Consolidate IoT Edge Computing with Lightweight Virtualization

Roberto Morabito, Vittorio Cozzolino, Aaron Yi Ding, Nicklas Beijar, and Jörg Ott

***Abstract*—Lightweight Virtualization (LV) technologies have refashioned the world of software development by introducing flexibility and new ways of managing and distributing software. Edge computing complements today's powerful centralized data centers with a large number of distributed nodes that provide virtualization close to the data source and end users. This emerging paradigm offers ubiquitous processing capabilities on a wide range of heterogeneous hardware characterized by different processing power and energy availability. The scope of this article is to present an in-depth analysis on the requirements of edge computing in the perspective of three selected use cases particularly interesting for harnessing the power of the Internet of Things (IoT). We discuss and compare the applicability of two LV technologies, containers and unikernels, as platforms for enabling scalability, security and manageability required by such pervasive applications that soon may be part of our everyday life. To inspire further research, we identify open problems and highlight future directions to serve as a road map for both industry and academia.**

***Index Terms*—IoT, Edge Computing, Container, Unikernel**

I. Introduction

OVER the last decade, the development of the Internet of Things (IoT) has been uphold by the cloud-based infrastructures that aim to cope with the increasing number of IoT services provided by various connected devices. From the initial design, IoT was conceived as extending the Internet with a new class of devices and use cases [1]. This has obviously generated an intrinsic association between IoT and cloud, where the cloud-based network infrastructures are optimized to support a multitude of IoT-centric operations such as service management, computation offloading, data storage, and offline analysis of data.

However, this notion of cloud-connected IoT deployment assumes that most IoT edge networks need to be connected to the cloud, e.g., through some edge gateway and tunnel approach. This centralized model has been challenged recently for meeting the more and more stringent performance requirements of IoT services, especially in terms of latency and bandwidth. In specific, the existing model is not suitable when: a) IoT edge networks create data that needs to be accessed and processed locally, b) piping everything to the cloud and back is not acceptable under delay constraints, and c) the amount of data is too large to transfer to the cloud (in real-time) without causing congestion on the backhaul. Clearly, the highly fragmented and heterogeneous IoT landscape needs to encompass novel and reactive approaches for dealing with these challenges.

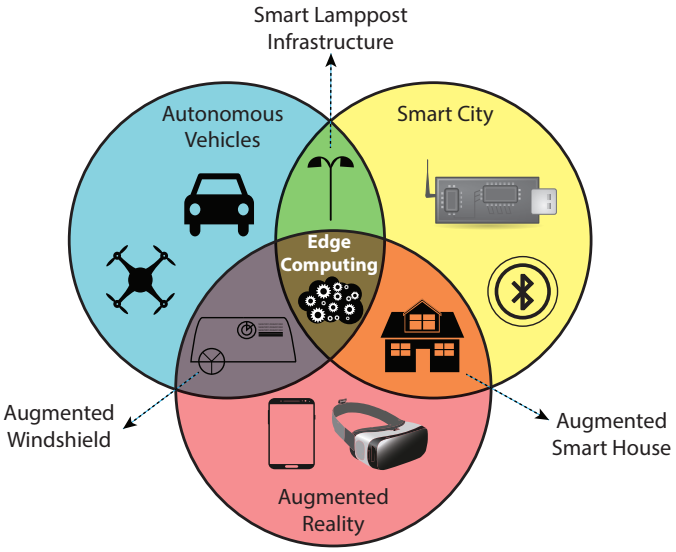


Figure 1: A subset of use cases and services enabled by IoT edge computing

One emerging paradigm, edge computing, represents a new trend to improve the overall infrastructure efficiency by delivering low-latency, bandwidth-efficient and resilient services to IoT users. Although this new approach is not intended to replace the cloud-based infrastructure, it expands the cloud by increasing the computing and storage resources available at the network edge. One typical example is IoT edge offloading [2], which revisits the conventional cloud-based computation offloading where mobile devices resort to resourceful servers to handle heavy computation [3]. To cater for the demands of new IoT services, the computation is reversely dispatched by the servers to constrained devices deployed at the network edge, close to users and data generators.

By harnessing the power of distributed edge resources, the IoT edge computing model can support novel service scenarios such as, for example, autonomous vehicles/drones, smart cities infrastructure and augmented reality (AR). As highlighted in Figure 1, these three representative domains intersect with each other. Edge computing is the linking knot that helps spawn and promote appealing joint services.

Concerning the key aspects of edge computing including scalability, multi-tenancy, security, privacy and flexibility, the fast evolving lightweight virtualization technologies (discussed

R. Morabito, N. Beijar are with Ericsson, Jorvas, Finland. E-Mail: roberto.morabito@ericsson.com, nicklas.beijar@ericsson.com

V. Cozzolino, A. Y. Ding and J. Ott are with Technical University of Munich, Munich, Germany. E-Mail: cozzolin@in.tum.de, ding@in.tum.de, ott@in.tum.de

R. Morabito and V. Cozzolino made equal contributions to this article.

in Section IV) have been sorted to fulfill the requirements given their matching features. Meanwhile, we still lack comprehensive guidelines to illustrate how can we exploit the full potential of lightweight virtualization to enhance edge computing, especially for those pleading IoT use cases.

As a solid step towards realizing the IoT edge computing vision, we aim to answer through this article a major question: *Can Lightweight Virtualization (LV), in its different flavors, be exploited for empowering edge architectures and be suitable in a wide range of IoT pervasive environments?* Our use-case study, comparison analysis, and prospect outlook further address the following questions:

- Which LV features can match the increasingly strict requirements of IoT services in constrained environments?
- How can LV and IoT edge scenarios be efficiently utilized together?
- Which challenges must be tackled to effectively exploit the benefits introduced by LV in this context?

The remainder of this article is organized as follows. Motivations of the proposed work are presented in Section II. Section III introduces first the requirements that different *Edge for IoT* cases entail, and then the suitability of LV on mitigating and satisfying them. We introduce LV technologies and illustrate three specific use cases in Section IV and V. Finally, we unveil the open issues and challenges before concluding the article.

II. Motivation

In the context of IoT, edge computing introduces an intermediate layer in the conventional IoT-Cloud computing model. The envisioned edge-driven IoT environment consists of three components: IoT devices, edge layer, and cloud backend. Being a central part of the ecosystem, the 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. Depending on the specific scenario, its functionalities can be executed in cellular base stations, IoT gateways, or more generally, low-power nodes and small data-centers. These may be owned and operated by the user, by a cloud provider or a telecom operator (in Mobile Edge Computing).

Although the placement of a “middle layer" between the end devices and cloud is an architectural concept that is widely utilized in common network infrastructures, such conventional middle layer targets mainly connectivity, routing, and network-oriented functionalities. For example, Network Function Virtualization (NFV) [4], [5] virtualizes typical network elements, such as firewalls, network address translators, switches, and core network components.

For IoT ecosystems, edge computing aims to meet IoT service providers' demand of owning a dedicated infrastructure that is independent of a given technology or use case, and which is capable of satisfy the demanding IoT services' performance requirements. More importantly, in contrary to the plain middle layer solutions, the IoT-centric edge computing must entail programmability and flexibility to deliver ubiquitous processing capabilities across a wide range of heterogeneous hardware. For instance, besides managing IoT home network, the edge layer can simultaneously provide image processing for home camera and data pre-processing operations.

Obviously, the heterogeneous characteristics of various instances and applications deployed on top of the edge layer will generate unique challenges that need to be addressed. From the architectural perspective, this implies that edge layer has to efficiently and mutually cooperate both with cloud-based services and IoT devices, by acting as a bridge between elements that require distinct way of interaction.

In this context, it is crucial to equip the edge layer with tools that allow a flexible, performing, and automated way of efficient services provisioning. Hence, edge elements have to embed service provisioning methods that are independent of the managed applications and communication patterns, and at the same time suitable to different types of traffic and to the application needs, through a cross-layer support. The key is to ensure a virtuous trade-off between design requirements, specific performance targets, and applications manageability spanning the entire three-tier IoT edge computing architecture.

III. Empowering IoT Edge Computing with LV

To fully attain the potential of edge computing for IoT, we need to address four concerns: abstraction, programmability interoperability, and elasticity. In particular for the three-tier IoT edge computing architecture, it is crucial to provide simple and yet efficient configuration and instantiation methods that are independent of the technologies used by different IoT and cloud providers. The tools embedded in edge layer should share common functionalities, exploit common APIs for orchestrating interconnections different networking technologies.

To help us acquire a synoptic view, we highlight the dominant requirements of representative use cases in Figure 2, which encompasses scalability, multi-tenancy, privacy & security, latency, and extensibility.

Compared to alternative virtualization solutions such as hypervisors, we envision a trend towards using lightweight virtualization (LV) technologies in the IoT edge computing. These emerging software solutions can provide the needed supports in terms of hardware abstraction, programmability interoperability, and elasticity. A direct benefit that emerges from employing LV in the IoT edge domain is by avoiding the strict dependency on any given technology or use case. Within a lightweight virtualized instance, either container or unikernel (discussed in Section IV), we can efficiently deploy applications designed to manage and use extremely different technologies. In addition, equipping edge elements with newer services will be made easier since we only need to configure and instantiate stand-alone virtualized applications. This feature avoids complex re-programming and updating operations that are part of the software lifecycle management. Through LV, such complexity is circumvented because updating a particular service requires changes only within a specific virtualized instance.

Scenarios	Requirements				
	Scalability	Multi-tenancy	Privacy & Security	Latency	Extensibility (Open API)
Autonomus Vehicles	*Non critical*	*Non critical*	*Critical. Autonomous vehicles possess sensitive information about the user. Moreover, the constant need of sensors data for navigation make cars a primary target for malicious users.*	*Critical. Cars have strict real-time requirements*	*Non critical. Each car manufacturer will probably run exclusively their own software to ensure security and reliability.*
Augmented Reality	*Critical*	*Critical*	*Critical when processing sensitive multimedia streams.*	*Critical. AR applications require real-time information feed to ensure a smooth and acceptable experience.*	*Critical. Open API are important in this case to enable new services and features.*
Smart Sensors Networks	*Critical*	*Critical due to the number of potential users.*	*Depends on the specific Smart context (for Smart Health it's critical but not for Smart Environment). Strict control over which data can be public is required.*	*Depends. For example, in the case of Machine Type Communications (MTC) it's critical.*	*Critical to enable the creation of an 'IoT Marketplace' where developers can offer new and innovative application exploiting collected data.*
Smart Grid	*Non critical. Data and messages are exchanged at a fixed, predefined rate.*	*Non critical. The infrastructure is usually controlled by a single provider.*	*Critical. Disclosure and analysis of energy consumption information can lead to user profiling and tracking.*	*Critical especially for messages as Phasor Measurement Unit (PMU) or Advanced Metering Infrastructure (AMI).*	*Non critical*
E-Health	*Critical. IoT healthcare networks must be able to meet the growing demand of services from both individuals and health organizations.*	*Critical as multiple healthcare organizations and/or heterogeneous IoT medical devices could share the same network infrastructure.*	*Critical. IoT-edge medical devices deal with personal heath data, which need to be securely stored. Integrity, privacy, and confidentiality must be kept.*	*Depends. It's critical in use-cases as remote surgery. Nevertheless, response time can be acceptable in other scenarios.*	*Critical to support new application able to offer a more accurate patients health condition monitoring.*
Distributed Surveillance	*Critical. Several control units are needed in order to grant the system of better usability and robustness.*	*Non-critical. A single provider usually controls the infrastructure.*	*Critical considering the sensitive information handled.*	*Critical to promptly identify suspects or recognize on-going crimes.*	*Non-critical. Same as Autonomous vehicles.*
Big Data Analytics	*Critical. A big data analytics system must be able to support very large datasets. All the components must be scalable to accommodate the constantly growing amount of data to be handled.*	*Critical. A single Big Data system has to be able to co-locate different use cases, applications, or data sets.*	*Critical. Users share large amount of personal data and sensitive content through their personal devices towards applications (e.g., social networks) and public clouds. Equipping Big Data systems of secure frameworks capable to store and manage user data with high sensitiveness represents a critical aspect.*	*Non critical*	*Critical to improve and deploy different algorithms and tools.*
Network Function Virtualization (NFV)	*Critical. Demand of new services is high and constantly growing.*	*Critical. Resources are shared among customers. A large number of multi-tenant networks run over a physical network.*	*Critical. The use of additional software (e.g., hypervisors, containers or unikernels) extends the chain of trust. Resource pooling and multi-tenancy bring further security/privacy threats.*	*Critical. NFV need to leverage real-time delivery services. NFV introduces additional sources of latency through the virtualization layer.*	*Non critical*

Figure 2: Example of Edge-IoT scenarios requirements

To foster integration with the cloud, LV can also enable cross-platform deployment, allowing a common execution environment across cloud, edge elements, and even constrained IoT devices. The cross-platform deployment benefit introduced by LV further allows both cloud and edge, regardless of their computational hardware capability, to "speak the same language". As suggested in [2], using the same LV instance will enable us to efficiently run them both at the edge and in the cloud, hence achieving a decentralized IoT edge service provisioning architecture. This consequently meets the strict performance requirements of demanding IoT scenarios, and further ensures the crucial requirement of multi-tenancy.

We also note that there are scenarios where virtualization technology is not a suitable option, for manifold reasons. In general, virtualization entails additional delay and resources utilization, which can be challenging for certain real-time or mission-critical tasks that demand low and predictable latency. Moreover, there are fundamental hardware requirements to run a virtualized environment (e.g. a CPU with specific architectural features) that are not easily found on low-end IoT and edge devices.

IV. Overview of Lightweight Virtualization

System virtualization has drastically evolved in the last years offering system architects and developers a plethora of tools to exploit. Therefore, understanding how and when to utilize a specific technology based on the hardware constraints and applicative requirements is a crucial step of the system design phase. Shifting our focus on edge computing and IoT, we identify two main candidates that could address the challenges unique to this domain: containers and unikernels.

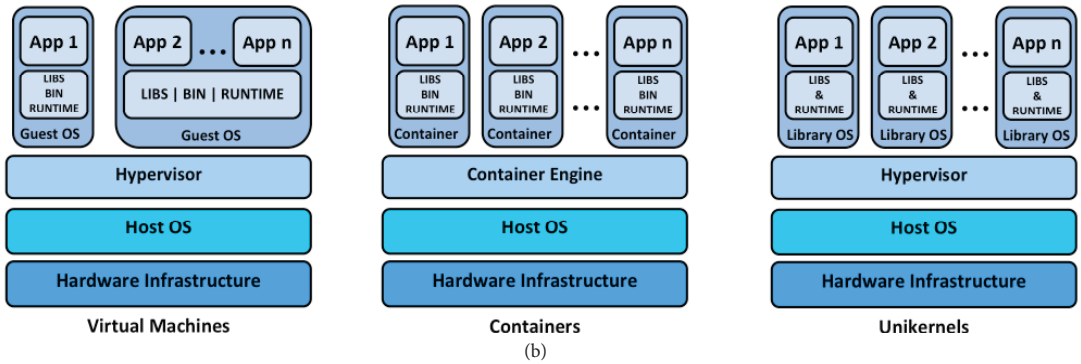
Figure 3 presents both quantitative metrics and architectural differences between the aforementioned technologies, highlighting their main characteristics.

A. Container-based Virtualization: Docker

Container-based virtualization provides a different level of abstraction in terms of virtualization and isolation when compared to other virtualization solutions. In particular, containers can be considered as one of the lightweight alternatives to hypervisor-based virtualization. The conventional hypervisor-based virtualization has been the *de facto* technology used during the last decade for implementing server virtualization and isolation. Hypervisors operate at the hardware level — that is, building customizable virtual hardware and virtual device drivers — thus supporting standalone Virtual Machines (VMs) that are independent and isolated from the underlying host system. In each VM instance, a full Operating System (OS) is typically installed on top of the virtualized hardware, thus generating large VMs images. Furthermore, the emulation of

LV Technique	Property					
	Instantiation time	Image size	Memory footprint	Programming language dependency	Hardware portability	Live migration support
Virtual Machine • KVM • QEMU	*~5/10 secs*	*~1000 MBs*	*~100 MBs*	*No*	*High*	*Yes*
Container • Docker (http://www.docker.com/) • rkt (https://coreos.com/rkt) • OpenVZ https://openvz.org/ • LXC https://linuxcontainers.org/	*~800/1000 msecs*	*~50 MBs*	*~5 MBs*	*No*	*High*	*No*
Unikernel • MirageOS (https://mirage.io/) • HaLVM (http://galois.com/project/) • IncludeOS (www.includeos.org) • ClickOS (http://cnp.neclab.eu/clickos/) • OSv (osv.io)	*~< 50 msecs*	*~< 5MBs (bundle)*	*~8 MBs*	*Yes (i.e., MirageOS unikernels can only be written in OCaml)*	*High*	*No. Requires manual implementation*

(a)



(b)

Figure 3: LV techniques comparison. (a) Quantitative analysis; (b) Core architectural differences

virtual hardware devices and related drivers produces non-negligible performance overhead.

Differently, containers implement processes isolation at the OS level, thus avoiding the virtualization of hardware and drivers [6]. In specific, containers share the same OS kernel with the underlying host machine, meanwhile making it possible to isolate stand-alone applications that own independent characteristics, i.e., independent virtual network interfaces, independent process space, and separate file systems. This shared kernel feature allows containers to achieve a higher density of virtualized instances on a single machine thanks to the reduced image volume.

Containers have achieved much more relevance and practical use recently with the advent of Docker, a high-level platform that has made containers very popular in a short time frame. Docker introduces an underlying *container engine*, together with a practical and versatile API, which allows easily building, running, managing, and removing containerized applications. A *Docker container*, which is a runnable instance of *Docker image*, uses a base image stored in specific private or public registries. Docker uses an overlay file-system (*UnionFS*) to add a read-write layer on top of the image. UnionFS allows to store Docker images as a series of layers and consequently saving disk space. In fact, the different image layers can be cached in the disk allowing to speed up the building process, and re-use the same cached layer for the building of different images.

The lightweight features embedded in containers ease the integration of such technology in various networking fields. In specific to IoT edge computing, containers can enable us to efficiently run containerized applications even in devices characterized by lower processing capabilities, such as Single-Board Computers [7].

B. Library Operating Systems: Unikernels

Unikernels are single-purpose appliances that are at compile time specialized into standalone kernels [8], and sealed against modification after deployment. The concept of unikernels has emerged from the observation that most applications running in the cloud do not require many of the services coming with common operating systems. Additionally, unikernels provide increased security through a reduced attack surface and better performance by dropping unnecessary components from the applications.

Unikernels were designed initially with the cloud in mind but their small footprint and flexibility make them fit also well with the upcoming IoT edge ecosystem as illustrated through different research attempts [9]-[2]. The main differences among existing unikernel implementations sprout from the underlying programming language in use. MirageOS [8] and HaLVM are unikernels based on functional languages with pervasive type-safety in the running code. Other solutions like IncludeOS and ClickOS are C++ unikernels; the former offering a C++ platform to which bind generic applications, while the latter is highly specializing in offering dynamic network processing (based on Click modular router). OSv is based on Java and therefore heavier than the others, but more flexible.

Security and unikernels are tightly coupled. The attack surface of a unikernel is strictly confined to the application

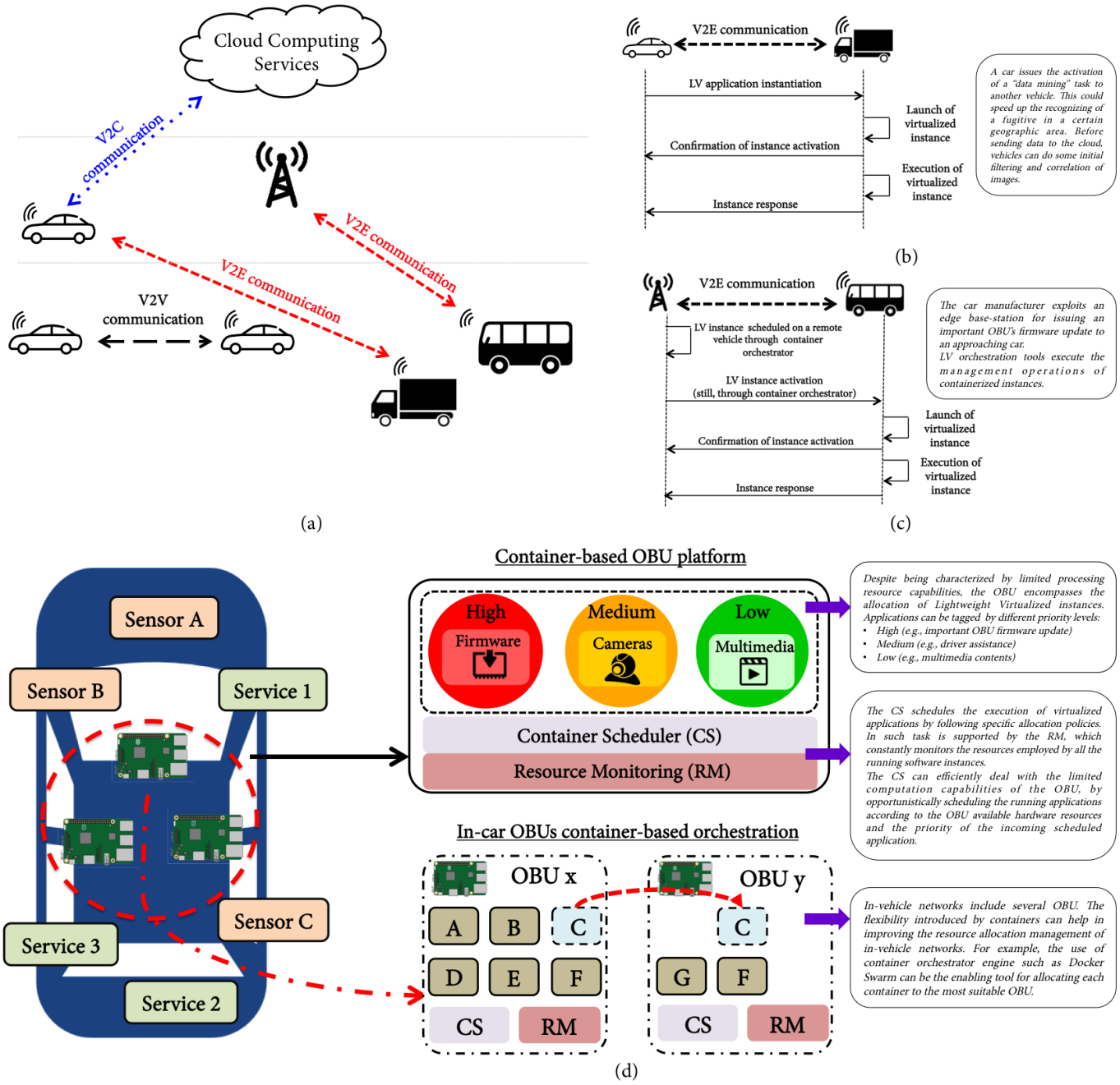


Figure 4: a) Vehicular Edge Computing scenario in its entirety. Vehicular-to-Edge (V2E) interactions examples: b) Car-to-Car V2E communication, c) and Base Station-to-Car V2E communication; d) Container-based virtualization is used for an easier OBU's customization. Furthermore, within the same vehicle orchestration tools are exploited for task offloading among different OBUs.

embedded within. There is no uniform operating layer in a unikernel, and everything is directly compiled into the application layer. Therefore, each unikernel may have a different set of vulnerabilities, which implies that an exploit that can penetrate one may not be threatening the others. Unikernels are principally designed to be stateless. Therefore, they are perfect to embed generally algorithms (e.g. compression, encryption, data aggregation functions) or NFV.

V. Use-case Scenarios

In this section, we present three use cases matching the scenarios presented in Figure 1. Additionally, we illustrate the reasons for adopting a specific LV technology for each case.

A. Towards the Vehicular Edge Computing

The importance of virtualization in vehicular scenarios has been widely acknowledged in the past. Vehicular Cloud-Computing (VCC) represents an efficient architectural model in supporting the *Internet of Vehicles* (IoV) [10]. However, we

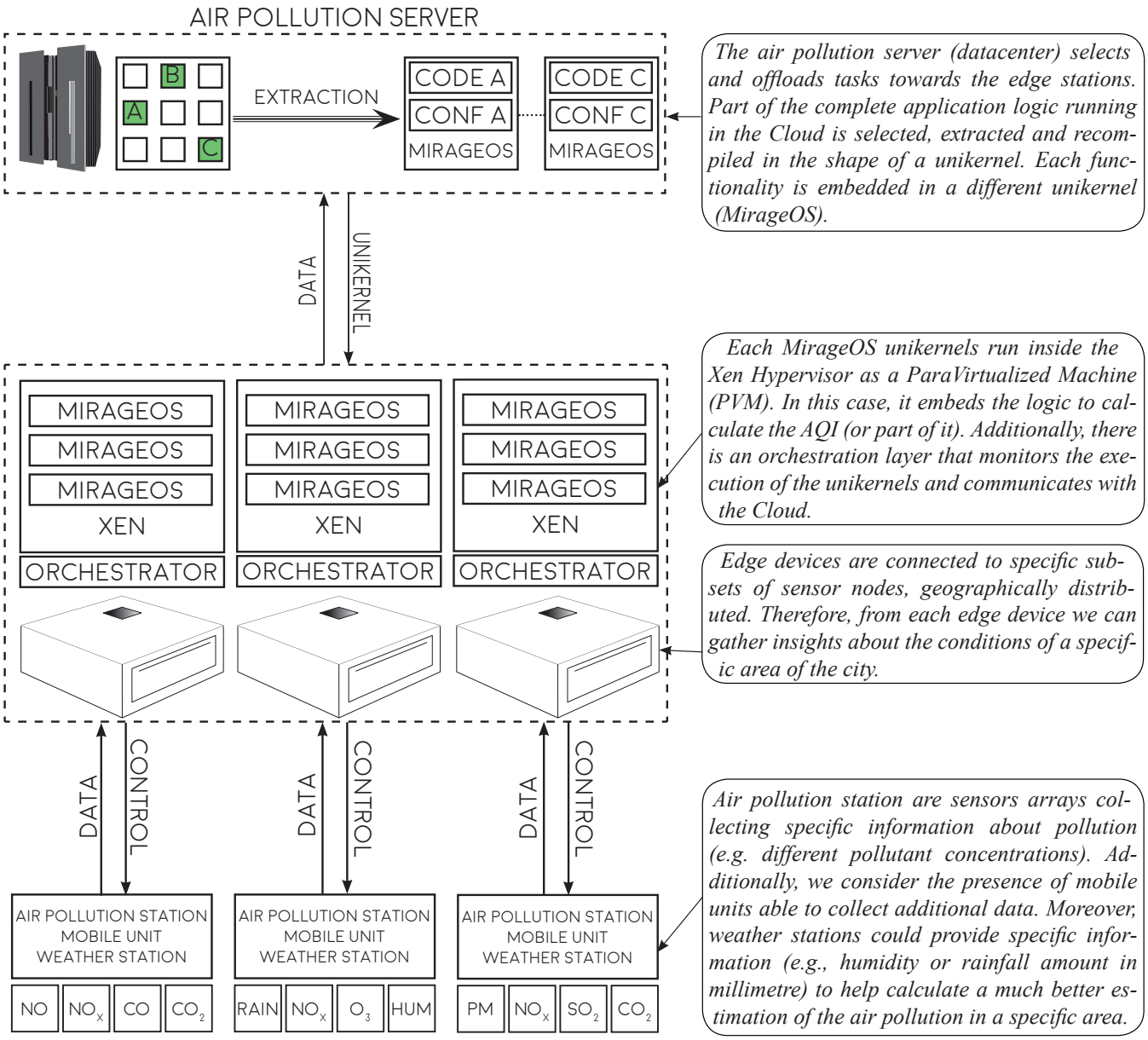


Figure 5: Air pollution scenario. An offloadable task is Air Quality Index (AQI) calculation, a number used by government agencies to communicate to the public how polluted the air currently is or how polluted it is forecast to become. Calculation of the AQI can be executed locally by edge nodes enhancing real-time monitoring.

envision the need to establish a Vehicular Edge Computing (VEC) paradigm, which will play a crucial role in future development of more efficient Vehicle-to-Everything (V2X) systems. VEC can cope with the increasingly strict requirements of V2X applications, and will rely on the growing processing capabilities that the different actors of IoV encompass, including cars' On-Board Unit (OBU), Edge Elements (EEs), Cloud Services. In VEC environments, various units can play the role of EE. Base stations, IoT gateways, and other vehicles themselves can operate as EE by executing specific tasks e.g., lightweight data mining operations, generic off-loading processing, dashcam images filtering, etc. In such context, LV can enable the VEC paradigm, and be exploited in multiple scenarios, spanning from an efficient and flexible customization of cars' OBU to Vehicular-to-Edge (V2E) interactions.

Figure 4 depicts the VEC scenario in its entirety (Fig. 4a), together with practical examples of the way in which LV can be employed in V2E Interactions (Fig. 4b-c) and distributed In-Car Platforms (Fig. 4d).

V2E Interactions. Differently from already well-established Vehicular-to-Vehicular (V2V) communication, V2E aims to encompass computation offloading, tasks outsourcing, and software management operations. In practice, LV-enabled OBU can execute a specific task issued by another vehicle or any other EEs, and vice versa, as shown in the two examples shown in Fig. 4b-c.

In-Car Platforms. Container-based virtualization can be used for OBUs customization. It offers high flexibility in the

platform's software management, and allows overcoming the complex software updating procedures required by OBUs [11]. Through conventional VM, car manufacturers can access all CAN (Controller Area Network) bus sensors through OBD and dashcam. However, given OBUs are embedded systems with limited computational resources, LV's lightweight features avoid the performance overhead and allow scaling up/down the running applications according to specific priorities. Furthermore, by taking into consideration that several OBUs can be distributed within a car, virtualization orchestration tools can be used for OBUs' tasks outsourcing — still by following specific OBU resource management policies. More details about the usage of LV in In-Car platforms can be found in Fig. 4d

B. Edge Computing for Smart City

In the context of Smart City, the measurement of environmental data has become an important issue especially for highly crowded metropolis. Currently, air pollution monitoring is achieved with a sparse deployment of stationary, expensive measurement units embedding both sensors[1] and computing units. Air pollution is predicted based on the measured data in combination with complex mathematical models [12]. Since the cost of deploying and maintaining such pollution station are often prohibitive, we envision crowd-sensing as a tangible solution that combines LV and edge computing.

Edge computing offers resources close to the crowd-sensing entities, which can offload through a direct connection their collected data without using a mobile connection. LV allows to offload, distribute and execute part of the required mathematical computation on the EEs without worrying about compatibility issues. For instance, multiple LV images can be created on demand, each one containing only the code necessary to process the data of a single sensor. The partial results will be then subsequently uploaded to a more powerful edge device (e.g. edge data-center) to be merged. Figure 5 provides more details regarding how unikernels can support both the execution of specific algorithms related to air pollution control and provide pre-processing of input data for simulations running in the Cloud.

The described approach can reduce the load on the core network, end-to-end latency and also the cloud (and air pollution stations) provisioning costs. Regarding specific LV technology, we consider unikernels a promising candidate. The algorithms used to assess air pollution levels are generally static and stateless. In other words, they can be considered as black-boxes with a defined range of inputs/outputs. In case of necessity, the algorithm can be simply changed by replacing it with a new unikernel instance without incurring a long network transfer time[2].

C. Augmented Reality

Wearable devices are typically resource-constrained compared to computer hardware of same vintage PC. The core features of a wearable/mobile device are light-weight, comfort, design and battery life. CPU speed, memory and system capabilities are only secondary, contrary to what are required by the PC market. Therefore, it is not surprising that, overall, wearable/mobile devices are not designed to run computationally intensive tasks.

A common approach to solve the problem is offloading AR tasks to cloud services in order to reduce the power consumption on the device and cope with, eventually, insufficient mobile processing. The drawback is that using cloud service will introduce additional latency, which is crucial for real-time applications. This is especially important for AR applications, where responsiveness and user immersion are paramount. Humans are extremely sensitive to delays affecting real-time interactions (e.g., a phone call). Different studies revealed the speed at which the human brain can identify faces in a dark scene and the requirements of virtual reality application to achieve perpetual stability [13]-[14]. Longer delays in such highly interactive and multimedia-based applications will lower the end-users' experience.

An use cases where there is a strong interplay between local computational resources and AR (or, broadly speaking, computer vision) is augmented windshields for autonomous vehicles. The driver, at this point, passive, might shift is attention completely on the windscreen instead of checking the console in search for speed information. Additionally, the windshield will also provide traffic conditions information, personal agenda, news feed, gaming interfaces, social networks and so forth. In order to craft and manage such a visually-rich experience, an edge board mounted on the car is considered necessary.

Therefore, with the support of edge computing and LV, we have the possibility to offload expensive image processing tasks to EEs in proximity instead of resorting to the cloud back ends. Therefore, we can limit the latency impact, assuming that the computation time is *device-invariant*. The use of virtualization in such context is additionally motivated by the following factors: *multi-tenancy* (i.e., multiple users executing multiple tasks) and *tasks isolation* for privacy.

For this specific use case, a combination of Docker and Unikernel represents a potential approach, as shown in Figure 6. A Docker image containing multiple unikernel can be composed and shipped, each one representing a different AR stage/task. Therefore, the Docker image can offer the orchestration and control API to external applications while, under the hood, unikernels would take care of running the required computations.

VI. Open Issues and Challenges

In this section, we discuss the technical challenges for integrating LV into IoT edge computing and further identify open directions for future research.

A. Orchestration and Monitoring

Orchestration of edge elements (EE) and cloud architectures brings several challenges. Edge-IoT scenarios require specific tools to deal with the different processor architectures and

[1]Usually gas detection sensors (NO, NO_x, O_3, CO, CO_2 and particulate matter) plus humidity, rain detection and wind speed/direction.

[2]Unikernels are, by design, much smaller than other virtualization techniques.

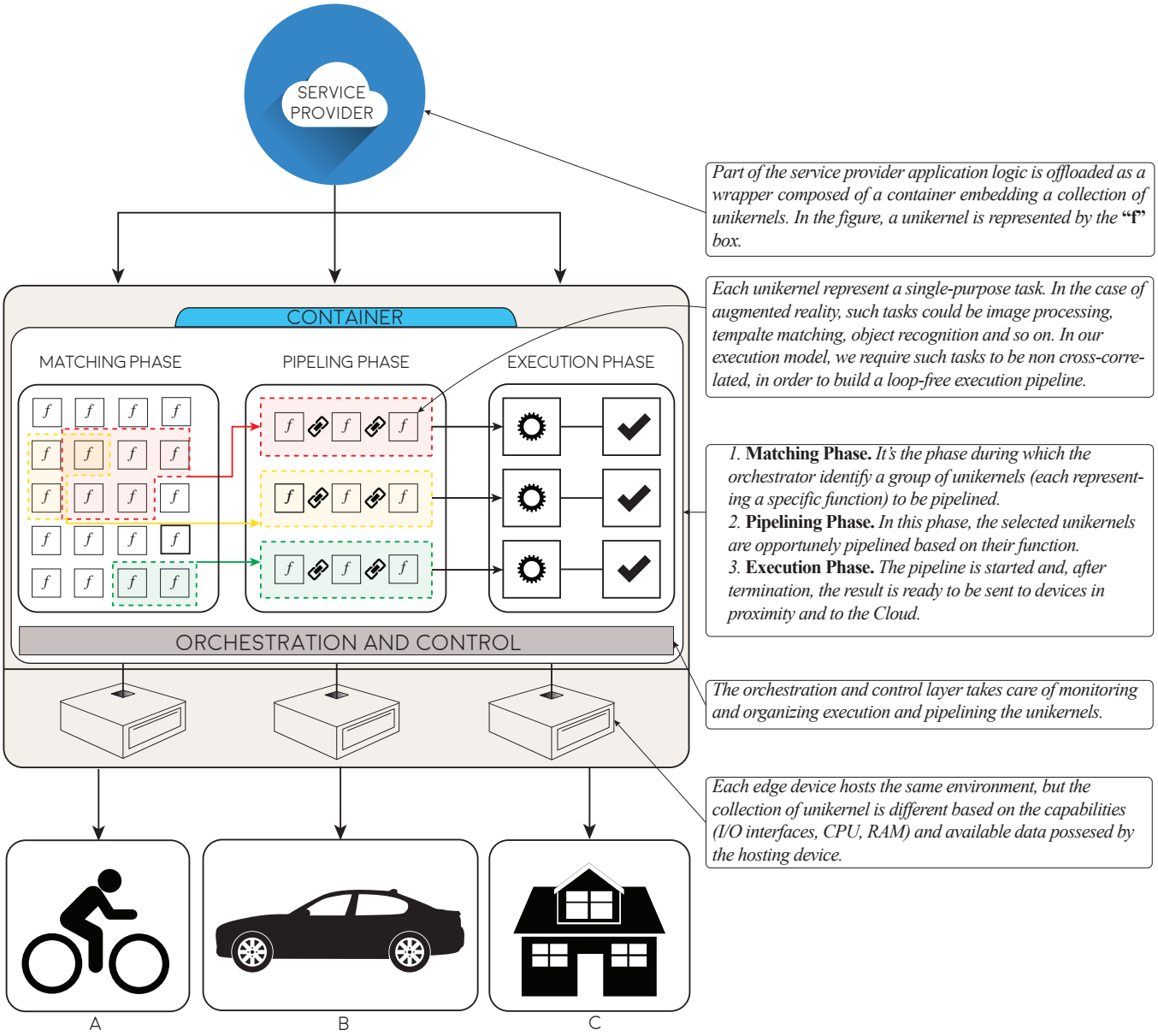


Figure 6: A. Biker receiving personalized advertisements rendered in augmented reality on his smart glasses; B. A smart car populating its augmented windshield with contextualized, live feed information; C. Augmented smart home, where we control IoT devices in proximity through virtual interfaces.

storage capacity of EEs and cloud services. Controlling the network traffic requires to orchestrate cloud and edge, an increasingly challenging task with manifold EEs deployed. Hence, it becomes crucial deploying lightweight orchestration modules that do not overburden the EE, and that seeks a fair balance between synchronization and network load. Other key aspects concern the definition of optimized policies for an efficient *vertical scaling*, in which applications are automatically prioritized and scaled up/down between EE and Cloud, according to specific QoS requirements or computing resources saturation of EEs. Mobility is also a relevant aspect. User devices might move in relation to the edge-processing device providing the service. Therefore, the service may need to be re-deployed multiple times at different locations to serve transparently mobile users. In particular, if the service is specific to an individual user, the number of transfers may be high. Destroying and re-deploying is preferred instead of moving the service together with its running state. For cloud-native service the general recommendation is to avoid storing state locally or only to use disposable state. For services requiring local state, the service must store the current state at an external stable location before exiting and load it again on restart. Particular attention must be paid to ensure that the new edge node has available resources for serving the new device, and the platform may provide alternative nodes or prioritization among services in case of over allocation.

As regards as *monitoring solutions*, both technologies need high-performing, lightweight and scalable monitoring frame-

works. This requirement is strictly related to the fact that these tools may need to run on EEs characterized by lower resource computation capabilities. Another key requirement for monitoring engines is the possibility to track on real-time the individual resources' of each virtualized instance. Implementation of such frameworks becomes, in parallel with orchestration mechanisms, crucial in resource optimization and on developing efficient edge-cloud instances-placement algorithms and policies.

B. Security and Privacy

In the analyzed domain, one challenge is the *certification* of virtualized applications. We need to guarantee their authenticity and validity, by including a signing and validation infrastructure to discriminate *legit* from *tampered* instances. Without such mechanisms, there is the concrete risk of executing malicious code and infringe the security requirements. It is crucial to encourage the development of lightweight security mechanisms, which take into account the strict requirements of IoT applications/scenarios and not impair the lightweight features of the analyzed virtualization technologies, preserving their capacity to not generate performance overhead. From the privacy perspective, EEs may be shared between multiple tenants. It is crucial to be able to isolate tenants' data, but also controlling the use of tenants' dedicated resources — e.g., CPU and memory. Finally, sharing data between tenants at the EE level, without going through the cloud-infrastructure, requires the definition of EE policies and specific access control mechanisms.

C. Standards and Regulations

IoT and EC are developing faster than standards and regulations. The presence of multiple industry partners and researchers working in this field gave birth to different ramifications and interpretations of the same paradigm. Without standards and regulations, merging different approaches will be a non-trivial task exacerbated by the heterogeneity of the involved technologies. For LV technologies, lately there has been a growing effort to lay some guidelines and describe the challenges in the process of building NFV platforms [5]. Nevertheless, this only partially covers the type of functionalities we advocate to offload to EE nodes. Therefore, we consider necessary an additional standardization effort which seeks to lay down precise guidelines towards the employment of LV in a wider range of IoT use-case scenarios.

D. Elasticity in service provisioning

This feature is strictly dependent to the LV engines capacity of quickly allocating/deallocating virtualized instances. Data reported in Table 3a clearly show how both container and unikernel can promptly scale up/down. Furthermore, LV API also allow to *freeze* the execution of an instance and quickly restore it through checkpoint/restore mechanisms. However, there is still a lack of research to evaluate the interactions among multiple EEs, without neglecting that current LV engines implementations not provide fully support for live migration. Specific frameworks that support proactive service migrations for *stateless* applications have been already proposed [15]. However, support for *stateful* applications migration need to be soon integrated for fully exploiting LV benefits in these scenarios.

E. Management Frameworks and Applications portability

Employment of containers technologies have had disruptive rise in the last years, and the enormous effort that open source communities have provided on continually improving fully featured management frameworks has paid off. Unikernels seem to be still not enough mature for being included in production-ready environments, and a greater effort is required for featuring the same *portability* of containers. Packaging applications through unikernel may require an implementation effort that somehow slows down, and in some cases limits, the adaptability towards existing software and hardware platforms. This difference comes from the different way in which the two technologies are built. Containers are application agnostic while unikernels are limited by the programming language and libraries exposed by the underlying minimalistic OS.

F. Data Storage

Containers and, in particular, unikernels are not suitable for storing persistent data, such as data collected from IoT sensors. Moreover, storing important data on edge nodes can be risky both because of the volatile nature of edge nodes and because of the security risks related to easier physical exposure of the nodes. Therefore, data typically needs to be stored in centralized nodes and retrieved on demand. This may reduce the feasibility of LV based edge computation in very data intensive applications. Moreover, some applications requiring nodes to access data of all other nodes data, e.g. for distributed analytics, may be unfeasible to distribute. Automatically optimizing the data storage location of distributed applications is a topic requiring further research. On the other hand, many IoT applications use volatile data locally while persistent data can be minimized and stored centrally.

G. Telco Industry Readiness and Perspectives

The telecommunication sector is currently in a major paradigm shift moving in the direction of *softwarization* of the former hardware based network elements – a concept called NFV [4]. As a first step, the current network functions are directly mapped to corresponding virtualized versions implemented as VMs. The fifth generation (5G) will move toward a more cloud native approach, where different network functions are divided into smaller components that can be individually deployed and scaled and communicate to each other using a message bus. Using MEC as a platform, virtualized network functions (VNFs) can be placed at the edge of the network, and decomposition further encourages the use of LV technologies and the allocation of individual service components to the edge. From the operator perspective, *edge* typically means the base station, but virtualization on Customer Premise Equipment (CPE), such as residential

gateways, may extend the edge further. NFV is the main driver for edge computing in the mobile networks, and a necessity for opening the operator network for third party applications. While operators may have difficulties competing with established players in the cloud market, their presence close to the user make them more competitive for edge-dominated computation. The adoption of LV technologies in telco networks requires a change of mindset in the industry but also technology questions remain for ensuring the reliability and security required for telecommunication networks. As unikernels can be deployed on the same hypervisors as VMs with minor impact on orchestration infrastructure, they are more likely than containers as replacements for VMs.

VII. Outlook

In this article, we examine the challenging problem of integrating LV with IoT edge networks. We first discuss which are the current issues involving EC and IoT network architectures. Therefore, we present three different IoT use-cases, in which LV solutions can bring a set of benefits and a desirable *design flexibility*. Our analysis provides a clear holistic vision of such integration, which promotes innovative network designs to fully exploit the advantages of LV and IoT resources. Finally, we also discuss key technical challenges and identify open questions for future research in this area.

References

[1] A. Zanella, N. Bui, A. Castellani, L. Vangelista, and M. Zorzi, "Internet of Things for Smart Cities," *IEEE Internet of Things Journal*, vol. 1, no. 1, pp. 22–32, 2014.

[2] V. Cozzolino, A. Y. Ding, and J. Ott, "Fades: Fine-grained edge offloading with unikernels," in *Proceedings of the Workshop on Hot Topics in Container Networking and Networked Systems*, ser. HotConNet '17. New York, NY, USA: ACM, 2017, pp. 36–41.

[3] K. et al., "A survey of computation offloading for mobile systems," *Mobile Networks and Applications*, vol. 18, no. 1, pp. 129–140, 2013.

[4] L. M. Vaquero and L. Rodero-Merino, "Finding your way in the fog: Towards a comprehensive definition of fog computing," *SIGCOMM Comput. Commun. Rev.*, vol. 44, no. 5, pp. 27–32, Oct. 2014.

[5] S. Natarajan, R. Krishnan, A. Ghanwani, D. Krishnaswamy, P. Willis, and A. Chaudhary, "An analysis of lightweight virtualization technologies for nfv," 2017. [Online]. Available: https://tools.ietf.org/html/draft-natarajan-nfvrg-containers-for-nfv-03

[6] W. Felter, A. Ferreira, R. Rajamony, and J. Rubio, "An updated performance comparison of virtual machines and linux containers," in *2015 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, March 2015, pp. 171–172.

[7] R. Morabito, "Virtualization on internet of things edge devices with container technologies: A performance evaluation," *IEEE Access*, vol. 5, pp. 8835–8850, 2017.

[8] A. Madhavapeddy, R. Mortier, C. Rotsos, D. Scott, B. Singh, T. Gazagnaire, S. Smith, S. Hand, and J. Crowcroft, "Unikernels: Library operating systems for the cloud," in *ACM SIGPLAN Notices*, vol. 48, no. 4. ACM, 2013, pp. 461–472.

[9] A. Madhavapeddy, T. Leonard, M. Skjegstad, T. Gazagnaire, D. Sheets, D. J. Scott, R. Mortier, A. Chaudhry, B. Singh, J. Ludlam *et al.*, "Jitsu: Just-in-time summoning of unikernels." in *NSDI*, 2015, pp. 559–573.

[10] M. Gerla, E.-K. Lee, G. Pau, and U. Lee, "Internet of vehicles: From intelligent grid to autonomous cars and vehicular clouds," in *Internet of Things (WF-IoT), 2014 IEEE World Forum on*. IEEE, 2014, pp. 241–246.

[11] R. Morabito, R. Petrolo, V. Loscrı, N. Mitton, G. Ruggeri, and A. Molinaro, "Lightweight virtualization as enabling technology for future smart cars," in *International Symposium on Integrated Network Management (IM)*, 2017.

[12] M. Doering, "High-resolution large-scale air pollution monitoring: Approaches and challenges," in *Proceedings of the 3rd ACM International Workshop on MobiArch*, ser. HotPlanet '11. New York, NY, USA: ACM, 2011, pp. 5–10.

[13] S. R. Ellis, K. Mania, B. D. Adelstein, and M. I. Hill, "Generalizeability of latency detection in a variety of virtual environments," in *Proceedings of the Human Factors and Ergonomics Society Annual Meeting*, vol. 48, no. 23. SAGE Publications Sage CA: Los Angeles, CA, 2004, pp. 2632–2636.

[14] M. B. Lewis and A. J. Edmonds, "Face detection: Mapping human performance," *Perception*, vol. 32, no. 8, pp. 903–920, 2003.

[15] I. Farris, T. Taleb, H. Flinck, and A. Iera, "Providing ultra-short latency to user-centric 5g applications at the mobile network edge," *Transactions on Emerging Telecommunications Technologies*, 2017.