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 from the perspective of three selected use cases that are particularly interesting for harnessing the power of the Internet of Things. We discuss and compare the applicability of two LV technologies, containers and unikernels, as platforms for enabling the scalability, security, and manageability required by such pervasive applications that soon may be part of our everyday lives. To inspire further research, we identify open problems and highlight future directions to serve as a road map for both industry and academia.

Introduction

Over the last decade, the development of the Internet of Things (IoT) has been upheld 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.

This notion of cloud-connected IoT deployment assumes that most IoT networks need to connect to the cloud (e.g., through some gateway and tunnel approach). However, the increasing stringent performance requirements of IoT services, especially in terms of latency and bandwidth, challenge this deployment. Specifically, the existing model is not suitable when:

1. IoT networks create data that needs to be accessed and processed locally.
2. Piping everything to the cloud and back is not acceptable under delay constraints.
3. 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.

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 autonomous vehicles/drones, smart city infrastructure, and augmented reality (AR). As highlighted in Fig. 1, these three representative domains intersect. Edge computing is the link that helps spawn and promote appealing joint services.

Edge computing aims to satisfy key requirements such as scalability, multi-tenancy, security, privacy, and flexibility. In this respect, the fast evolving lightweight virtualization technologies seek to fulfill such demands, given their matching features. Meanwhile, we still lack comprehensive guidelines to illustrate how we can exploit the full potential of lightweight virtualization to enhance edge computing, especially for IoT use cases.

As a solid step toward 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 prospective 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?

Digital Object Identifier:
10.1109/MNET.2018.1700175

Roberto Morabito and Nicklas Beijar are with Ericsson; Vittorio Cozzolino, Aaron Yi Ding, and Jörg Ott are with Technical University of Munich.

0890-8044/18/$25.00 © 2018 IEEE

The remainder of this article is organized as follows. Motivations of the proposed work are presented in the following section. Then we introduce first the requirements that different *edge for IoT* cases entail, and the suitability of LV for mitigating and satisfying them. We next introduce LV technologies and illustrate three specific use cases. Finally, we unveil the open issues and challenges before concluding the article.

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 back-end. Being a central part of the ecosystem, the edge layer plays 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, a cloud provider, or a telecom operator (in mobile edge computing).

The placement of a "middle layer" between the end devices and cloud is an architectural concept that is widely utilized in common network infrastructures. Conventional middle layer functionalities mainly target connectivity, routing, and network-oriented operations. For example, network functions 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. In addition, it seeks to 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 an 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 the edge layer has to efficiently and mutually cooperate with both cloud-based services and IoT devices, by acting as a bridge between elements that require a distinct way of interacting.

In this context, it is crucial to equip the edge layer with tools that allow a flexible, well 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. Furthermore, by means of cross-layer support, edge elements must adapt to different types of traffic and to the application needs. 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.

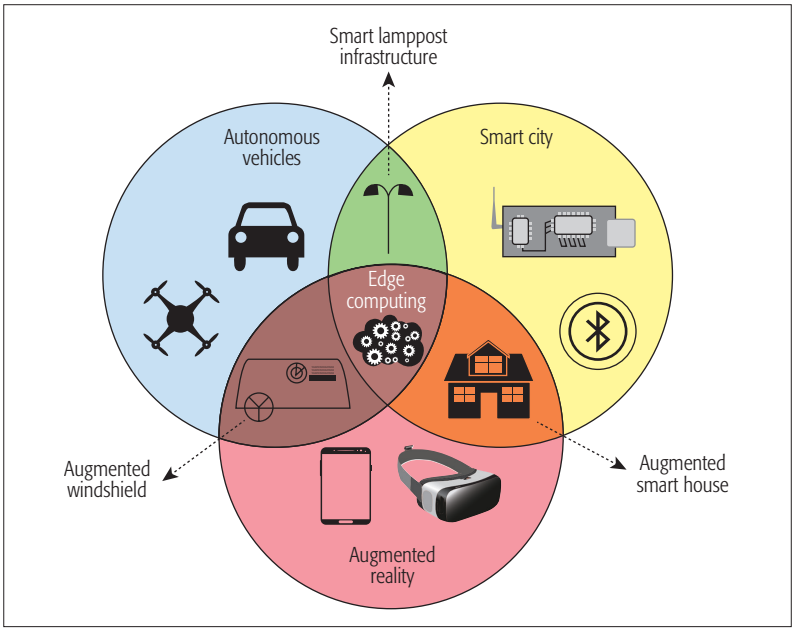


FIGURE 1. A subset of use cases and services enabled by IoT edge computing.

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 but efficient configuration and instantiation methods that are independent of the technologies used by different IoT and cloud providers. The tools embedded in the edge layer should share common functionalities and exploit common application programming interfaces (APIs) for orchestrating interconnections with different networking technologies.

To help us acquire a synoptic view, we highlight the dominant requirements of representative use cases in Table 1, which encompasses scalability, multi-tenancy, privacy and security, latency, and extensibility.

Compared to alternative virtualization solutions such as hypervisors, we envision a trend toward using LV technologies in IoT edge computing. These emerging software solutions can provide the needed support in terms of hardware abstraction, programmability, interoperability, and elasticity. A direct benefit that emerges from employing LV in the IoT edge domain is avoiding the strict dependency on any given technology or use case. Within an LV instance, either a container or a unikernel, 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 standalone virtualized applications. This feature avoids complex reprogramming and updating operations that are part of the software life cycle

Scenarios	Requirements				
	Scalability	Multi-tenancy	Privacy and 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 is 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 is critical in use-cases such 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 functions 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 needs to leverage real-time delivery services. NFV introduces additional sources of latency through the virtualization layer.	Non-critical

TABLE 1. Example of edge-IoT scenarios requirements.

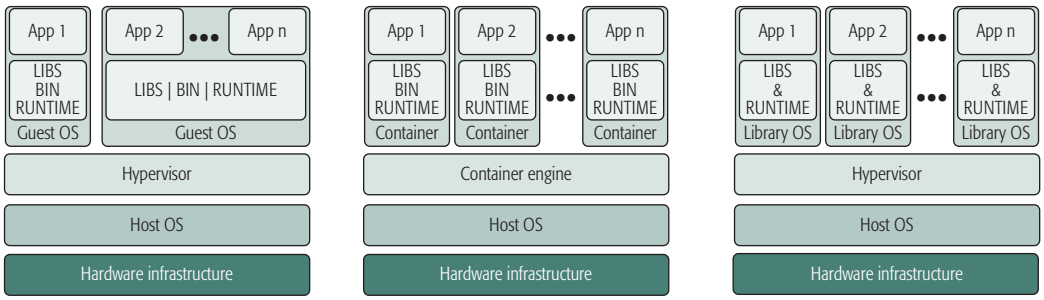
management. Through LV, such complexity is circumvented because updating a particular service requires changes only within a specific virtualized instance.

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

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) • Open VZ https://openvz.org/) • LXC https://linuxcontainers.org/	*~800/100 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*	*~< 5 MBs (bundle)*	*~ 8 MBs*	*Yes (i.e., MirageOS unikernels can only be written in OCaml)*	*High*	*No. Requires manual implementation*

(a)



(b)

FIGURE 2. LV techniques comparison: a) quantitative analysis; b) core architectural differences.

entails additional delay and resource 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.

Overview of Lightweight Virtualization

System virtualization has drastically evolved in the last few 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 to edge computing and IoT, we identify two main candidates that could address the challenges unique to this domain: containers and unikernels.

Figure 2 presents both quantitative metrics and architectural differences between the aforementioned technologies, highlighting their main characteristics.

Container-Based Virtualization: Docker

Container-based virtualization provides a different level of abstraction in terms of virtualization and isolation compared to other virtualization solutions. In particular, containers can be considered as one of the lightweight alternatives to hypervisor-based virtualization. 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 VM images. Furthermore, the emulation of virtual hardware devices and related drivers produces non-negligible performance overhead.

Differently, containers implement process isolation at the OS level, thus avoiding the virtualization of hardware and drivers [6]. Specifically, containers share the same OS kernel with the underlying host machine, meanwhile making it possible to isolate standalone applications that own independent characteristics: 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 timeframe. 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 a 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 Docker images to be stored as a series of layers, consequently saving disk space. In fact, the different image layers can be cached in the disk, allowing the building process to

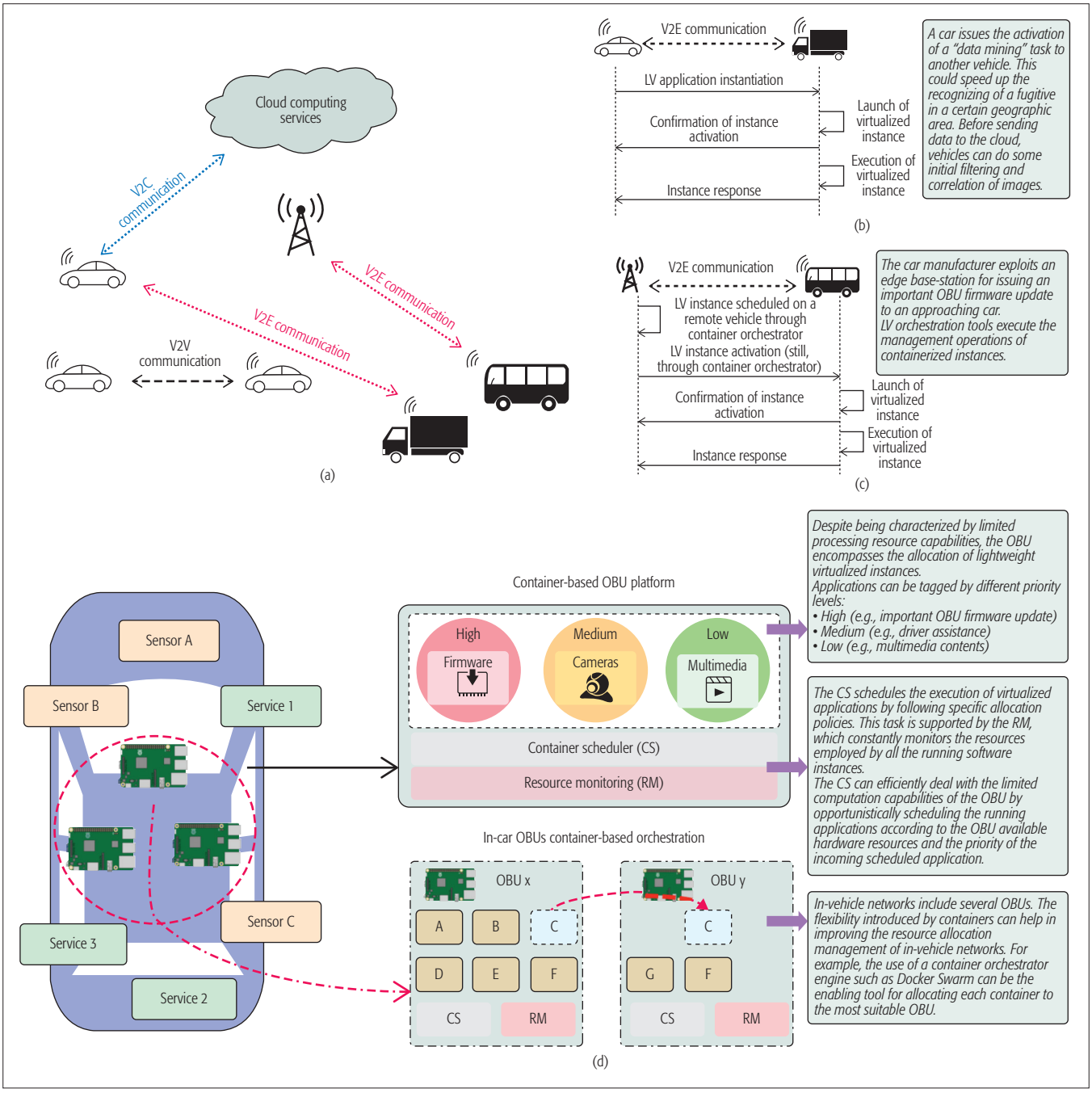


FIGURE 3. a) Vehicular edge computing scenario in its entirety, vehicular-to-edge (V2E) interactions examples; b) car-to-car V2E communication; c) base station-to-car V2E communication; d) container-based virtualization is used for easier OBU customization. Furthermore, within the same vehicle orchestration tools are exploited for task offloading among different OBUs.

be sped up, and reusing the same cached layer for building different images.

The lightweight features embedded in containers ease the integration of such technology in various networking fields. 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].

Library Operating Systems: Unikernels

Unikernels are single-purpose appliances that are specialized at compile time 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 to come 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 also fit well with the upcoming IoT edge ecosystem as illustrated through different research attempts [2–9]. 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 to bind generic applications, while the latter is highly specialized in offering dynamic network processing (based on the 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 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 to others. Unikernels are principally designed to be stateless. Therefore, they are perfect to embed general algorithms (e.g., compression, encryption, data aggregation functions) or NFV.

Use Case Scenarios

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

Toward 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 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 in IoV encompass, including cars' onboard units (OBUs), edge elements (EEs), and 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, including lightweight data mining operations, generic offloading processing, dashcam image filtering, and so on. In such context, LV can enable the VEC paradigm and be exploited in multiple scenarios, spanning from an efficient and flexible customization of cars' OBUs to vehicle-to-edge (V2E) interactions.

Figure 3 depicts the VEC scenario in its entirety (Fig. 3a), together with practical examples of the way in which LV can be employed in V2E interactions (Figs. 3b and 3c) and distributed in-car platforms (Fig. 3d).

V2E Interactions: Different from already well established vehicle-to-vehicle (V2V) communication, V2E aims to encompass computation offloading, task outsourcing, and software management operations. In practice, an 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 Figs. 3b and 3c.

In-Car Platforms: Container-based virtualization can be used for OBU customization. It offers high flexibility in the platform's software management, and allows overcoming the complex software updating procedures required by OBUs [11]. Through a conventional VM, car manufacturers can access all controller area network (CAN) bus sensors through OBU and dashcam. However, given that 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' task outsourcing, still following specific OBU resource management policies. More detail on the usage of LV for in-car platforms can be found in Fig. 3d.

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 threaten the others.

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 urban areas. Currently, air pollution monitoring is achieved with 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 stations is often prohibitive, we envision crowdsensing as a tangible solution that combines LV and edge computing.

Edge computing offers resources close to the crowdsensing entities, which can offload their collected data through direct connection without using a mobile connection. LV allows part of the required mathematical computation to be offloaded onto, and distributed and executed by, EEs without worrying about compatibility issues. For instance, multiple LV images can be created on demand, each containing only the code necessary to process the data of a single sensor. The partial results will then be subsequently uploaded to a more powerful edge device (e.g., edge data center) to be merged. Figure 4 provides more detail regarding how unikernels can both support 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 blackboxes 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]

[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.

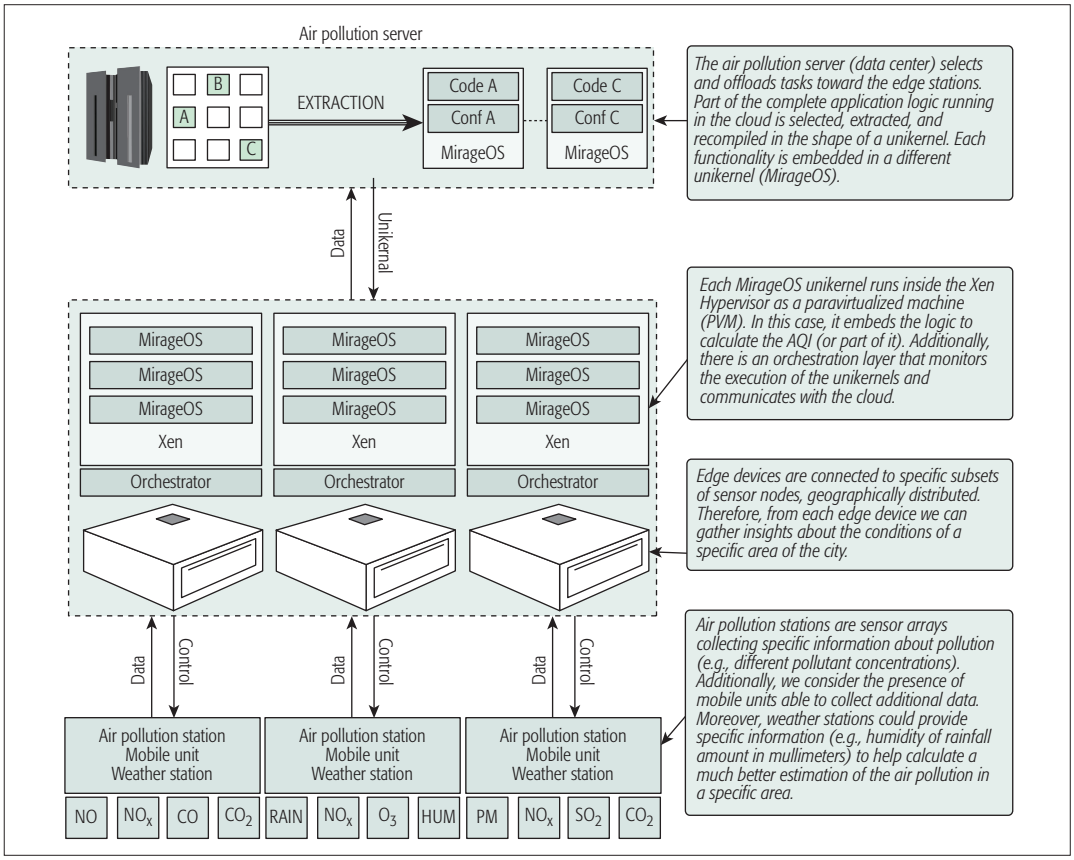


FIGURE 4. Air pollution scenario. An offloadable task is air quality index (AQI) calculation, a number used by government agencies to communicate to the public how polluted the air currently is or how polluted it is forecast to become. Calculation of the AQI can be executed locally by edge nodes enhancing real-time monitoring.

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 is required by the PC market. Therefore, it is not surprising that overall, wearable/mobile devices are not designed to run computationally intensive tasks.

Augmented Reality

Wearable devices are typically resource-constrained compared to the computer hardware of the 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 is 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 eventually cope with 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 have revealed the speed at which the human brain can identify faces in a dark scene and the requirements of a 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.

A use case where there is 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 his/her attention completely on the windscreen instead of checking the console to search for speed information. Additionally, the windshield will also provide traffic condition information, a personal agenda, a 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 cloud back-ends. Therefore, we can limit the latency impact, assuming that the computation time is *device-invariant*. The use of virtualization in such a context is additionally motivated by the following factors: *multi-tenancy* (i.e., multiple users executing multiple tasks) and *task isolation* for privacy.

For this specific use case, a combination of Docker and Unikernel represents a potential approach, as shown in Fig. 5. A Docker image containing multiple unikernels can be composed

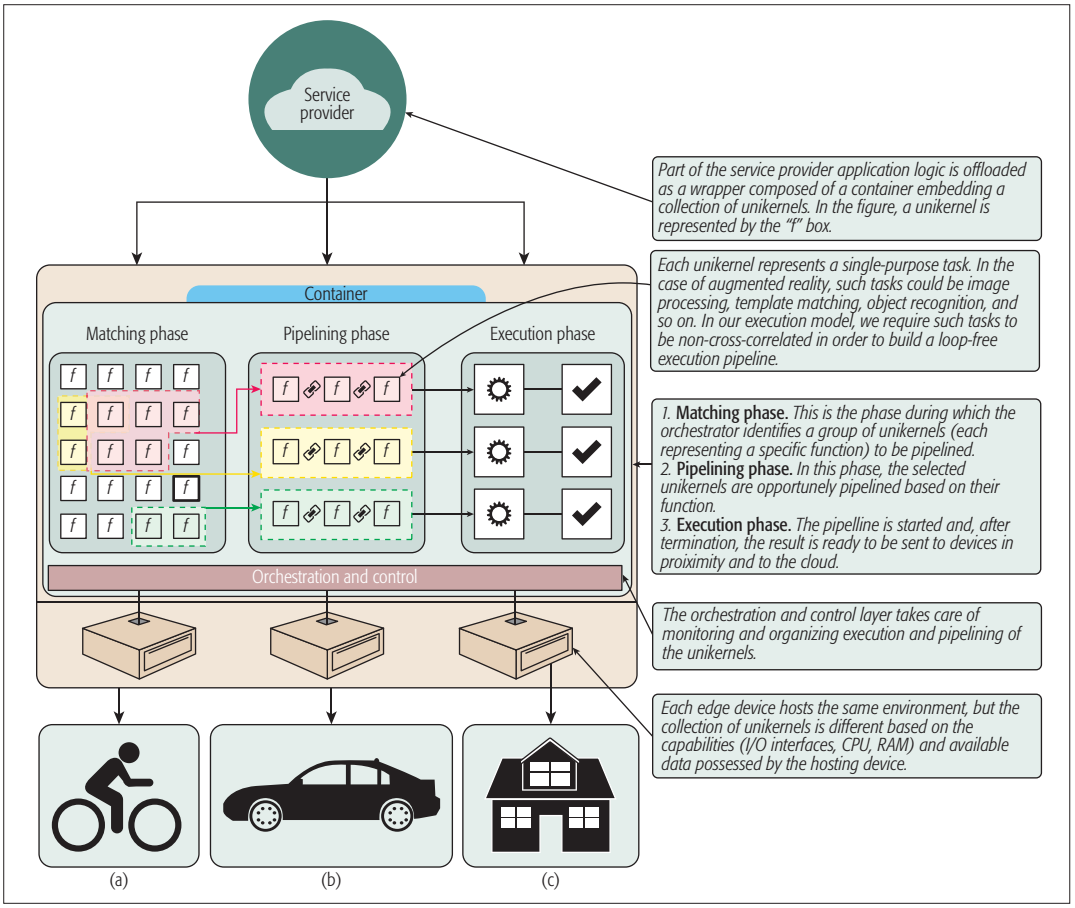


FIGURE 5. a) A biker receiving personalized advertisements rendered in augmented reality on his/her smart glasses; b) a smart car populating its augmented windshield with contextualized, live feed information; c) an augmented smart home, where we control IoT devices in proximity through virtual interfaces.

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.

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.

Orchestration and Monitoring

Orchestration of edge elements (EEs) and cloud architectures brings several challenges. Edge-IoT scenarios require specific tools to deal with the different processor architectures and storage capacity of EEs and cloud services. Controlling the network traffic requires the cloud and edge to be orchestrated, an increasingly challenging task with manifold EEs deployed. Hence, it becomes crucial to deploy lightweight orchestration modules that do not overburden the EE, and to seek a fair balance between synchronization and network load. Other key aspects concern the definition of optimized policies for efficient vertical scaling, in which applications are automatically prioritized and scaled up/down between EE and cloud, according to specific QoS requirements or computing resource 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 redeployed multiple times at different locations to transparently serve mobile users. In particular, if the service is specific to an individual user, the number of transfers may be high. Destroying and redeploying 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 the case of overallocation.

Controlling the network traffic requires the cloud and edge to be orchestrated, an increasingly challenging task with manifold EEs deployed. Hence, it is crucial to deploy lightweight orchestration modules that do not overburden the EE, and to seek a fair balance between synchronization and network load.

Regarding *monitoring solutions*, both technologies need high-performing, lightweight, and scalable monitoring frameworks. 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

The presence of multiple industry partners and researchers working in this field has given birth to different ramifications and interpretations of the same paradigm. Without standards and regulations, merging different approaches will be a nontrivial task exacerbated by the heterogeneity of the involved technologies.

requirement for monitoring engines is the possibility to track, in real time, the individual resources of each virtualized instance. Implementation of such frameworks becomes, in parallel with orchestration mechanisms, crucial in resource optimization and in developing efficient edge-cloud instance-placement algorithms and policies.

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 infringing on the security requirements. It is crucial to encourage the development of lightweight security mechanisms that take into account the strict requirements of IoT applications/scenarios and do 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 control the use of tenants' dedicated resources — for example, 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.

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 has given birth to different ramifications and interpretations of the same paradigm. Without standards and regulations, merging different approaches will be a nontrivial task exacerbated by the heterogeneity of the involved technologies. For LV technologies, lately there has been a growing effort to lay down 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 an additional standardization effort to be necessary that seeks to lay down precise guidelines toward the employment of LV in a wider range of IoT use case scenarios.

Elasticity in Service Provisioning

This feature is strictly dependent on the LV engines' capacity of quickly allocating/deallocating virtualized instances. Data reported in the table of Fig. 2a clearly show how both container and unikernel can promptly scale up/down. Furthermore, LV APIs also allow the *freezing* of the execution of an instance and quickly restoring it through checkpoint/restore mechanisms. However, there is still a lack of research to evaluate the interactions among multiple EEs, without neglecting the fact that current LV engines implementations do not provide full support for live migration. Specific frameworks that support proactive service migrations for *stateless* applications have already been proposed [15]. However, support for *stateful* applications' migration need to be integrated soon for fully exploiting LV benefits in these scenarios.

Management Frameworks and Application Portability

Employment of container technologies have had a disruptive rise in the last years, and the enormous effort that open source communities have provided on continually improving full-featured management frameworks has paid off. Unikernels seem to still not be mature enough to be included in production-ready environments, and greater effort is required for featuring the same portability of containers. Packaging applications through unikernels may require an implementation effort that somehow slows down, and in some cases limits, the adaptability toward 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.

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 because of both the volatile nature of edge nodes and the security risks related to easier physical exposure of the nodes. Therefore, data typically need 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 infeasible 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.

Telco Industry Readiness and Perspectives

The telecommunications sector is currently in a major paradigm shift moving in the direction of *softwarization* of formerly 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 with 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 premises equipment (CPE), such as residential gateways,

may extend the edge further. NFV is the main driver for edge computing in mobile networks, and a necessity for opening up 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 makes them more competitive for edge-dominated computation. The adoption of LV technologies in telco networks requires a change of mindset in the industry, but 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 to be replacements for VMs.

Summary

In this article, we examine the challenging problem of integrating LV with IoT edge networks. We first discuss 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 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 *et al.*, "Internet of Things for Smart Cities," *IEEE Internet of Things J.*, vol. 1, no. 1, 2014, pp. 22–32.

[2] V. Cozzolino, A. Y. Ding, and J. Ott, "Fades: Fine-Grained Edge Offloading with Unikernels," *Proc. ACM Wksp. Hot Topics in Container Networking and Networked Systems, ser. HotConNet '17*, 2017, pp. 36–41.

[3] K. *et al.*, "A Survey of Computation Offloading for Mobile Systems," *Mobile Networks and Applications*, vol. 18, no. 1, 2013, pp. 129–40.

[4] L. M. Vaquero and L. Rodero-Merino, "Finding Your Way in the Fog: Towards a Comprehensive Definition of Fog Computing," *SIGCOMM Comp. Commun. Rev.*, vol. 44, no. 5, Oct. 2014, pp. 27–32.

[5] S. Natarajan *et al.*, "An Analysis of Lightweight Virtualization Technologies for NFV," 2017; https://tools.ietf.org/html/draft-natarajan-nfvrg-containers-for-nfv-03.

[6] W. Felter *et al.*, "An Updated Performance Comparison of Virtual Machines and Linux Containers," *Proc. 2015 IEEE Int'l. Symp. Performance Analysis of Systems and Software*, Mar. 2015, pp. 171–72.

[7] R. Morabito, "Virtualization on Internet of Things Edge Devices with Container Technologies: A Performance Evaluation," *IEEE Access*, vol. 5, 2017, pp. 8835–50.

[8] A. Madhavapeddy *et al.*, "Unikernels: Library Operating Systems for the Cloud," *ACM SIGPLAN Notices*, vol. 48, no. 4. 2013, pp. 461–72.

[9] A. Madhavapeddy *et al.*, "Jitsu: Just-in-Time Summoning of Unikernels," *NSDI*, 2015, pp. 559–73.

[10] M. Gerla *et al.*, "Internet of Vehicles: From Intelligent Grid to Autonomous Cars and Vehicular Clouds," *Proc. 2014 IEEE World Forum on Internet of Things*, 2014, pp. 241–46.

[11] R. Morabito *et al.*, "Lightweight Virtualization as Enabling Technology for Future Smart Cars," *Proc. Int'l. Symp. Integrated Network Management*, 2017.

The adoption of LV technologies in telco networks requires a change of mindset in the industry, but technology questions remain for ensuring the reliability and security required for telecommunication networks.

[12] M. Doering, "High-Resolution Large-Scale Air Pollution Monitoring: Approaches and Challenges," *Proc. 3rd ACM Int'l. Wksp. MobiArch, ser. HotPlanet '11*, 2011, pp. 5–10.

[13] S. R. Ellis *et al.*, "Generalizeability of Latency Detection in a Variety of Virtual Environments," *Proc. Human Factors and Ergonomics Society Annual Meeting*, vol. 48, no. 23. SAGE Publications, 2004, pp. 2632–36.

[14] M. B. Lewis and A. J. Edmonds, "Face Detection: Mapping Human Performance," *Perception*, vol. 32, no. 8, 2003, pp. 903–20.

[15] I. Farris et al., "Providing Ultra-Short Latency to User-Centric 5G Applications at the Mobile Network Edge," *Trans. Emerging Telecommun. Technologies*, 2017.

Biographies

Roberto Morabito has worked for Ericsson Research Finland since May 2014 and has been involved in the FP7 ITN METRICS project. He has also been a Ph.D. student since September 2014 at Aalto University in the Department of Communications and Networking. His research interests include multi-access edge computing, the Internet of Things, and virtualization technologies. In 2013, he received his Master's degree in computer and telecommunications systems engineering from Mediterranea University of Reggio Calabria.

Vittorio Cozzolino is a Ph.D. researcher at the Technical University of Munich, Germany, where is working at the Chair of Connected Mobility. His research focuses on novel OS lightweight virtualization techniques, the Internet of Things, computation offloading, and edge networks. His recent research covers the development of a system for fine-grained edge offloading based on lightweight virtualization. In 2014, he obtained his computer science engineering Master's degree from the University Federico II of Naples, Italy.

Aaron Yi Ding received his M.Sc. (with distinction) and Ph.D. (with distinction) degrees from the University of Helsinki, Finland. He is a post-doctoral associate and a project leader with the Technical University of Munich. He was a visiting scholar at Columbia University in 2014 and the University of Cambridge in 2013 under the supervision of Prof. H. Schulzrinne and Prof. J. Crowcroft, respectively. His research interests include mobile edge computing, IoT security, and system networking. He was a recipient of the ACM SIGCOMM Best of CCR and the Nokia Foundation Scholarships.

Nicklas Beijar received his D.Sc. in networking technology from Aalto University, Finland, in 2010 and his M.Sc. from Helsinki University of Technology in 2002. He joined Ericsson Research in 2013 to work on Internet of Things and cloud technologies, in particular on distributed computing at the network edge. He is currently focusing on network slicing and cloud security. Prior to this, he worked at Aalto University as a research scientist and postdoctoral researcher on topics related to IP telephony, routing protocols for ad hoc network, peer-to-peer systems, and distributed search algorithms.

Jörg Ott holds the Chair of Connected Mobility in the Department of Informatics at the Technical University of Munich. He is also an adjunct professor for networking technology at Aalto University. He received his diploma and doctoral (Dr.-Ing.) degree in computer science from TU Berlin in 1991 and 1997, respectively, and his diploma in industrial engineering from TFH Berlin in 1995. His research interests are in network architecture, (Internet) protocol design, and decentralized networked systems.