, we executed the guidelines proposed by Testdevlab team [80]. This guideline enables collecting accurate data. The authors suggested that the following steps must be executed in the IoMT device before running the experiment: factory reset, run the security upgrades, disable GPS, disable Bluetooth and NFC, disable 4G network, disable mobile data, set the screen bright to the minimum available, enable the safe battery mode.

The data was collected in the IoMT device using the software Batterystats². This tool was developed by Google Android Developers Team, and it provides a set of metrics for understanding the resource consumption in Android devices. The data produced by Batterystats were analyzed using the software Battery Historian³, also created by Google Android Developers Team.

The first metric analyzed was the CPU Consumption (CCO), and it helps to answer question Q1. Figure 6.3 depicts CCO data from the experiment in each round. The blue bars represent the CPU usage when we use the V-PRISM approach, and the red bars denote the traditional approach. The amount of CPU resources usage in the IoMT device is stable when we use V-PRISM approach, but it grows in the traditional method when the number of IoMT applications connected to the device increases.

²<https://developer.android.com/topic/performance/power/setup-battery-historian>

³<https://developer.android.com/topic/performance/power/battery-historian>

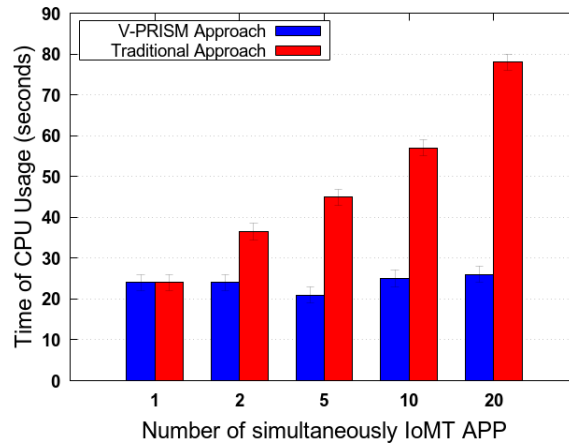


Figure 6.3: IoMT device CPU consumption

The second metric analyzed was the Megabits Transferred (MBT), the amount of data transferred in the IoT Network, and helps to answer question Q2. Figure 6.4 depicts the data from the experiments performed to collect the MBT in each round. The blue bars represent the CPU usage when we adopt the V-PRISM approach, and the red one is when we adopt the traditional approach. The total of data transferred over the IoT Network is stable when we use V-PRISM independently of the number of IoMT applications, however in the traditional approach, the total of data transferred grows as the number of IoMT applications grows.

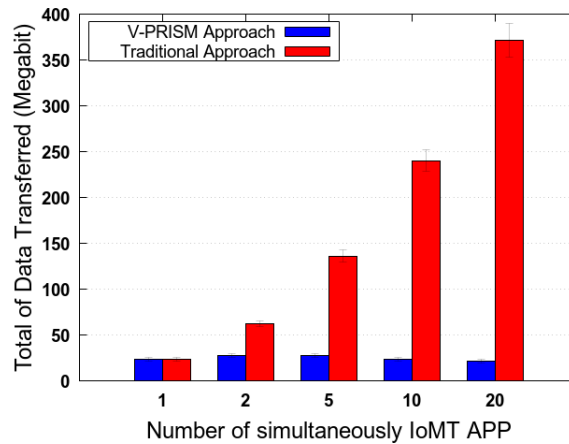


Figure 6.4: IoMT device network bandwidth consumption

The third metric analyzed was the amount of battery consumed: Battery Consumed (BCO), and it helps to answer question Q3. Figure 6.5 depicts the data from the experiments performed to collect the BCO in each round. The blue bars represent the battery consumed when we use the V-PRISM approach, and the red one is when we use the tra-

ditional method. The battery consumed when we use V-PRISM approach remains stable, but it rises in the traditional approach when the number of IoMT applications connected to the device increases.

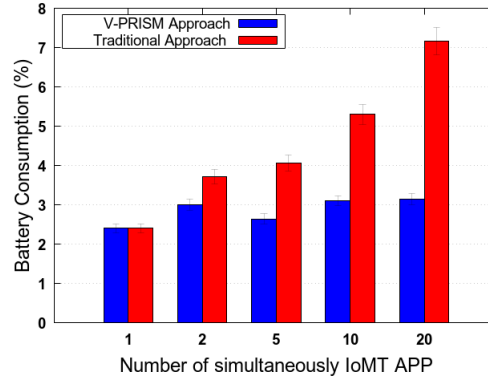


Figure 6.5: IoMT device battery consumption

For question Q1 *"Does the proposed architecture reduce the amount of CPU usage in the IoMT device, compared with a traditional approach?"* we can conclude based on the observed data that there was a significant reduction of CPU usage when using V-PRISM. For 20 IoMT applications, the adoption of V-PRISM reduces the CPU usage in 66%. This result is achieved because for each new application connected to the device, a new thread is started. Each thread sends data for its specific IoMT application, and each thread consumes CPU resources.

For question Q2 *"Does the proposed architecture reduce the bandwidth usage in the IoT network, compared with a traditional approach?"* we can conclude based on the observed data that there was a significant reduction of bandwidth usage. For 20 IoMT applications, the adoption of V-PRISM reduces the bandwidth usage in 94%.

For question Q3 *"Does the proposed architecture reduce the amount of battery usage in the IoMT device, compared with a traditional approach?"* we can conclude based on the observed data that there was a significant reduction of battery consumption. With 20 IoMT applications, the adoption of V-PRISM reduces the battery consumption in 56%.

The data presented in Figures 6.3, 6.4 and 6.5 show that the adoption of the proposed architecture can improve the resource consumption in the IoMT environment. It can be observed that, as expected, the greater the number of IoMT applications directly accessing the IoMT device to collect the data, the greater the resource consumption in the IoMT device and IoMT network. In contrast, when the access is made through V-PRISM approach, resource consumption in the IoMT environment remains stable.

Finally, we can conclude that goal G1 that is *Analyze if the adoption of V-PRISM in IoMT environment improves its resource consumption in the IoMT device and IoMT network in contrast with the traditional approach* was achieved once the answers of Q1, Q2 and Q3 indicate that the adoption of V-PRISM reduces the resources used in the IoMT device and IoMT network. In the next section, we will discuss experiments to identify if the placement (edge or cloud nodes) of the V-PRISM components affects the QoS aspects of IoMT applications.

6.2 Comparison Between Edge and Cloud

To evaluate the characteristics of placing V-PRISM components in the edge and cloud for processing multimedia streams, we performed two experiments. In the first experiment, we processed audio stream, and in the second one, we processed video stream. Each experiment and results will be described in the next sections. They will be used to provide answers for the questions related to Goal **G2** that is *Analyze if the adoption of V-PRISM to process multimedia stream in the edge improves QoS aspects of IoMT application in contrast with the cloud approach*. Table 6.4 presents the questions for G2 goal. Table 6.5 presents the metrics used to answer the questions related to G2 goal.

Table 6.4: Questions for G2 goal

Question	Question description
Q4	Does the deployment of the proposed architecture in the edge reduce the total of data loss during the multimedia stream processing, compared with the cloud deployment?
Q5	Does the deployment of the proposed architecture in the edge reduce the total delay in the IoMT applications, compared with the cloud deployment?

Table 6.5: Summary of the metrics used to answer the questions of G2 goal.

Metric	Metric description	How it is calculated	Quest.
FRL	Frame Loss (FRL) is the percentage of lost frames with the image that should trigger the event detection alert.	In our experiment, it is the ratio of the number of QR Code frames detected by the total number of QR Code frames sent by the device.	Q4
QFD	Quantity of Frame Detected (QFD) is the percentage of frames with the image that will trigger the event detection alert.	1 - FRL.	Q4
QDE	Quantity of Data Extracted (QDE) is the percentage of frames whose data could be extracted by the VMS.	It is the ratio of the number of QR Code frames detected whose data could be extracted by the number of QR Code frames detected.	Q4
DTN	Detection Time of Noise (DTN) is the time difference between the detection of successive noise.	In the input audio, each noise was played with 5 seconds interval, DTN is the time difference of two consecutive noise detection events.	Q5
FFD	First Frame Detection (FFD) is the time when the first video frame was available to be processed.	It is the time difference between the instant when the virtual device starts sending the video stream and the instant when the first video frame with QR Code was available in the VMS.	Q5
FDM	Frame Detection Difference (FDD) is the difference between the time instants of successive detection events.	In the video used as input for the experiment, each QR Code is displayed for 1 second, the FDM represents the time difference between two consecutive detection values.	Q5

6.2.1 Noise Detector Experiment

This experiment was conducted to evaluate aspects of audio stream processing in the edge and cloud. The experiment is depicted in Figure 6.6. It was developed to obtain the metric Detection Time Noise (DTN) that will be used for helping to answer question Q5. The *Noise Detector* VMS, described in Section 5.1.3, was the entity that provided the data for this experiment.

For the deployment of the experiment, we used the following environment configurations. The edge node was emulated using a virtual machine with 1 GB RAM and 1 vCPU, 1.8 GHz, Intel i7 8565U. The edge node is running inside a local WiFi network at Niterói, Brazil, and from now on this node will be only called edge node. The cloud node was a virtual machine with 1 vCPUs, 2.5 GHz, Intel Xeon Family, and 1 GB of RAM. It is located physically at the United States at North of Virginia in Amazon Cloud, and from now on this node will be only called cloud node. The one-way-delay was calculated using the ping command.

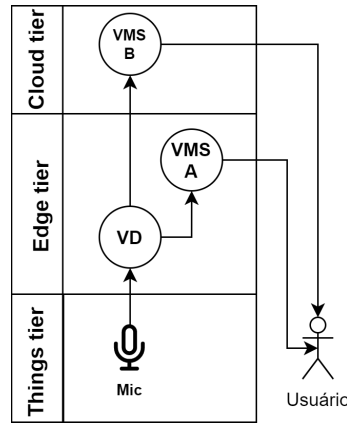


Figure 6.6: Experiment design of audio processing in edge and cloud nodes

The first step of the experiment was deploying two *Noise Detector* VMS, one in the edge node and the other in the cloud node. The microphone was placed at the same room of the edge node where the virtual device (VD) was deployed. A noise source was configured to make a beep sound, and it was played 80 times with 5 second interval between each beep.

The delay between the VD and the cloud was 67.5ms, whereas the delay between the VD and the edge node where the VMS was deployed was negligible. The available bandwidth between the edge environment and cloud was 100Mbps. We performed multiple tests in different moments to minimize the network congestion effect.

The audio stream produced has a bitrate was 1Mb/s. The stream, as we can see in Figure 6.6, is sent in parallel to both VMSs. The time interval between successive noise detection events in the VMS in the cloud and in the edge was calculated, it is the DTN metric. In Table 6.6 is depicted the results of the experiment. The average noise detection time in the edge was 918ms with a standard deviation of 88ms; in the cloud, it was 967ms with a standard deviation of 137ms. The noise detection in the edge node was in average 49ms lower in comparison to the cloud node.

Table 6.6: Interval between successive noise detection events.

	AVG Noise Detection (ms)	STD Deviation (ms)
Edge	918	88
Cloud	967	137

The standard deviation of detection in the edge was lower than the cloud, thus pointing to higher predictability of delay in the edge than in the cloud. Another important point to notice is that the average difference between detection time in the edge and in the cloud was less than the delay between the VD and cloud, meaning that cloud processing time was shorter than edge processing time. This is expected, since the cloud has a higher computing power than edge devices. Therefore, we can conclude that the choice between edge or cloud processing should consider not only the delay between the physical device and the cloud but also the processing time of the multimedia stream.

Multimedia and real-time applications are latency-sensitive [50]. Thus, proposals that reduce the total delay are increasingly important since the demand for this kind of application has been rising. In the next section we will investigate the delay in video stream processing.

6.2.2 Video Processing Experiment

The latency and total time of video stream processing are critical QoS parameters [59] that must be addressed during the design phase of an IoMT application. The experiment described in this section assesses QoS aspects of video processing in VMS deployed in the edge and cloud nodes. In this experiment, we collected metrics Frame Loss (FRL), Quantity of Frame Detected (QFD), Quantity of Data Extracted (QDE), First Frame Detection (FFD) and Frame Detection Difference (FDD). We used the same environment for edge and cloud described in Section 6.2.1.

For this experiment we use the *Video QR Code Detection* VMS, described in Section 5.1.3. It was used as an event detection over a video stream. This VMS detects and extracts information of a QR Code inside a video frame. The video used as data source for this experiment has 70 seconds and was encoded in H.264 with 25 frames per second (1750 frames in total), and with 200x200 spatial resolution. We inserted a QR Code image in 450 frames of these frames. One example of video used the experiments is depicted in Figure 6.7.

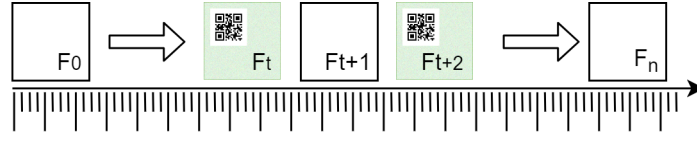


Figure 6.7: Example of Video frame with the QR Code

We run the experiment for each environment ten times. Table 6.7 shows the mean values of the data collected. The column named *QR Code Detected* represents the number of video frames with a QR Code that were detected. The column *Data Extracted* represents the number of video frames with a QR Code detected whose stored data could be retrieved. The column *FRL* represents the percentage of frames with QR Code not detected in the VMS. The column named *QFD* represents the percentage of frames with QR Code detected in the VMS. The column named *QDE* represents the percentage of frames with QR Code whose data were extracted by the VMS.

Table 6.7: FRL, QFD and QDE metrics over 450 QR Code frames.

Tier	QR Code Detected	Data Extracted	FRL	QFD	QDE
Edge	318	219	29%	71%	69%
Cloud	357	303	20%	80%	85%

Extracting features from a video stream, like QR Code detection, is an intensive task in CPU and memory. The data presented in Table 6.7 show that the computational power of the cloud was decisive to obtain better processing results in contrast with the edge. It is important to note that some QR Codes were detected, but the data was not extracted because of packet loss or CPU shortage.

To obtain metrics *First Frame Detection* (FFD), and *Frame Detection Difference* (FDD), we run the experiment depicted in Figure 6.8. To execute the components of this experiment, we use the Containernet [67] network emulator. Containernet is a Mininet extension [45], and it allows the execution of Docker container as host machines inside the emulated network. As each VMS is running inside Docker containers, we can connect V-PRISM components to Containernet network to run the experiment. The script that defines the topology is developed in Python. The network links between the elements have parameters that can be configured depending on the experiment goals. We used the delay parameter to differentiate the traffic for cloud and edge node artificially. In this emulation, we defined the same limits of CPU and memory for the edge and cloud node, since we are interested in the delay metric only.

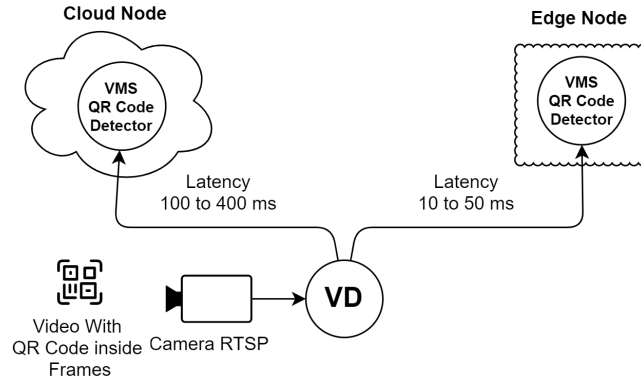
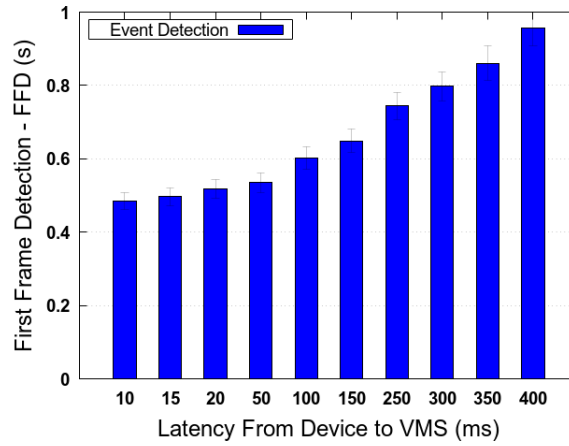


Figure 6.8: Experiment design to obtain metrics for total latency

In Figure 6.9, each bar represents a node where the VMS *Video QR Code Detection* is running. The nodes with latency between 10ms and 50ms represent the edge nodes. The nodes with a latency greater than 50ms represent the cloud nodes. We can observe that the time for the detection of the first QR Code (y-axis of the graph) grows as the latency grows. It means that even in a situation where there is sufficient bandwidth between the IoMT device and the node that runs the VMS, the latency affects the total detection time significantly.

Figure 6.9: *First Frame Detection* (FFD)

In Figure 6.10, each bar represents a node where the VMS *Video QR Code Detection* is running. The nodes with latency between 10ms and 50ms represent the edge nodes, and the nodes with a latency greater than 50ms represent the cloud nodes. The interval between each successive QR Code frame in the original video is 1 second. We can observe that the FDD metric is approximately 1.1 second independently of the latency. It means that the latency does not affect the time between successive event detection in the same

multimedia video stream.

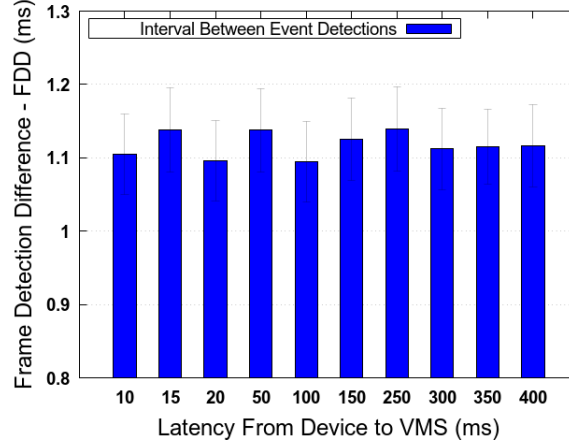


Figure 6.10: *Frame Detection Difference (FDD)*

Based on metrics FRL, QFD and QDE, we can answer question Q4 *Does the deployment of the proposed architecture in the edge reduce the total of data loss during the multimedia stream processing, compared with the cloud deployment?* as following. The adoption of V-PRISM in the edge was not able to reduce the overall data loss based on our experiment. The main cause of this behavior is that the cloud has more computational power and could process the multimedia stream at a higher rate than the edge.

Based on metrics DTN, FFD and FDD, we can answer question Q5 *Does the deployment of the proposed architecture in the edge reduce the total delay in the IoMT applications, compared with the cloud deployment?* as following. The adoption of V-PRISM in the edge can reduce the overall delay during the processing of multimedia stream, but the time difference between successive event detection is still the same independently of the placement of the VMS.

Finally, we can define that goal G2, namely *Analyze if the adoption of V-PRISM to process multimedia stream in the edge improves QoS aspects of IoMT application* was answered once the results presented in this section show that the adoption of V-PRISM in edge nodes cannot reduce the total data loss. However, it can reduce the total time for event detection in video stream when the VMS is placed in edge nodes, improving some aspects of QoS.

6.3 Development of New VMS Types

Software development for IoMT environment poses many challenges. Some of them are the high heterogeneity of applications, protocols and devices [68]. Considering that V-PRISM (in this case, ALFA) can be extended with the creation of new VMS types, it is important to assess how V-PRISM contributes to software development in IoMT. Thus, we defined goal **G3** as *Analyze if the use of V-PRISM facilitates the development of VMSs*. Table 6.8 presents the questions for goal G3 and Table 6.9 presents the metrics used to answer those questions.

Table 6.8: Questions used for addressing G3 goal

Question	Question description
Q6	Have the developers declared that V-PRISM is easy to use?
Q7	How effectively are the V-PRISM mechanisms for dealing with the implementation of new VMS types?

Table 6.9: Summary of the metrics used to answer the questions of G3 goal.

Metric	Metric description	Question
ETU	Easy to Use (ETU), here we are interested if the developers could easily create new VMS types.	Q6
VEF	V-PRISM Effectiveness (VEF), here we are interested if our architecture provides tools that effectively help the creation of new VMS types.	Q7

To answer G3 we performed an experiment where developers used V-PRISM architecture to create new VMS types. The developers were master and PhD students in the program of Computing at Universidade Federal Fluminense. The VMS type creation was the final test of the Multimedia Systems course. There were 3 participants in the experiment, each one developing a different VMS. Table 6.10 describes the function of each VMS.

Table 6.10: New VMS type description

Name	Category	Description
<i>UDP Flex</i>	Improver	This VMS provides data forward and data stream analytics.
<i>Gunshot Alert</i>	Detector	This VMS identifies gunshots in an audio stream.
<i>S2T</i>	Transformer	This VMS converts a voice stream into text.

The VMS *UDP Flex* can act in two modes. As a data forwarder, when it receives the data in a particular port and forwards it to one or more destinations. When used as a data analytics, besides forwarding data, it generates logs that can be used by other components to perform proactive actions. For example, an alert can be sent if the throughput of a stream is below than a threshold. The main challenge of this VMS is that it should perform its tasks as quickly as possible, therefore it was developed in C programming language.

The VMS *Gunshot Alert* can identify gunshots in an audio stream. The main challenge of this VMS is to extract the basic features of the stream to execute the detection algorithm. It was developed in Python, and the multimedia stream library was GStreamer. When a gunshot is detected, the VMS sends metadata about the event to an MQTT Server. After that, some application can consume this data and execute some action.

The VMS *S2T* can convert the words spoken in a voice stream into text. The main challenge of this VMS is to use machine learning techniques. This VMS can perform the conversion in the edge node using the SpeechRecognition library⁴, or using the Google Speech Recognition⁵, an API running in the cloud. The text extracted from the audio stream is sent to an MQTT Server. After that, another application can consume this data and execute an action.

Before starting the development of the VMS, the participants received training about how V-PRISM works and what the main components of the architecture do. The objective of each VMS was selected by the developer. After that, we created a chat group where they collaborated exchanging information, knowledge and tips about the development of VMSs. All the participants were able to conclude the implementation of their VMS type.

After finishing the VMS implementation, the developers filled out a form depicted in Table 6.11. The data was analyzed to answer the questions Q6 and Q7 presented in Table 6.8. It is important to note that the intention was to perform an informal data collection and analysis. We are committed to extrapolate this experiment with a vast number of developers as future work. Even adopting an informal approach, this part of our work helps understanding the implications of the adoptions of V-PRISM.

The form questions F1 to F5 provide data for the metric *Easy to Use* (ETU) that helps to answer the question Q6 *Have the developers declared that V-PRISM is easy to use?*. Table 6.12 presents the snippets of the developer's answers.

⁴<https://pypi.org/project/SpeechRecognition/>

⁵<https://cloud.google.com/speech-to-text>

Table 6.11: Form to collect the perception of the developers about V-PRISM

	Description	Type	Question
<i>F1</i>	I am a specialist in multimedia stream software development.	Likert	Q6 and Q7
<i>F2</i>	I am a specialist in IoT software development.	Likert	Q6 and Q7
<i>F3</i>	I have experience with deploying applications using containers.	Likert	Q6 and Q7
<i>F4</i>	Which technologies were used to develop your VMS?	Open Text	Q6
<i>F5</i>	Have you already mastered these technologies, or was it necessary to learn?	Open Text	Q6
<i>F6</i>	What aspect of V-PRISM facilitated the development of your VMS?	Open Text	Q7
<i>F7</i>	What were the biggest challenges faced for the development of your VMS?	Open Text	Q7

Table 6.12: Answer snippets that help to understand metric ETU

	Answer Snippet
R1	I had already mastered Python and had made experimental use of the SpeechRecognition library.
R2	I already master the language to develop the VMS type, but I needed to learn the MQTT pattern.
R3	I believe that my inexperience with the Linux operating system, total lack of knowledge about Docker made the integration part take longer than usual. However, the integration of my source code with V-PRISM was done in hours, even by an inexperienced person.
R4	The documentation and fast access with the developer of V-PRISM.
R5	V-PRISM developer support, the GitHub of the project and developer videos on YouTube make development easier.

The form questions F1, F2, F3, F6 and F7 provide data for the metric *V-PRISM Effectiveness* (VEF) that helps to answer the question Q7 *How effectively are the V-PRISM mechanisms for dealing with the implementation of new VMS*. Table 6.13 presents snippets of the developer's answers.

Based on the data presented in Table 6.12, we can answer question Q6 as following, the developers declared that V-PRISM is easy to use because it provides a guideline to development of new VMS types, and they can use technologies, programming languages and techniques that they already know. Besides that, the documentation and examples of VMS types provided by V-PRISM help to accelerate the process development.

Table 6.13: Answer snippets that help to understand metric VEF

	Answer Snippet
R6	The Logs page on the web interface was vital to understand the behavior of my VMS.
R7	Viewing examples of other VMS source, Dockerfile and start.sh facilitated the process.
R8	The central aspect was to develop without thinking about the integration with V-PRISM.
R9	I believe that the operation on containers helped a lot, as there are many different technologies behind multimedia processing.

Based on the data presented in Table 6.13, we can answer question Q7 as following. The V-PRISM mechanisms are effective because they facilitate the manipulation of the multimedia stream pipeline helping the developer to understand the multimedia application flow. Besides that, the deploying method of the VMS allows the developers to focus on the main objective of each VMS type, leaving the complexity of the heterogeneity of edge nodes for the V-PRISM architecture.

Although there were a few users, we can still conclude that we achieved goal G3 once we identify evidence during the answering of Q6 and Q7, indicating that V-PRISM facilitates the development of new VMS types. Another indication that corroborates this conclusion is that all developers that started the experiment finished their work. Even more, this experiment shows that developers that do not know V-PRISM were able to create new VMS types validating the idea of an architecture that can be extended by the creation of new VMS and VD Types.

6.4 Other Examples of V-PRISM Use Cases

In this section, we will present three other use cases for V-PRISM. The first one depicts a fault-tolerant scenario in a surveillance system. The second one shows a scenario of multimedia stream sharing. The last one presents the use of pipeline multimedia stream processing.

The first scenario presented is a fault-tolerant surveillance system. In this scenario, a same place is monitored by more than one surveillance camera. In Figure 6.11, we present this scenario.

In Figure 6.11(a) we have a scenario where both cameras are working. In this case,

the VMS was configured to save the stream of *Cam A* in the data store. In Figure 6.11(b), *Cam A* stops working and the VMS automatically changes the input stream and saves the video produced by *Cam B*. This type of automation enables a dynamic environment where autonomous and semi-autonomous systems decided over the multimedia stream what is the best action to perform in a fault-tolerant situation.

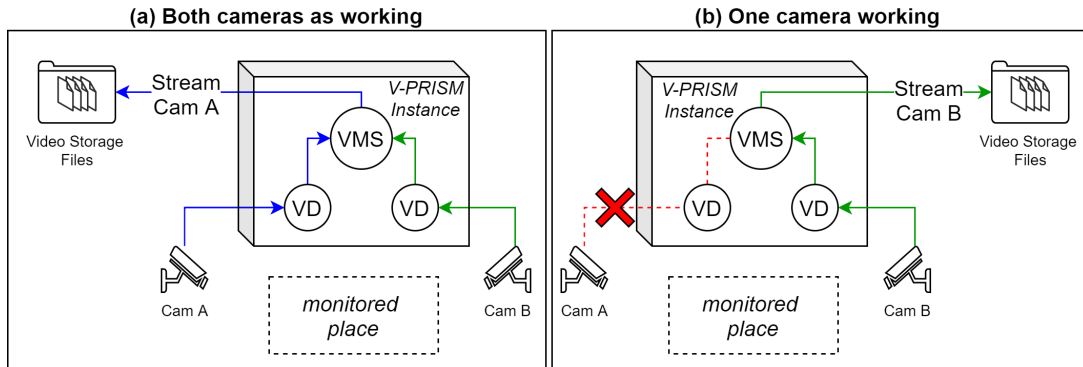


Figure 6.11: Fault-tolerant scenario

The second scenario shows an example of multimedia stream sharing in an IoMT environment. The scenario is depicted in Figure 6.12. Three different VMSs are receiving the multimedia stream provided by a camera. Fire Detection is a VMS that identifies the presence of fire in a video stream. The Intruder Alert is a VMS that sends an alert if a person is detected in the environment during the night and, the Smart Lock is a VMS that opens a door only by an authorized person.

The deployment and management of multimedia devices and edge nodes are difficult and expensive. Here, we can see two different situations where V-PRISM can maximize the usage of the already deployed IoT infrastructure. The first case is when the same device can be used as a source of data to multiple VMS, the second case is when the same edge node can run multiples VMS. Besides that, the same VMS can receive multimedia streams from different virtual devices. This strategy can also increase Return On Investment (ROI) in IoT infrastructure.

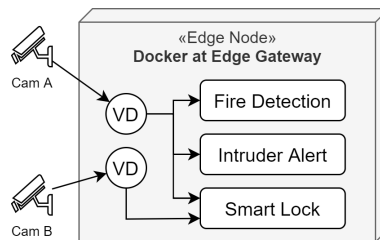


Figure 6.12: ROI increasing scenario

This last scenario presents a multimedia stream pipeline processing. A pipeline multimedia stream processing is the act of broken down elaborate process into isolated simple process. Each isolated process are chained in a logical sequence to obtain a defined result. Each simple part of the chain can be deployed into different edge nodes. The V-PRISM architecture adoption enables the creation of on-demand task execution pipelines that can be spanned over multiple resource-constrained edge nodes. Besides that, the opportunistic nature of IoT service ecosystems, that is crucial to capture the real potential of IoT, presented in [17], can be explored in V-PRISM once the already deployed devices and edge nodes can be used unpredictably. This level of abstraction allows the infrastructure owner to adjust the architecture in real-time to solve a vast amount of problems.

As depicted in Figure 6.13, each edge node is created for different vendors. Besides, they also have different CPU, memory and storage capabilities. V-PRISM automatically manages the process of creating the VMSs, making the binding and the allocating of each VMS in the appropriate edge node.

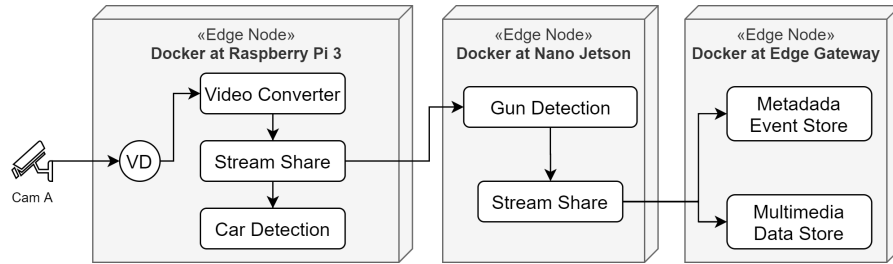


Figure 6.13: Execution of on-demand multimedia stream pipeline

In this chapter, we presented the evaluation methodology used to validate V-PRISM. In Section 6.1, we analyzed the adoption of V-PRISM over the resource consumption in IoT devices. In Section 6.2, we compared the QoS aspects of V-PRISM usage in the edge and cloud nodes. In Section 6.3, we analyzed if the use of V-PRISM facilitates the development of new VMS types. And finally, in Section 6.4, we presented other examples of use cases of V-PRISM. In the next chapter, we will present our final remarks, main contributions, limitations, and future work.

Chapter 7

Conclusion

In this work, we presented V-PRISM, an innovative architecture to virtualize and manage virtual multimedia sensors. Our proposal relies on a three-tier architecture where devices, placed in the things tier, generate multimedia streams that are processed by entities called virtual multimedia sensors (VMS), deployed in the edge of the network, whose data output is delivered to IoMT applications that typically are running in the cloud.

V-PRISM adopts the concept of lightweight virtualization. This approach enables the deployment of a self-contained application, releasing developers from the responsibility to manage the heterogeneity existent in edge computing. Moreover, it facilitates the distribution of VMS among different edge node vendors, thus avoiding dependence on any technology, besides consuming fewer resources than traditional virtualization approaches.

We also proposed a new hierarchical VMS categorization that follows the multimedia stream abstraction provided by the VMS as the parameter to differentiate the VMS types. This categorization provides templates for each VMS type, which can facilitate the work of the developers and also help to guide the total ammount of resources that will be needed to instantiate each VMS type based on the category it belongs to.

To validate V-PRISM, we developed an implementation as a proof-of-concept (PoC) named ALFA. This implementation instantiates the components of our three-tier architecture presented in Chapter 4. This PoC is released with an open-source license and can be accessed in GitHub ¹. Three of VMSs available in this repository were implemented by the developers that participated of the experiment detailed in Section 6.3.

Other important feature of our implementation is that we also created various VMS and VD types, described in Sections 5.2 and 5.3. These VMS and VD will help the

¹<https://github.com/midiacom/alfa>

developers to start their new components. ALFA counts with an extensible mechanism to run different types of algorithms to resource allocation that are used to the deployment of VMS in multiple edge nodes. In Chapters 4, 5 and 6 we presented evidences and data to answer the research questions posed in the introduction of this work, which are:

RQ1: *Can a multimedia sensor virtualization architecture enable a single multimedia device to provide multimedia streams for different applications?*

Answer: As presented in Chapters 4 and 5, the proposed architecture and the proof-of-concept show that is viable to create a structure where one single device can be a source of data to multiple VMS. This approach leverages the sharing of the physical nodes and has the potential to increase the ROI of the IoT infrastructure once new applications can be deployed without significant time and money investment.

RQ2: *Can a multimedia sensor virtualization architecture reduce CPU usage and battery consumption in IoT devices?*

Answer: As presented in Section 6.1, the adoption of V-PRISM can significantly reduce the resources consumed in the IoT device. Energy consumption and resource constraints are key factors in IoT environments, so the adoption of V-PRISM can help to improve such usage while promoting a greener, more sustainable solution in comparison to non-virtualized environments.

RQ3: *Can a multimedia sensor virtualization architecture reduce bandwidth usage in the IoT network?*

Answer: As presented in Section 6.1 the adoption of V-PRISM can significantly reduce the total bandwidth consumed in the IoT network. In environments with limited network capabilities, this strategy can enable the use of latency-sensitive applications.

RQ4: *Can an edge-based multimedia sensor virtualization architecture reduce the delivery time of a multimedia stream to an IoMT application, compared to using a virtual multimedia sensor hosted in the cloud?*

Answer: In Section 6.2, we performed experiments with audio and video event detection. The results show that the adoption of V-PRISM in edge nodes can reduce the total time of multimedia stream processing. In contrast, the adoption of a cloud deployment brings a more robust strategy for avoiding data loss.

Considering the goals and the research questions defined in this work, we can conclude that the main desired contributions were achieved. We proposed an architecture to

manage VMSs in edge computing environments; We created a hierarchical classification for VMSs based on the functionalities and resources consumed, and finally, we developed several types of VMS that can be used as templates for facilitating the creation of new VMS types and VD types.

In this work, we focused our efforts on defining an architecture and validate its technical viability. Despite our efforts, some limitations were detected during the work. The main limitations and issues of our work are:

- There are some theories used to formally validate software architectures, one example is the Architecture Tradeoff Analysis Method (ATAM) [40]. One key requirement to execute this validation is the number of participants. Due to time limitations and lack of resources we performed an informal data collection and analyzed in Section 6.3, thus we cannot extrapolate those results for a large group of developers.
- It is important to highlight that containerization is not the "silver bullet" that kills all the heterogeneity problems. For some VMS types, it was necessary to create the containers for each specific type of edge node. It occurs because different architectures, sometimes, have different libraries for the same propose. MongoDB is an example, it has a specific image for Raspberry 3, with configurations inside the Docker image for this specific architecture.
- The interest for light virtualization has been growing in recent years. Because of that, many light virtualization engines were developed. Docker, Kubernetes, KVM and LXC are some examples. We develop ALFA only to work within Docker. It should be relevant to test V-PRISM with different lightweight virtualization platforms to compare if our architecture could be developed in other virtualization engines.

As discussed in Chapter 3, for the best of our knowledge, V-PRISM is the first architecture that provides the virtualization of multimedia sensors using light virtualization in the edge. We have already published parts of our work in the *IEEE Virtual World Forum on Internet of Things 2020*. The title of our work was *V-PRISM: An Edge-Based IoT Architecture to Virtualize Multimedia Sensors* [13], in the published paper, we cover the first V-PRISM version. And in the *XXV Workshop de Gerência e Operação de Redes e Serviços (WGRS)* we publish the work [12] that describe with more details our PoC.

During the development of this work, we had many new ideas about features and characteristics that could be integrated into V-PRISM, which are interesting future work:

- The IoMT environment is composed of sensors and actuators. In this version of V-PRISM we specified only the virtual sensors. In the next version of our architecture, we will also include the virtual entity for dealing with multimedia actuators. This idea can bring to actuators the same benefits of the virtualization paradigm created using sensor virtualization;
- The IoT sensor produces mostly discrete data, like temperature, light and humidity sensor. We will investigate if the proposed architecture can be used to process these types of stream too.
- FIWARE is a curated framework of open source platform components to accelerate the development of smart solutions [30]. This framework is globally used in smart cities projects. FIWARE already mastered the manipulation of discrete data, but, the integration with multimedia sensors are in development. We will work in the integration of ALFA with the FIWARE framework.
- Some VMSs receive a multimedia stream from many virtual devices whose data may be provided by different tools. In this work, we do not discuss this feature in detail, but it will be important to develop some temporal synchronization mechanisms in the future.
- The available resources in the edge environment can change rapidly. For example, new edge nodes can be included or removed to adjust the system demands. Because of that, it is relevant to develop some strategy to better allocate and scale VMSs horizontally and vertically.
- Our resource allocation algorithm used a naive approach. We intend to perform the implementation and tests with more robust strategies.
- Develop new CPU-intensive VMS types such as facial recognition and speech translation, to analyze the influence of the VMS type and QoS aspects.

References

- [1] AAZAM, M.; HUH, E. N. Fog computing micro datacenter based dynamic resource estimation and pricing model for IoT. *Proceedings - International Conference on Advanced Information Networking and Applications, AINA 2015-April* (2015), 687–694.
- [2] AHMED, A.; PIERRE, G.; AHMED, A.; PIERRE, G.; CONTAINER, D.; COMPUTING, F.; AHMED, A.; PIERRE, G. Docker Container Deployment in Fog Computing Infrastructures. In *IEEE International Conference on Mobile Cloud Computing* (Oxford, United Kingdom, 2018).
- [3] AL MACHOT, F.; KYAMAKYA, K.; DIEBER, B.; RINNER, B. Real time complex event detection for resource-limited multimedia sensor networks. *2011 8th IEEE International Conference on Advanced Video and Signal Based Surveillance, AVSS 2011* (2011), 468–473.
- [4] ALAMRI, A.; ANSARI, W. S.; HASSAN, M. M.; HOSSAIN, M. S.; ALELAIWI, A.; HOSSAIN, M. A. A survey on sensor-cloud: Architecture, applications, and approaches. *International Journal of Distributed Sensor Networks 2013* (2013).
- [5] ALVES, M. P.; DELICATO, F. C.; SANTOS, I. L.; PIRES, P. F. LW-CoEdge: a lightweight virtualization model and collaboration process for edge computing. *World Wide Web 23*, 2 (mar 2020), 1127–1175.
- [6] ALVI, S. A.; AFZAL, B.; SHAH, G. A.; ATZORI, L.; MAHMOOD, W. Internet of multimedia things: Vision and challenges. *Ad Hoc Networks 33*, May (2015).
- [7] ANTONINI, M.; VECCHIO, M.; ANTONELLI, F.; DUCANGE, P.; PERERA, C. Smart audio sensors in the internet of things edge for anomaly detection. *IEEE Access 6* (2018), 67594–67610.
- [8] ARAL, A.; BRANDIC, I.; URIARTE, R. B.; DE NICOLA, R.; SCOCA, V. Addressing Application Latency Requirements through Edge Scheduling. *Journal of Grid Computing 17*, 4 (2019), 677–698.
- [9] ARUNA, K.; PRADEEP, G. Performance and Scalability Improvement Using IoT-Based Edge Computing Container Technologies. *SN Computer Science 1*, 2 (2020), 1–7.
- [10] BARNETT, T.; JAIN, J.; USHA, A.; KHURANA, T. Cisco Visual Networking Index (VNI) Complete Forecast Update , 2017 – 2022. Tech. Rep. December, Cisco, 2018.
- [11] BASILI, V. R. Software modeling and measurement: the Goal/Question/Metric paradigm, 1992.

- [12] BATTISTI, A.; MUCHALUAT-SAADE, D. C.; DELICATO, F. C. Uma proposta de arquitetura para virtualização de sensores multimídia na borda da rede. In *Anais do XXV Workshop de Gerência e Operação de Redes e Serviços* (2020), SBC.
- [13] BATTISTI, A.; MUCHALUAT-SAADE, D. C.; DELICATO, F. C. V-PRISM: An edge-based iot architecture to virtualize multimedia sensors. In *2020 IEEE 6th World Forum on Internet of Things (WF-IoT)* (2020).
- [14] BOSE, S.; GUPTA, A.; MUKHERJEE, N.; ADHIKARY, S. Towards a sensor-cloud infrastructure with sensor virtualization. *Proceedings of the International Symposium on Mobile Ad Hoc Networking and Computing (MobiHoc) 2015-June* (2015), 25–30.
- [15] BOTTA, A.; DE DONATO, W.; PERSICO, V.; PESCAPÉ, A. Integration of Cloud computing and Internet of Things: A survey. *Future Generation Computer Systems* 56 (2016), 684–700.
- [16] BUCKLAND, M. K. Information as thing. *Journal of the American Society for Information Science* 42, 5 (jun 1991), 351–360.
- [17] CASADEI, R.; FORTINO, G.; PIANINI, D.; RUSSO, W.; SAVAGLIO, C.; VIROLI, M. Modelling and simulation of Opportunistic IoT Services with Aggregate Computing. *Future Generation Computer Systems* 91 (2019), 252–262.
- [18] CELESTI, A.; MULFARI, D.; GALLETTA, A.; FAZIO, M.; CARNEVALE, L.; VIL-LARI, M. A study on container virtualization for guarantee quality of service in Cloud-of-Things. *Future Generation Computer Systems* 99 (2019), 356–364.
- [19] CHENARU, O.; HANGANU, C. E.; POPESCU, D.; ICHIM, L. Virtual sensor for behavior pattern identification in a smart home application. *2019 8th International Conference on Systems and Control, ICSC 2019* (2019), 388–392.
- [20] CICIRIELLO, P.; MOTTOLA, L.; PICCO, G. P. Building virtual sensors and actuators over logical neighborhoods. *ACM International Conference Proceeding Series* 218 (2006), 19–24.
- [21] COZZOLINO, V.; OTT, J.; DING, A. Y.; MORTIER, R. ECCO: Edge-Cloud Chaining and Orchestration Framework for Road Context Assessment. In *2020 IEEE/ACM Fifth International Conference on Internet-of-Things Design and Implementation (IoTDI)* (apr 2020), IEEE, pp. 223–230.
- [22] CRISPEN, R. G.; STUCKEY, L. D. STRUCTURAL MODEL: Architecture for software designers. *Proceedings of the Conference on TRI-Ada 1994* (1994), 272–281.
- [23] DAS, A.; PATTERSON, S.; WITTIE, M. EdgeBench: Benchmarking edge computing platforms. *Proceedings - 11th IEEE/ACM International Conference on Utility and Cloud Computing Companion, UCC Companion 2018* (2019), 175–180.
- [24] DE CASTRO PERDOMO, D.; VITERBO, J.; SAADE, D. C. M. A location-based architecture for video stream selection in the context of IoMT. *Proceedings of the 25th Brazilian Symposium on Multimedia and the Web, WebMedia 2019* (2019), 461–468.

- [25] DELICATO, F. C.; PIRES, P. F.; BATISTA, T. *Resource Management for Internet of Things*, 1 ed. Springer, 2017.
- [26] DOCKER. What is a Container?, 2020. <https://www.docker.com/resources/what-container>, Last accessed on 2020-04-08.
- [27] DONASSOLO, B.; FAJJARI, I.; LEGRAND, A.; MERTIKOPOULOS, P. Demo: Fog Based Framework for IoT Service Orchestration. *2019 16th IEEE Annual Consumer Communications and Networking Conference, CCNC 2019* (2019), 1–6.
- [28] DUA, R.; RAJA, A. R.; KAKADIA, D. Virtualization vs containerization to support PaaS. *Proceedings - 2014 IEEE International Conference on Cloud Engineering, IC2E 2014* (2014), 610–614.
- [29] ETSI. MEC 003 - V2.1.1 - Multi-access Edge Computing (MEC); Framework and Reference Architecture. Tech. rep., ETSI, Valbonne/França, 2019.
- [30] FIWARE. About us, 2019. <https://www.fiware.org/about-us/>, Last accessed on 2020-07-01.
- [31] GAT, E.; SLACK, M.; MILLER, D.; FIRBY, R. Path planning and execution monitoring for a planetary rover. In *Proceedings., IEEE International Conference on Robotics and Automation* (1990), IEEE Comput. Soc. Press, pp. 20–25.
- [32] GUPTA, A.; MUKHERJEE, N. A Cloudlet Platform with Virtual Sensors for Smart Edge Computing. *IEEE Internet of Things Journal* 6, 5 (2019), 8455–8462.
- [33] HARDY, N.; MAROOF, A. A. ViSIAR - a virtual sensor integration architecture. *Robotica* 17, 6 (1999), 635–647.
- [34] HASSIJA, V.; CHAMOLA, V.; SAXENA, V.; JAIN, D.; GOYAL, P.; SIKDAR, B. A Survey on IoT Security: Application Areas, Security Threats, and Solution Architectures. *IEEE Access* 7 (2019), 82721–82743.
- [35] IEEE COMMUNICATIONS SOCIETY. *IEEE Std 1934-2018 : IEEE Standard for Adoption of OpenFog Reference Architecture for Fog Computing*. IEEE, 2018.
- [36] ISLAM, M. M.; HASSAN, M. M.; LEE, G.-W.; HUH, E.-N. A Survey on Virtualization of Wireless Sensor Networks. *Sensors* 12, 2 (feb 2012), 2175–2207.
- [37] JEONG, Y.; JOO, H.; HONG, G.; SHIN, D.; LEE, S. AVIoT: Web-based interactive authoring and visualization of indoor internet of things. *IEEE Transactions on Consumer Electronics* 61, 3 (2015), 295–301.
- [38] KABADAYI, S.; PRIDGEN, A.; JULIEN, C. Virtual sensors: Abstracting data from physical sensors. *Proceedings - WoWMoM 2006: 2006 International Symposium on a World of Wireless, Mobile and Multimedia Networks 2006* (2006), 587–592.
- [39] KARAAGAC, A.; DALIPI, E.; CROMBEZ, P.; DE POORTER, E.; HOEBEKE, J. Light-weight streaming protocol for the Internet of Multimedia Things: Voice streaming over NB-IoT. *Pervasive and Mobile Computing* 59 (2019), 101044.

- [40] KAZMAN, R.; KLEIN, M.; CLEMENTS, P. Atom: Method for architecture evaluation. Tech. Rep. CMU/SEI-2000-TR-004, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA, 2000.
- [41] KHAN, I.; BELQASMI, F.; GLITHO, R.; CRESPI, N.; MORROW, M.; POLAKOS, P. Wireless sensor network virtualization: A survey. *IEEE Communications Surveys and Tutorials* 18, 1 (2016), 553–576.
- [42] KHANNA, G.; BEATY, K.; KAR, G.; KOCHUT, A. Application performance management in virtualized server environments. *IEEE Symposium Record on Network Operations and Management Symposium* (2006), 373–381.
- [43] KIM-HUNG, L.; DATTA, S. K.; BONNET, C.; HAMON, F.; BOUDONNE, A. A scalable IoT framework to design logical data flow using virtual sensor. *International Conference on Wireless and Mobile Computing, Networking and Communications 2017-October*, ii (2017).
- [44] LAI, X.; YANG, T.; WANG, Z.; CHEN, P. IoT Implementation of Kalman Filter to Improve Accuracy of Air Quality Monitoring and Prediction. *Appl. Sci.* 9 (2019).
- [45] LANTZ, B.; HELLER, B.; MCKEOWN, N. A network in a laptop. In *Proceedings of the Ninth ACM SIGCOMM Workshop on Hot Topics in Networks - Hotnets '10* (New York, New York, USA, jun 2010), vol. 3, ACM Press, pp. 1–6.
- [46] LEMOS, M.; RABELO, R.; MENDES, D.; CARVALHO, C.; HOLANDA, R. An approach for provisioning virtual sensors in sensor clouds. *International Journal of Network Management* 29, 2 (mar 2019), e2062.
- [47] LEPPÄNEN, T.; SAVAGLIO, C.; FORTINO, G. Service modeling for opportunistic edge computing systems with feature engineering. *Computer Communications* 157, April (2020), 308–319.
- [48] LI, W.; SANTOS, I.; DELICATO, F. C.; PIRES, P. F.; PIRMEZ, L.; WEI, W.; SONG, H.; ZOMAYA, A.; KHAN, S. System modelling and performance evaluation of a three-tier Cloud of Things. *Future Generation Computer Systems* 70 (2017), 104–125.
- [49] MADRIA, S.; KUMAR, V.; DALVI, R. Sensor cloud: A cloud of virtual sensors. *IEEE Software* 31, 2 (2014), 70–77.
- [50] MAHESHWARI, S.; RAYCHAUDHURI, D.; SESKAR, I.; BRONZINO, F. Scalability and performance evaluation of edge cloud systems for latency constrained applications. *Proceedings - 2018 3rd ACM/IEEE Symposium on Edge Computing, SEC 2018* (2018), 286–299.
- [51] MAO, Y.; YOU, C.; ZHANG, J.; HUANG, K.; LETAIEF, K. B. A Survey on Mobile Edge Computing: The Communication Perspective. *IEEE Communications Surveys and Tutorials* 19, 4 (2017), 2322–2358.
- [52] MASSE, M. *REST API Design Rulebook: Designing Consistent RESTful Web Service Interfaces*. " O'Reilly Media, Inc.", 2011.

- [53] MORABITO, R. Virtualization on internet of things edge devices with container technologies: A performance evaluation. *IEEE Access* 5 (2017), 8835–8850.
- [54] MORABITO, R.; COZZOLINO, V.; DING, A. Y.; BEIJAR, N.; OTT, J. Consolidate IoT Edge Computing with Lightweight Virtualization. *IEEE Network* 32, 1 (2018), 102–111.
- [55] MORRIS, I. ETSI Drops 'Mobile' From MEC, 2016. [https://www.lightreading.com/mobile/mec-\(mobile-edge-computing\)/etsi-drops-mobile-from-mec/d/d-id/726273](https://www.lightreading.com/mobile/mec-(mobile-edge-computing)/etsi-drops-mobile-from-mec/d/d-id/726273), Last accessed on 2020-02-22.
- [56] MOURADIAN, C.; EBRAHIMNEZHAD, F.; JEBBAR, Y.; AHLUWALIA, J. K.; AFRASIABI, S. N.; GLITHO, R. H.; MOGHE, A. An IoT Platform-as-a-Service for NFV Based-Hybrid Cloud/Fog Systems. *IEEE Internet of Things Journal* 4662, c (2020), 1–1.
- [57] MOURADIAN, C.; NABOULSI, D.; YANGUI, S.; GLITHO, R. H.; MORROW, M. J.; POLAKOS, P. A. A Comprehensive Survey on Fog Computing: State-of-the-Art and Research Challenges. *IEEE Communications Surveys and Tutorials* 20, 1 (2018), 416–464.
- [58] MUKHERJEE, M.; SHU, L.; WANG, D. Survey of fog computing: Fundamental, network applications, and research challenges. *IEEE Comm Surveys and Tutorials* 20, 3 (2018), 1826–1857.
- [59] NAUMAN, A.; QADRI, Y. A.; AMJAD, M.; ZIKRIA, Y. B.; AFZAL, M. K.; KIM, S. W. Multimedia internet of things: A comprehensive survey. *IEEE Access* 8 (2020), 8202–8250.
- [60] NETO, A. J.; ZHAO, Z.; RODRIGUES, J. J.; CAMBOIM, H. B.; BRAUN, T. Fog-based crime-assistance in smart IoT transportation system. *IEEE Access* 6 (2018), 11101–11111.
- [61] PAHL, C.; HELMER, S.; MIORI, L.; SANIN, J.; LEE, B. A container-based edge cloud PaaS architecture based on raspberry Pi clusters. *Proceedings - 2016 4th International Conference on Future Internet of Things and Cloud Workshops, W-FiCloud 2016* (2016), 117–124.
- [62] PANG, H. H.; TAN, K. L. Authenticating query results in edge computing. *Proceedings - International Conference on Data Engineering* 20 (2004), 560–571.
- [63] PANIAGUA, C.; ELIASSON, J.; DELSING, J. Efficient Device-to-Device Service Invocation Using Arrowhead Orchestration. *IEEE Internet of Things Journal* 7, 1 (2020), 429–439.
- [64] PASRICHA, S.; AYOUB, R.; KISHINEVSKY, M.; MANDAL, S. K.; OGRAS, U. Y. A Survey on Energy Management for Mobile and IoT Devices. *IEEE Design and Test* 2356, c (2020), 1–15.
- [65] PATEL, M.; HU, Y.; HÉDÉ, P.; JOUBERT, J.; THORNTON, C.; NAUGHTON, B.; JULIAN, R. R.; CHAN, C.; YOUNG, V.; TAN, S. J.; LYNCH, D. Mobile Edge Computing – Introductory Technical White Paper. *ETSI White Paper* 11, 1 (2014), 1–36.

- [66] PECHOTO, M. M.; UEYAMA, J.; DE ALBUQUERQUE, J. P. E-noé : Rede de sensores sem fio para monitorar rios urbanos. *Congresso Brasileiro Sobre Desastres Naturais* (2012).
- [67] PEUSTER, M.; KARL, H.; VAN ROSSEM, S. Medicine: Rapid prototyping of production-ready network services in multi-pop environments. In *2016 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN)* (Nov 2016), pp. 148–153.
- [68] RAZZAQUE, M. A.; MILOJEVIC-JEVRIĆ, M.; PALADE, A.; CLA, S. Middleware for internet of things: A survey. *IEEE Internet of Things Journal* 3, 1 (2016), 70–95.
- [69] RODRIGUES, J. J. P.; SENDRA COMPTE, S.; DE LA TORRA DIEZ, I. Body Area Networks. *e-Health Systems* (2016), 97–121.
- [70] SANTOS, F.; GUERRA, R. OSIRIS Framework: construindo sistemas de monitoramento com redes de sensores sem fio para compartilhar dados. *SBRC* (2015).
- [71] SANTOS, I. L.; PIRMEZ, L.; DELICATO, F. C.; KHAN, S. U.; ZOMAYA, A. Y. Olympus: The cloud of sensors. *IEEE Cloud Computing* 2, 2 (2015), 48–56.
- [72] SANTOS, I. L. D.; DELICATO, F. C.; PIRES, P. F.; ALVES, M. P.; OLIVEIRA, A.; CALMON, T. S. Data-Centric Resource Management in Edge-Cloud Systems for the IoT. *Open Journal of Internet Of Things (OJIOT)* 5, 1 (2019), 29–46.
- [73] SARAČEVIĆ, M. H.; ŠABANOVIĆ, M.; AZIZOVIĆ, E. Comparative analysis of AMF, JSON and XML technologies for data transfer between the server and the client. *Periodicals of Engineering and Natural Sciences (PEN)* 4, 2 (2016), 257–261.
- [74] SATYANARAYANAN, M. The emergence of edge computing. *Computer* 50, 1 (2017), 30–39.
- [75] SATYANARAYANAN, M.; BAHL, P.; CACERES, R.; DAVIES, N. The Case for VM-Based Cloudlets in Mobile Computing. *IEEE Pervasive Computing* 8, 4 (oct 2009), 14–23.
- [76] SCHULZRINNE, H.; RAO, A.; LANPHIER, R. Rfc2326: Real time streaming protocol (rtsp), 1998.
- [77] SERALATHAN, Y.; OH, T. T.; JADHAV, S.; MYERS, J.; JEONG, J. P.; KIM, Y. H.; KIM, J. N. IoT security vulnerability: A case study of a Web camera. *International Conference on Advanced Communication Technology, ICACT 2018-February* (2018), 172–177.
- [78] SHELBY, Z.; HARTKE, K.; BORMANN, C. The Constrained Application Protocol (CoAP). RFC 7252, June 2014.
- [79] SHI, W.; DUSTDAR, S. The Promise of Edge Computing. *Computer* 49, 5 (may 2016), 78–81.
- [80] SOLOPOVS, S. HOW WE TEST MOBILE APPLICATION PERFORMANCE AS A THIRD PARTY, 2018. <https://www.testdevlab.com/blog/2018/12/how-we-test-mobile-application-performance-as-a-third-party/>, Last accessed on 2020-06-24.

- [81] TAIVALSAARI, A.; MIKKONEN, T. Gateways to Heaven. In *Proceedings of the 17th International Conference on Advances in Mobile Computing & Multimedia* (New York, NY, USA, dec 2019), ACM, pp. 219–225.
- [82] THANGAVEL, D.; MA, X.; VALERA, A.; TAN, H. X.; TAN, C. K. Y. Performance evaluation of MQTT and CoAP via a common middleware. *IEEE ISSNIP 2014 - 2014 IEEE 9th International Conference on Intelligent Sensors, Sensor Networks and Information Processing, Conference Proceedings*, April (2014), 21–24.
- [83] WANG, Q.; ZHAO, Y.; WANG, W.; MINOLI, D.; SOHRABY, K.; ZHU, H.; OCCHIOGROSSO, B. Multimedia IoT systems and applications. *GIoT S 2017 - Global Internet of Things Summit, Proceedings*, 2 (2017).
- [84] WANG, S.; GAO, J. Z.; LI, W.; LI, Y.; WANG, K.; LU, S. Building smart city drone for graffiti detection and clean-up. *Proceedings - 2019 IEEE SmartWorld, Ubiquitous Intelligence and Computing, Advanced and Trusted Computing, Scalable Computing and Communications, Internet of People and Smart City Innovation, SmartWorld/UIC/ATC/SCALCOM/IOP/SCI 2019* (2019), 1922–1928.
- [85] WEI, X.; WU, L. A new proposed sensor cloud architecture based on fog computing for internet of things. *Proceedings - 2019 IEEE International Congress on Cybermatics* (2019), 615–620.
- [86] XAVIER, B.; FERRETO, T.; JERSAK, L. Time Provisioning Evaluation of KVM, Docker and Unikernels in a Cloud Platform. *Proceedings - 2016 16th IEEE/ACM International Symposium on Cluster, Cloud, and Grid Computing, CCGrid 2016* (2016), 277–280.
- [87] XIE, Y.; HU, Y.; CHEN, Y.; LIU, Y.; SHOU, G. A Video Analytics-Based Intelligent Indoor Positioning System Using Edge Computing For IoT. In *2018 International Conference on Cyber-Enabled Distributed Computing and Knowledge Discovery (CyberC)* (oct 2018), IEEE, pp. 118–1187.
- [88] XU, J.; ANDREPOULOS, Y.; XIAO, Y.; VAN DER SCHAAR, M. Non-stationary resource allocation policies for delay-constrained video streaming: Application to video over internet-of-things-enabled networks. *IEEE Journal on Selected Areas in Communications* 32, 4 (2014), 782–794.
- [89] XU, J.; PALANISAMY, B.; LUDWIG, H.; WANG, Q. Zenith: Utility-Aware Resource Allocation for Edge Computing. *Proceedings - 2017 IEEE 1st International Conference on Edge Computing, EDGE 2017* (2017), 47–54.
- [90] YANG, B.; CHAI, W. K.; PAVLOU, G.; KATSAROS, K. V. Seamless Support of Low Latency Mobile Applications with NFV-Enabled Mobile Edge-Cloud. *Proceedings - 2016 5th IEEE International Conference on Cloud Networking, CloudNet 2016* (2016), 136–141.
- [91] YU, W.; LIANG, F.; HE, X.; HATCHER, W. G.; LU, C.; LIN, J.; YANG, X. A Survey on the Edge Computing for the Internet of Things. *IEEE Access* 6, c (2017), 6900–6919.

-
- [92] ZHANG, C.; CHANG, E. C. Processing of mixed-sensitivity video surveillance streams on hybrid clouds. *IEEE International Conference on Cloud Computing, CLOUD* (2014), 9–16.
- [93] ZHANG, J.; LI, Z.; SANDOVAL, O.; XIN, N.; REN, Y.; MARTIN, R. A.; IANNUCCI, B.; GRISS, M.; ROSENBERG, S.; CAO, J.; ROWE, A. Supporting personizable virtual internet of things. *Proceedings - IEEE 10th International Conference on Ubiquitous Intelligence and Computing, UIC 2013 and IEEE 10th International Conference on Autonomic and Trusted Computing, ATC 2013* (2013), 329–336.
- [94] ZIKRIA, Y. B.; AFZAL, M. K.; KIM, S. W. Internet of Multimedia Things (IoMT): Opportunities, Challenges and Solutions. *Sensors (Basel, Switzerland)* 20, 8 (2020), 1–8.