Received July 4, 2019, accepted July 26, 2019, date of publication August 21, 2019, date of current version September 6, 2019.

Digital Object Identifier 10.1109/ACCESS.2019.2936714

Decentralized Iot Edge Nanoservice Architecture for Future Gadget-Free Computing

ERKKI HARJULA[1], PEKKA KARHULA[2], JOHIRUL ISLAM[1], TEEMU LEPPÄNEN[3], AHSAN MANZOOR[4], MADHUSANKA LIYANAGE[1,5], JAGMOHAN CHAUHAN[6], TANESH KUMAR[1], IJAZ AHMAD[2], AND MIKA YLIANTTILA[1], (Senior Member, IEEE)

[1]Centre for Wireless Communications, University of Oulu, 90014 Oulu, Finland
[2]VTT Technical Research Centre of Finland Ltd., 02044 Espoo, Finland
[3]Centre for Ubiquitous Computing, University of Oulu, 90014 Oulu, Finland
[4]Rovio Entertainment Ltd., 02150 Espoo, Finland
[5]School of Computer Science, University College Dublin, Dublin, D04 V1W8 Ireland
[6]Department of Computer Science and Technology, University of Cambridge, Cambridge CB3 0FD, U.K.

Corresponding author: Erkki Harjula (erkki.harjula@oulu.fi)

This work was supported in part by the Academy of Finland, in part by the Technology Industries of Finland Centennial Foundation, and in part by the Jane and Aatos Erkko Foundation.

ABSTRACT In the envisioned ubiquitous world, services will follow users as they move across smart surroundings. Services are instantiated to users through the environment, appearing and disappearing as they move, which reduces the need for personal communication devices such as smartphones or tablets. To facilitate this development, service architectures need to support virtualized, on-demand service composition based on the hardware and software resources available at the current user location. The technical context for this type of user interaction with digital services through smart surroundings is called Internet of Everything (IoE). Today's service architectures will be too inflexible in this highly decentralized and dynamic environment. Hence, in this article we propose a novel service model called *nanoEdge*, where nodes collaboratively provide needed functions for virtual services that need to be deployed locally due to performance, efficiency or reliability requirements, for example. The main contributions of this article are the *nanoEdge* conceptual model and its proof-of-concept (PoC) implementation to show that the model is feasible with regard to performance and resource-efficiency. The successful demonstration of PoC implementation exemplifies future IoE service scenarios with today's hardware components.

INDEX TERMS Edge computing, fog computing, IoT, IoE, virtualization, microservices, nanoservices, gadget-free computing.

I. INTRODUCTION

The digital world we currently live in is dominated by gadgets and electronic devices. The transition from local computing to cloud computing has made it possible to access all our digital content and services ubiquitously with any Internet-connected device [1], [2]. Until today, we have used different digital gadgets we carry with us, such as smart phones, tablets and laptops, to access our digital world. However, a major paradigm shift concerning the relationship between us and the digital world is just around the corner: a change from separate person-to-person, person-to-machine and machine-to-machine computing towards Internet of Everything (IoE) computing, encompassing intelligent connection of people, processes, data and things, where machines and humans communicate seamlessly using converged services and applications [3]–[5]. Since most of the data and services already reside outside our devices thanks to cloud computing, the transition to ubiquitous gadget-free world is the natural next step. Internet of Things (IoT) [6]–[8], with its recent advancements towards more versatile sensing including multimedia sensors [9], [10], as well as emerging 5G and Edge Computing technologies [11]–[13] have already started the transition towards the new era of digitalization. In this new era, it is possible to connect everyday computational objects to the Internet with sufficient performance and providing the needed computational resources close to the user. We are already approaching the world where infrastructure around us

The associate editor coordinating the review of this article and approving it for publication was Tariq Umer.

is connected and intelligent enough to deliver us the required services without the need of carrying gadgets, i.e. mobile devices.

In our recent research, we have envisioned a path towards the paradigm shift in the relationship between people and the digital world [1] [5], [14], [15]. In this ubiquitous approach, the user lives ''naked'' without gadgets. Services materialize for the user only when they are needed and disappear when not needed. The digital surroundings form an intelligent environment around users, providing all the information, tools, and services the users need in their everyday life. New digital ''service bubbles'' are created on the fly for different situations where people interact with each other or with the surroundings, such as meetings, events, sharing a car ride, etc. These service bubbles can follow users as they move, or they may be put on hold and re-established later in another time and/or location. In these scenarios, it is beneficial to compose services in such a manner that maximizes the use of computational hardware and software available on site.

To realize the above-mentioned vision, the underlying service architectures need to support on-demand service composition based on the hardware and software resources available at the current location. In this paper, we propose a novel virtualized and decentralized nanoservice-based model, *nanoEdge*, where nodes, based on their hardware capacity and load, collaboratively provide local processing, storage, security and privacy services without relying on centralized entities. With the concept of *nanoservice*, we mean a miniature version of *microservices* that are widely used in today's distributed cloud-based service provisioning. The key idea behind using nanoservices is that all capable nodes in the proximity can participate in the service provisioning: the high-capacity nodes can provide more resources and low-capacity nodes can do with less. The solution is highly scalable and gives tools for autonomous service composition based on the current need.

From the performance viewpoint, locally composed services provide inherently higher throughput and lower latency. From the reliability viewpoint, local service composition enables offline functionality of services, i.e. the operation of local services can continue despite the problems with access and core networks. From the security viewpoint, most of the sensitive user data can be kept in local machines to reduce the risks for security and privacy attacks.

The rest of this paper is structured as follows: Section II goes through the state of the art and existing related work, Section III introduces the proposed model, and Section IV presents a PoC implementation built on it. Section V presents the feasibility study of the proposed model and discusses the results, and finally, Section VI concludes the article.

II. BACKGROUND AND RELATED WORK

During the recent years, we have witnessed the evolution from local to virtualized data storage, computation, network management, applications and workspaces in the form of Everything as a Service (XaaS, or EaaS) and Cloud Computing (CC) [16], [17]. Virtualization refers to the replication of a device or its resources in virtual form, bringing some clear benefits over traditional systems, such as easy management, flexibility, universal availability and decreased hardware requirements for end-user devices. However, due to centralized operation, the XaaS-model has also introduced new challenges. A widely known challenge is related to communication latency due to high physical and logical distance between end-nodes and the server where the application logic and data storage resides. Increased latency is particularly problematic with delay-tolerant applications such as gaming [18]. Furthermore, since the digital world penetrates deeper and deeper in our everyday life, a particular concern is on the fact that cloud-based operation makes systems more vulnerable for attacks against privacy and availability of services[2] [19]–[21]. We are living in a world where our data and the data collected from us is ruthlessly exploited. What is even worse, IoT - surrounding us almost everywhere - gives cybercriminals further tools to even threaten our health or life[3] [4] [22]. Therefore, end-users are becoming more and more concerned on exposing their personal data to public networks and data centers [19], [23], [24]. These are among the most important driving factors towards Edge Computing [12], [25], [26], which pushes various computing and data analysis capabilities from centralized locations to the edges of a network.

A. EDGE AND FOG COMPUTING

Edge Computing (EC) [25], [26] brings a new computational tier to cloud computing, between data center and end-devices. EC enables services to exploit the proximity of devices, e.g. by providing highly-reliable ultra-low latency and high data rate communication and the ability to control and limit the scope of propagation of private user data. Multi-access Edge Computing (MEC) is a standard solution by European Telecommunications Standards Institute (ETSI) for forthcoming 5G networks to offload processing and data storage from mobile (and IoT) devices to the edge of mobile networks instead of passing all of the data and computation to data centers or handling them locally [12], [13], [27], [28].

Fog Computing [29], [30] is a term closely related to Edge computing. The distinction between these two terms is vague due to various overlapping definitions found in the literature. Our view is that whereas Edge computing mainly refers to the computational Edge infrastructure, Fog computing has stronger focus on providing a platform for services above

[1]The Naked Approach project: http://www.nakedapproach.fi/

[2]When 'Smart Homes' Get Hacked: I Haunted A Complete Stranger's House Via The Internet, http://www.forbes.com/sites/ kashmirhill/2013/07/26/smart-homes-hack/

[3]Teen hacks car with $15 worth of parts: http://www.pcworld.com/article/2886749/teen-hacks-carwith-15-worth-of-parts.html

[4]BMW's Connected Drive feature vulnerable to hackers: http://www.autoblog.com/2015/02/03/bmws-connected-drive-feature-vulnerable-to-hackers

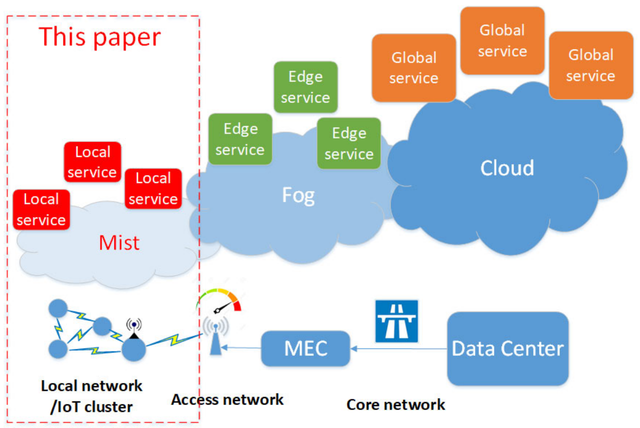


FIGURE 1. Generic edge computing architecture.

Edge infrastructure and local nodes (similarly to Cloud services that are deployed on data centers). Fog computing typically covers caching, data processing and analytics occurring near the source of the data that improve the performance at the edges of the network, reduces the burden on data centers and core networks and improves the resilience against networking problems [23], [30]. Virtualized functions and services located at IoT infrastructures are sometimes also referred to as Mist computing, as shown in Figure 1.

By moving some functions from data centers to the edge, CC systems can better serve applications requiring low latency while saving computational and networking resources at core networks and data centers. The parts of services that require low latency or provide functions for reducing data, such as filtering, fusion or other processing, are beneficial to deploy at MEC. The former because the E2E latency between the local node and MEC node is very low, and the latter because less data needs to be delivered to data centers. As can be seen, IoT and smart environments can greatly benefit from MEC residing at the mobile access networks.

However, the current model where MEC hosts are deployed at servers located within or near the access network base stations also has its limitations [31]. In many smart space and IoT applications, to deal with possible connectivity problems and to limit the propagation of sensitive data outside the domain, at least some degree of processing of the sensor data and the decision-making/control logic is beneficial to be managed locally on site. Therefore, in many scenarios it is beneficial to bring EC capacity within local IoT clusters, as illustrated in Figure 2. Since it cannot be expected that local IoT/IoE clusters include devices with sufficient stability and hardware capacity to accommodate full-functional MEC host, alternative decentralized solutions fitting better to the IoT/IoE environments need to be studied. The following subsections will focus on technologies that can be used as building blocks for decentralized EC solutions.

B. LIGHTWEIGHT VIRTUALIZATION

Lightweight virtualization technologies have revolutionized the world of software development by introducing flexibility and innovation to this domain and recent advances have led to the spread of such technologies in different contexts [16], [32]–[34]. In many IoT scenarios, a high number of nodes, ranging from few units to swarms of several thousands of nodes, may be deployed for a single service. These swarms need to adapt to changes in the environment, infrastructure and application deployments, and they occasionally need software upgrades for improved functionality, reliability or security [33]. The rapid development of IoT hardware capabilities has made virtualization a viable option, not only in data centers, but also on IoT devices, which are characterized by fewer computational resources, such as single-board computers working, e.g. as multimedia sensor nodes [32].

Today, there are basically three alternative lightweight virtualization technologies: hypervisor solutions such as KVM, container engines such as Docker, and unikernels such as MirageOS. KVM virtual machine offers hypervisor-based virtualization, supporting, e.g. multi-tenancy. The drawback of hypervised virtual machines is their relatively large memory footprint combined with slow boot-up time [33]. Container solutions, Docker technology [32], [34], in particular, are more suitable for use in IoT, thanks to their relatively low memory requirements and faster boot-up times. Many existing studies have successfully evaluated Dockers within the IoT domain [32], [33]. One of the most attractive features of containers from the viewpoint of IoT is that they can be automatically scheduled and orchestrated on top of any physical or virtualized computing infrastructure [35]. The third alternative, unikernels, provide the fastest boot-up time and smallest memory overhead of the three alternatives. However, the technology is still relatively immature [33].

C. MICROSERVICE ARCHITECTURES

In recent years, cloud services have been transforming from monolithic architectures towards *microservice architectures*, where services are composed of various microservices taking care of some limited set of functions [36], [37]. Microservices are an architectural style to build, manage, and evolve service architectures out of small, self-contained units [38]. Businesses such as Amazon and Netflix use microservices to build large, complex and horizontally scalable applications composed of microservices that are small, independent and highly decoupled processes communicating with each other using language-agnostic application programming interfaces (API). According to [39], microservice architecture provides several advantages over traditional monolithic architectures: (1) reduced complexity by using tiny services; (2) easier deployment and removal of system parts; (3) improved flexibility to use different frameworks and tools; (4) increased scalability; and (5) improved system resilience.

Docker, introduced in the previous subsection, provide a proven lightweight, low overhead and fast technology empowering the usage of microservice architectures [34], [35]. From the viewpoint of IoT, an important feature of Docker containers is that they can be deployed in a

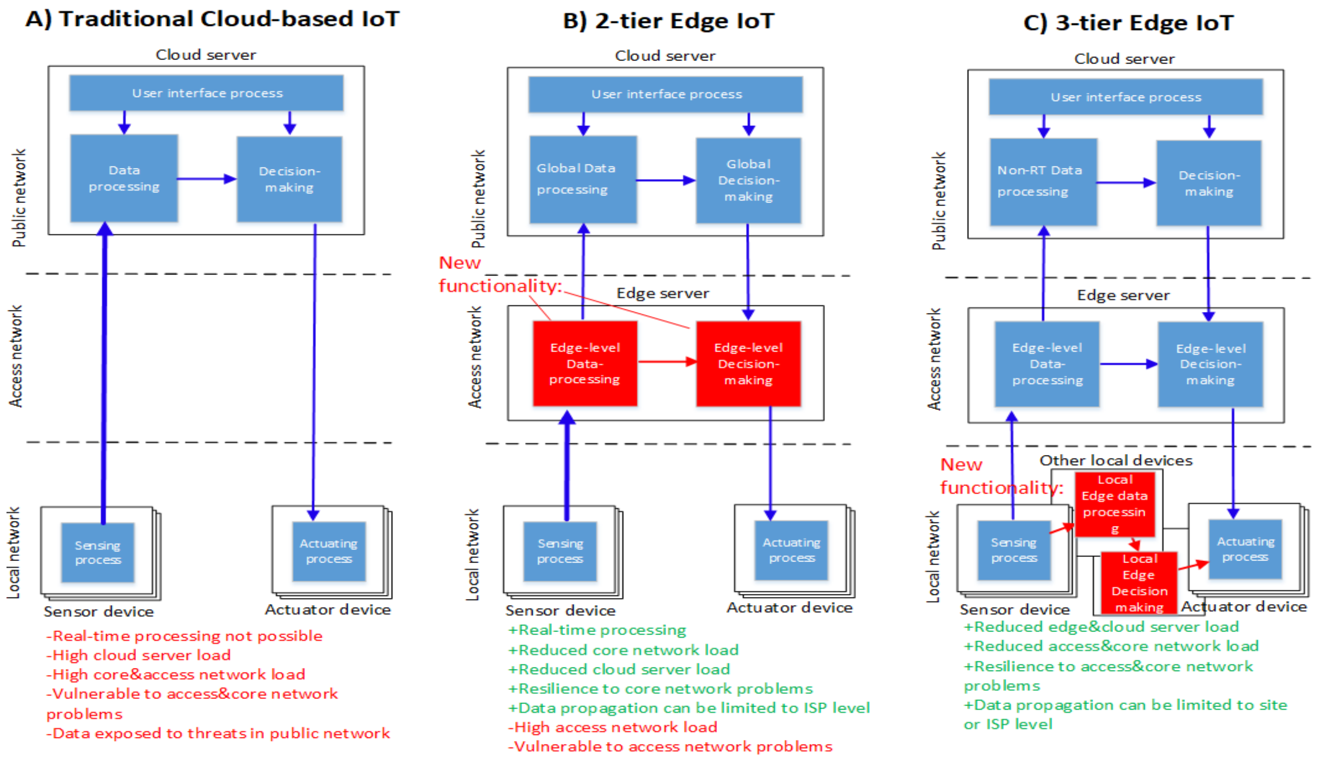

A) Traditional Cloud-based IoT
Cloud server
User interface process
Data processing
Decision-making
Public network
Access network
Local network
Sensing process
Actuating process
Sensor device
Actuator device
-Real-time processing not possible
-High cloud server load
-High core&access network load
-Vulnerable to access&core network problems
-Data exposed to threats in public network
B) 2-tier Edge IoT
Cloud server
User interface process
Global Data processing
Global Decision-making
New functionality:
Edge server
Edge-level Data-processing
Edge-level Decision-making
Sensing process
Actuating process
Sensor device
Actuator device
+Real-time processing
+Reduced core network load
+Reduced cloud server load
+Resilience to core network problems
+Data propagation can be limited to ISP level
-High access network load
-Vulnerable to access network problems
C) 3-tier Edge IoT
Cloud server
User interface process
Non-RT Data processing
Decision-making
Edge server
Edge-level Data-processing
Edge-level Decision-making
New functionality:
Other local devices
Local Edge data processing
Local Edge Decision making
Sensing process
Actuating process
Sensor device
Actuator device
+Reduced edge&cloud server load
+Reduced access&core network load
+Resilience to access&core network problems
+Data propagation can be limited to site or ISP level

FIGURE 2. Comparison of IoT architectural models.

manner where only one or few processes run inside a single container [39].

An important aspect related to the management of virtualized services is how to accommodate realizing, orchestrating and maintaining complex and dynamic microservice architectures. *Orchestration* is a technology for deploying, managing and termination of virtualized components, such as containers. The most commonly used container orchestration technologies are Docker Swarm, Kubernetes and Mesos, all providing automated support, e.g. to service discovery, load balancing, and software upgrades on the fly [35].

Microservices, if managed to be kept small enough, provide a promising approach for using microservices within the IoT domain to accommodate IoT functions locally. Butzin *et al.* [40] have investigated this idea and ended up to a conclusion that microservice approach and IoT share the same architectural goal and would therefore be a promising combination. In this paper, we introduce the term *nanoservice* to refer to ultra-lightweight microservices specifically developed for IoT scenarios. Since containers can be deployed in such a manner where only one or few processes run inside single container, we consider it a suitable technology to implement nanoservice-based orchestrated IoT architectures.

D. MOBILE SOFTWARE AGENTS

Software agents [41], [42] enable the implementation of system components capable of autonomously and intelligently acting and interacting with other system components on behalf of their owners. Agents can be reactive, proactive and even capable of learning, based on observing the results of their actions. In addition, mobile software agents (in short, mobile agents) can autonomously migrate between devices, during their task execution. This allows re-purposing and personalizing the behavior of system components, e.g. IoT devices, towards smart behavior and bringing new functionality or content into the system. In distributed systems, mobile agents have traditionally been used for relocating, composing and local aggregation of services [41].

The benefit is that mobile agents, owned by the users and with knowledge of the users' goals and behavior, can interact and react autonomously (in the background) to the system state and others' actions. Furthermore, an attractive feature of mobile agents from the viewpoint of IoT computing is their low overhead, enabling very small agent implementations. In [43], we have evaluated the mobile agent size and estimated the energy consumption caused by migration of mobile agents between wireless devices in a distributed computing scenario. Both the agent size and the energy consumed during agent migration remained very small. In the context of IoT, mobile agents have been used to bridge, share and personalize physical spaces between users and other system components, e.g. [42], [44], [45]. However, holistic system architectures utilizing agents for service provisioning in IoT edge computing have not been considered yet.

E. ACTOR AND CONSTRAINT PROGRAMMING

Actor programming model paradigm [46], describes complicated computations as a set of actors, i.e. individual components with singular purpose, to perform each task. Actor model-based systems are compositions of several actors: they do not have shared state, but instead communication between actors happens through message exchange. Thanks to this, the distributed (micro/nano-)service model combined with edge/fog computing can take great benefit from the Actor programming model paradigm. Haubenwaller and Vandikas have, for instance, utilized Actor programming for composing services in IoT Edge computing in [47]. The implementation is based on Akka, an open-source actor-based message-driven runtime built on Java Virtual Machine (JVM). They propose an approach where data is processed by computationally capable IoT devices rather than being sent to a central location for processing. The processing tasks are split to smaller tasks and deployed IoT devices in an efficient manner. Since actors are message-driven and each of them are expected to have a single responsibility, the main operational philosophy is close to what microservices should be like. Therefore, Actor model is a highly applicable model to define the high-level interaction model between microservices in microservices systems.

Constraint programming Model [48], for one, focuses on optimizing resource-efficiency of the systems under development. In constraint programming, the computer finds the answer satisfying constraints, when giving a problem as an aggregate of constraints to computer. Together, actor and constraint programming models are a promising combination for optimizing the resource- efficiency required by IoT.

III. NANOEDGE CONCEPT

We propose a nanoservice-based conceptual service model, *nanoEdge*, for future gadget-free IoE scenarios. It utilizes local computational resources for deploying parts of cloud services in proximity of data sources and/or service consumers. The model takes Edge Computing a step forward from a typical today's MEC-based architecture by providing means to deploy some parts of edge services to local nodes with sufficient hardware capacity.

This three-tier model allows addressing problems, such as high latencies and vulnerability to network problems arising from long distances between computation, data sources and service consumers, as illustrated in Figure 2. The model utilizes and combines the state-of-the-art concepts and models presented in the previous section in a novel way.

In this section, we describe the main operational principles of the proposed nanoEdge model and illustrate these principles in action with an example scenario. Then, we explain in more detail how services are composed, executed and maintained in our model. In this section, we avoid making bindings to any specific technologies on purpose, in order to maintain generality. However, later in Section IV, we present a PoC implementation with relevant today's technologies in order to analyze the feasibility of the model.

A. OVERVIEW AND BASIC CONCEPTS

The basic principle behind nanoEdge is that local services are composed of modular, independently operating but collaborating virtual service blocks, *nanoservices*, that are dynamically deployed to local nodes based on the need and their available capacity and load. As a concept, nanoservice is a simple service implemented for a single purpose, such as reading sensor data from a device and sending it to another node, or running an analysis task and returning the value for the requestor, i.e. a lightweight microservice [40]. In practice, nanoservice is a small virtually deployable program with an API for other services and nanoservices. The dynamical deployment means that the deployment can change based on the availability of nodes and resources, e.g. when new computational nodes become available, when existing nodes become unavailable, or when their loads change.

Nanoservices and services provide APIs for making them modular and interconnectable, i.e. services can use other services and nanoservices in their operation. *Local services* are services deployed to a site and composed of nanoservices available at the site. The site may be fixed (such as a building, room, park, etc.), or mobile (such as a car, plane or train). Local services may communicate with other services and be parts of larger services, depending on their privacy settings. Local services can be set as *private* or *public*. Private local services are locally visible and accessible through local API, whereas public local services are visible and accessible also for users and services outside the site through a public API. All non-local services are called as *external services*.

Services are created and administrated using an *API gateway*, which provides the needed building blocks for composing services. In a nutshell, the API gateway collects information about the computational capacity and available sensing and actuating resources from compatible physical nodes within proximity and passes this information to the *orchestrator*, which then deploys nanoservices on suitable physical nodes. The nanoservices are deployed to nodes from a *service registry*, which contains the nanoservice images.

Figure 3 visualizes the concept. Physical nodes are illustrated by gray circle shapes, nanoservices by hexagons, connectivities by line/lightning (depending on whether it is wired or wireless) and public and edge networks by cloud. Services consisting of the nanoservices are illustrated with orange ellipses and service registry with a green ellipse.

The formal presentation of services S and nanoservices s is presented as follows.

Let the set of available nanoservices in a site be:

$$s = \{s_1, s_2, \ldots, s_n\}. \tag{1}$$

Respectively, the set of services within the site is:

$$S = \{S_1, S_2, \ldots, S_n\}. \tag{2}$$

Within this set of services, each service is defined as a function of nanoservices and services, as follows:

$$S_i = \{f(s_j, S_k)\}; \quad s_j \in s; \ S_i, S_k \in S. \tag{3}$$

B. SERVICE COMPOSITION AND MANAGEMENT

In this subsection, we describe the main principles of service composition in our nanoEdge model. API gateway and

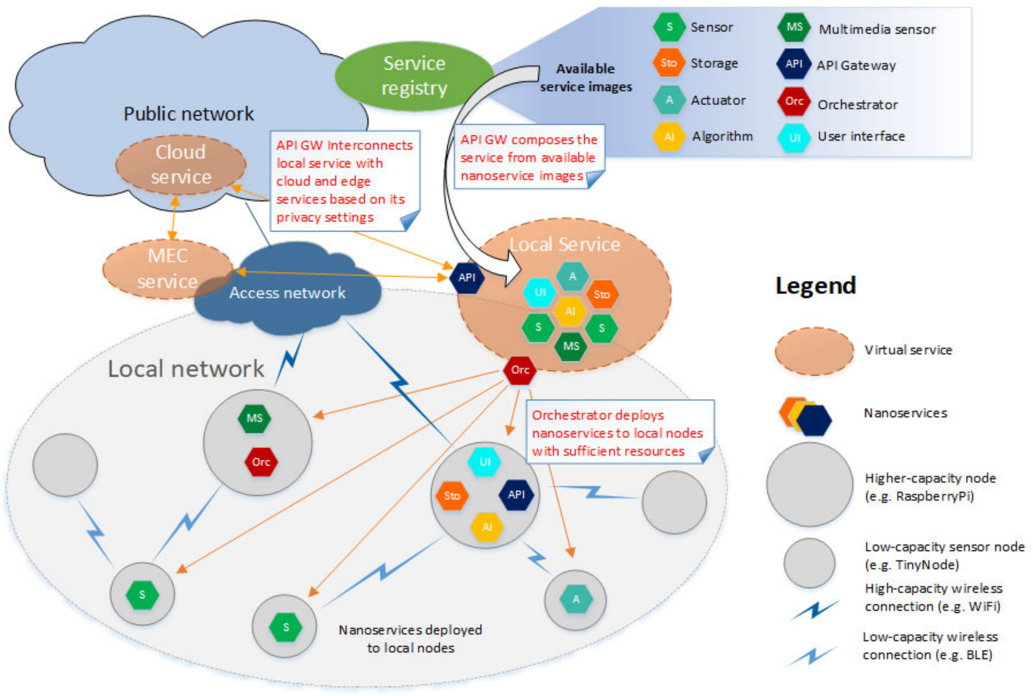

Public network
Cloud service
Service registry
Available service images
S Sensor
Sto Storage
A Actuator
AI Algorithm
MS Multimedia sensor
API API Gateway
Orc Orchestrator
UI User interface
API GW Interconnects local service with cloud and edge services based on its privacy settings
API GW composes the service from available nanoservice images
MEC service
Access network
Local Service
Local network
Orchestrator deploys nanoservices to local nodes with sufficient resources
Nanoservices deployed to local nodes
Legend
Virtual service
Nanoservices
Higher-capacity node (e.g. RaspberryPi)
Low-capacity sensor node (e.g. TinyNode)
High-capacity wireless connection (e.g. WiFi)
Low-capacity wireless connection (e.g. BLE)

FIGURE 3. The proposed *nanoEdge* concept.

Orchestrator are centric components for service composition and management. API Gateway is a core component through which administrators create, administrate and terminate local services and through which other services interact with the local services. API gateway uses Orchestrator to deploy, redeploy end undeploy the main service and the needed nanoservices to optimal locations. In Figure 3, API gateway is represented by a dark-blue hexagon and Orchestrator by a red hexagon. API gateway keeps track on all available physical resources within proximity, and passes this information to Orchestrator, making it able to deploy nanoservice images from a service registry (top of Figure 3) to different nodes based on their capabilities. Both public and local service registries can be used (in the figure, the service registry is located in a public cloud). The API gateway (and Orchestrator) are aware, e.g. which local nodes provide sensing and actuation capabilities, storage space, gateway functionalities, processing power for analysis, etc. Since we try to keep the model general, we do not define what is the exact way of delivering this information from nodes to the API gateway. The usual methods are broadcast advertisements, where the nodes advertise their capabilities by sending broadcast advertisements that the Orchestrator then intercepts, and polling, where Orchestrator either broadcasts or uni-/multicasts capability request messages for which nodes then respond with their capabilities. Furthermore, there are two options for implementing the orchestration: integrating it with the API gateway or implementing it as a separate component. In the figures, we illustrate orchestration as a separate Orchestrator component.

1) SERVICE INITIATION

As already mentioned, services are administrated through the API gateway. Its task is to match the available hardware capacity and features with available nanoservice images to present the deployable nanoservices for the administrator. The administrator can then define the service logic based on the nanoservices (using e.g. visual service composing tools). In addition to the service logic, the administrator also defines the API functions through which the service can be used by other services. At this phase, the administrator can also choose whether the service is local or public.

To illustrate the main principle of service composition in our model, we consider a very simple IoT service that regulates a room temperature and shows it on a wall-mounted LCD display. This service would be composed of a temperature sensor, radiator actuator, decision logic and UI nanoservices. Let's assume a hardware setup consisting, for instance, of a local node 1 providing a temperature sensor with some computational capacity and a small display with some control buttons, a local node 2 providing a fair amount of processing capacity and a large LCD touchscreen display, and a local node 3 that is a specialized for controlling the heating radiator. In this setup, a nanoservice for reading and sending the temperature data for other nodes could be deployed to node1, nanoservices containing the temperature regulation algorithm and the UI to node 2, and heating unit controller nanoservice could be deployed to node 3. It is important to notice that some nanoservice types are bound to certain nodes or physical locations (such as the radiator controller which is the only node capable of controlling the heating radiator),

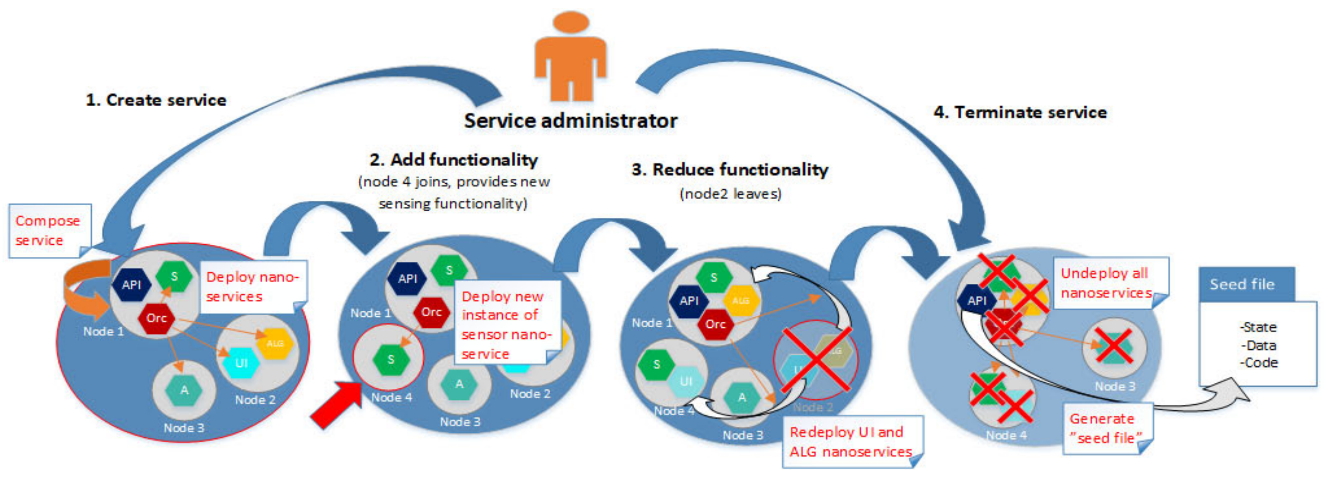


FIGURE 4. ***nanoEdge* service lifecycle.**

while some other nanoservice types may be independent of the physical node or location (such as a nanoservice providing a computational algorithm) that can be deployed on any of the local nodes with sufficient computational and networking capacity. In order to enable matching services with the available hardware, the hardware requirements of the available nanoservices need to be provided for the API gateway when advertising their capabilities (see previous subsection). The service initiation process of the scenario is illustrated on the leftmost section (phase 1) of Figure 4.

2) SERVICE RUNTIME MANAGEMENT

During service operation, its deployment can be modified, updated, extended or reduced, based on the changes in need and circumstances, as illustrated by the mid-section (phases 2 and 3) of Figure 4. As mentioned in the previous subsection, the API gateway keeps track of available hardware capabilities within the service site and passes this information to Orchestrator. When, e.g. new nodes enter the service site, the orchestrator becomes aware of them and can deploy new nanoservices on them if needed. Similarly, new nanoservice images can be added in the service registry during service operation. In the case of functionality, or security updates for existing nanoservice images, the orchestrator takes care of updating the deployed nanoservices based on them.

nanoEdge follows the principles of the Actor model, which means that the services and nanoservices do not have runtime interdependency between each other and external services, i.e. nanoservices and services may fail without causing the (nano)services interacting with them to fail. If a physical node hosting a nanoservice fails or moves away, the orchestrator checks whether the needed physical capabilities are available on another node, and in the positive case, redeploys the nanoservice to a new physical node with sufficient capabilities. In case none of the available nodes can provide the needed function, the orchestrator terminates the nanoservice and notifies API gateway of the termination. The service can then, based on its internal logic, decide how to react to the change.

3) SERVICE TERMINATION

When a service is terminated, the orchestrator undeploys all deployed nanoservices from physical nodes and notifies API gateway, which then notifies other services it has been interacting with. The service can optionally store its status and data (either all or a subset of it) to a *seed file* which can be used in possible future re-establishments of the service. This seed file can be stored either locally or, e.g. in a public cloud to make it universally accessible. Service termination is illustrated in phase 4 of Figure 4.

C. SERVICE RUNTIME OPERATION

The proposed nanoEdge model follows the Actor model [46]. In practise, this means that the service components - nanoservices - operate independently, i.e. they do not share their state and therefore are not interdependent. Nanoservices interact among each other in an asynchronous manner, i.e. service requests or messages can be sent at any time between nanoservices without regard to whether or not the recipient nanoservice is ready. Furthermore, since nanoservices are not interdependent, if a nanoservice fails, the rest of the system is not affected by more than by the impact of the failing of the task of the failing nanoservice. In error cases, the nanoservice making a request to another nanoservice can continue execution with alternative ways to complete its own task, or if it is impossible, report the failure to the (nano)service that has requested the service. In other words, failure tolerance is built in to the nanoEdge model.

IV. PROOF-OF-CONCEPT

To study the feasibility of the approach, we have defined a use case scenario (Section IVA) and a PoC prototype (Section IVB) that implements the scenario by employing a selected set of functionalities of the nanoEdge concept.

A. EXAMPLE SCENARIO

In the example scenario (illustrated in Figure 5), Alice calls up a meeting by sending invitations to Bob and Carl. Since highly confidential topics will be discussed in the meeting,

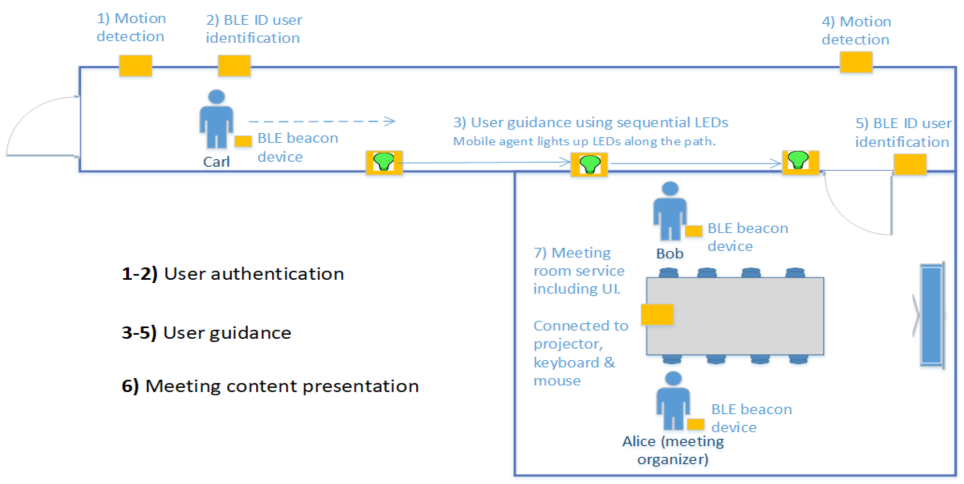

1) Motion detection
2) BLE ID user identification
4) Motion detection
BLE beacon device
Carl
3) User guidance using sequential LEDs
Mobile agent lights up LEDs along the path.
5) BLE ID user identification
BLE beacon device
Bob
7) Meeting room service including UI.
Connected to projector, keyboard & mouse
1-2) User authentication
3-5) User guidance
6) Meeting content presentation
BLE beacon device
Alice (meeting organizer)

FIGURE 5. Proof of concept scenario.

each participant needs to be identified and authenticated by the system or the participants before entering the meeting room. To enable effortless authentication, Alice, Bob and Carl carry personal wireless identification tags (e.g., smart key or ring) with them.

At first, Alice creates the meeting event with a cloud-based service management tool. She chooses a meeting service template and selects an available meeting room from a calendar view. The management tool presents the list of the available local functions (nanoservices) to Alice. Since the meeting will be confidential, Alice chooses the option requiring user authentication before users can enter the meeting room. Several user authentication methods are available. Alice decides to use Bluetooth Low Energy (BLE)-ID based method since it does not require manual effort to be identified, and therefore, fits well to this gadget-free scenario.

Then Alice chooses the methods to guide users to the meeting room. She chooses to use LEDs located on the walls/floors of lobby and corridors on the way to the meeting room. She sets the BLE-ID presence detector in the meeting room to detect who is present in the room. Further, Alice also chooses her private content (emails, etc.) to be opened on the screen when she enters the meeting room herself, and furthermore, to move the private content to the background while opening the meeting content on the screen when other meeting participants start to arrive. She also defines the meeting content and status to be automatically saved after the meeting ends, in order to enable resuming the meeting content in follow-up meetings. Once the service has been defined, the main service is composed, the needed nanoservices are deployed to the local nodes, and both are set to idle mode to wait for the meeting to start.

Some minutes before the meeting starts, the service activates its nanoservices. When Alice enters the meeting room herself, the BLE-ID based presence detector in the room identifies Alice and the service opens her personal workspace on the screen, including personal documents, emails, etc., as specified beforehand. When guests, Bob and Carl, arrive at the building, they are identified by BLE-ID user identification in the main lobby. The system then automatically guides them to a correct meeting room using the visual indicators selected by Alice, in the lobby and corridors on the way to the meeting room. Once Bob, who arrives first, enters the meeting room, the presence detector in the room detects him and the content on the screen is adjusted so that Alice's personal documents and emails are moved to the background and meeting materials are opened. When Carl enters the meeting room, the meeting can start. When the meeting is over, Alice terminates the meeting service. At this phase, the meeting content and the service state are stored in a database and the meeting service can be reinstantiated in the future.

This scenario exemplifies the operation of our proposed model in the envisioned gadget-free world, where several nanoservices located on different types of physical devices in the building together form the meeting service. The users do not need to carry any personal devices with them, except the identification tags.

B. EXAMPLE SERVICE IMPLEMENTATION

Our example service consists of eight functional parts: the API Gateway (i.e. the main service), the management tool, the orchestrator, and five nanoservices: 1) Corridor presence detection, 1) Authentication engine, 3) BLE scanner, 4) User guidance, and 5) Meeting room service. The user presence detection, identification and guidance take place at a corridor leading to the meeting room and the meeting room service in the meeting room. The functional parts are described below.

1) MANAGEMENT TOOL, API GATEWAY AND ORCHESTRATOR

The meeting event is created with the service management tool connected to the nanoEdge API gateway, which is co-located with the Main Controller Service (MCS). When initiating a local service, the API gateway matches the nanoservices in the service registry with the list of nodes and their hardware resources within the physical/logical proximity of the selected meeting room. Based on this, the management tool presents the list of the available nanoservices to the administrator, who can then define the service utilizing the available nanoservices. Once the service has been defined, the API gateway contacts the orchestrator, which then automatically composes the MCS, deploys the needed nanoservices to the local nodes, and sets all nanoservices in idle mode to wait for activation.

2) MAIN CONTROLLER SERVICE

Main Controller Service (MCS) is the service which takes care of the main service logic and maintains the service state. It utilizes the nanoservices deployed in the proximity for accomplishing different tasks to implement different functions of the service. In our scenario, the service is composed as described in Figure 6. The control sequence is described with the red (authentication part), orange (guidance part) and green (meeting room part) arrows.

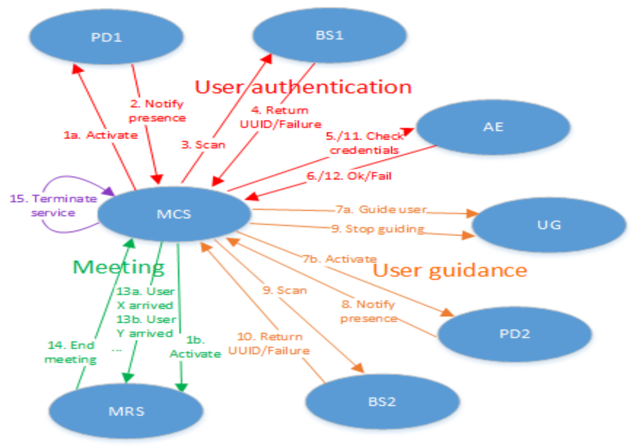


FIGURE 6. Service interaction in the example scenario.

Once the MCS is activated, it activates the Presence Detection nanoservices (PD) deployed at the motion sensor nodes located at the entrance of the corridor (PD1) and next to the meeting room door (PD2). The user identification process is started when a person enters the corridor. After receiving a notification of movement from PD1, the MCS activates the BLE scanner nanoservice located at the entrance of the corridor (BS1), which starts scanning for users to authenticate. When BS1 has successfully scanned the BLE-ID of the person in the corridor, it returns it to MCS, which then sends a query with the BLE-ID to the Authentication Engine nanoservice (AE) to check the user's credentials.

If credentials are ok, MCS sends a request to User Guidance nanoservice (UG) to start guiding the guest to the meeting room. When the guest arrives at the meeting room door, the PD2 located next to the meeting room door triggers another BLE scanner (BS2), co-located with it to check the identity of the arriving person. Then, MCS compares the BLE-IDs and if the arriving person's ID matches the guided person's ID, the guidance is considered successful and MCS requests UG to stop guiding (if there are no other users being guided at the same time). If the arriving person is the first guest or the meeting host (Alice), MCS also activates the Meeting room service (MRS) nanoservice. In case BS1 fails to detect the BLE-ID, or if the detection is successful but AE does not recognize the person, the person is considered an outsider and is therefore omitted from further actions.

When all guests have successfully been guided to the meeting room, MCS notifies MRS about the situation and the meeting is considered started. During the meeting, MRS interacts with MCS as defined by the service. When the meeting ends, the meeting host closes the service using the UI provided by MRS. At this point, MCS saves the session status and starts the process of terminating the service. In case MRS has not initiated the service termination after a timeout period following the scheduled end of the meeting, and if no activity has been detected from MRS, MCS terminates the service automatically. At the service termination phase, MCS undeploys all related nanoservices, saves its state and data to a seed file and terminates itself. The seed file can be used to redeploy the meeting service for future meetings.

MCS can be deployed to any local node with sufficient computational, storage and network capacity.

3) PRESENCE DETECTION

Presence Detection (PD) nanoservices are deployed on two devices capable of sensing motion: one is located at the entrance of the corridor (PD1) to detect incoming persons, and another next to the meeting room door (PD2) to detect when guided guests have arrived at the meeting room. When activated, the task of PD is to sense user presence at the proximity and notify MCS when presence is detected. PD includes an event handler and a callback function that notifies MCS when the event handler has detected motion in the proximity. PD can be deployed on a node with a sensor capable of detecting presence in its proximity, such as a PIR motion sensor, and sufficient computational capacity for running the nanoservice.

4) BLE SCANNER

In BLE-based authentication, users carry a BLE beacon device with a unique ID. BLE has the ability to exchange data using advertising packets. BLE Beacons take advantage of the GAP advertising mode to broadcast data out in periodical, specially formatted advertising packets. Each beacon uses a universally unique identifier (UUID) that identifies that beacon. Once activated, the BLE scanner nanoservice (BS) initiates the UUID-scanning and waits for five seconds for it to complete. If a device is detected, BS responds to MCS with the detected UUID. If UUID is not found within five seconds, BS sends MCS a response with the failure notification and returns to idle mode waiting for further requests. BS can be deployed on a node with BLE connectivity and sufficient computational capacity to run the nanoservice.

5) AUTHENTICATION ENGINE

The Authentication engine (AE) nanoservice is used to store information about authorized persons for checking their credentials to access the meeting room. In our scenario, after receiving the BLE UUID from BS, MCS sends an authentication request containing the BLE UUID to AE. AE then makes a comparison between the requested UUID and the user credential database, and responds with a message telling about the success or failure of the authentication. AE can be deployed to any local node with sufficient computational, storage and network capacity. The operation of PD, BS and AE is illustrated in phases 1-2 of Figure 5 and red arrows in Figure 6.

6) USER GUIDANCE

The User Guidance (UG) nanoservice has been implemented using user-specific mobile agents, atop the framework presented in [42], [45]. The mobile agent-based user guidance is illustrated in phases 3-5 of Figure 5 and orange arrows in Figure 6. Once MCS has triggered UG to start guidance, it first plans a path for the guest and, with the knowledge of system resources, creates a mobile agent migrating between LED controller devices mounted on the walls along the path for guiding the guest to the meeting room. This setup demonstrates a personalized mobile software agent "following" a real user in a physical environment in the infrastructure side, which dynamically and autonomously interacts with the local resources in the proximity of a user. The guidance is continued until MCS sends a request to stop guiding (in practice, after the guided user has arrived at the meeting room). UG nanoservice requires a controller device with sufficient computational, storage and network capacity to control access to the subsystem and two IoT devices equipped with LED strips and sufficient computational, storage and network capacity for the purpose of guiding a user to a preferred location.

7) MEETING ROOM SERVICE

When activated, the Meeting Room Service (MRS) nanoservice located at the meeting room starts waiting for the meeting host and guests. When the meeting host (Alice) and users enter the meeting room, MCS notifies MRS about the arrival of them. MRS provides the UI for the service using the available hardware, such as monitors, video projectors, keyboards, mice, etc. The shown content is based on who is present in the meeting room, as specified in MCS subsection. During the meeting, MRS interacts with MCS as defined by the service. When the meeting ends, the meeting host closes the service using the UI provided by MRS. At this point, MRS saves the modified files, meeting minutes, etc., turns off the display and notifies MCS, which starts the process of terminating the service. At the service termination phase, MCS undeploys all related nanoservices, including MRS, and saves its state and data to a seed file and terminates itself. MRS requires a device with sufficient computational, storage and network capacity to provide UI functions and to communicate with MCS. The operation of MRS is illustrated in phase 6 of Figure 5 and green arrows in Figure 6.

C. HARDWARE AND SOFTWARE SETUP

We have implemented a real-world PoC prototype of the nanoEdge in IoT environment, based on the nanoEdge concept and the example service scenario described above. To demonstrate the generality of the model, we have used several state-of-the-art technologies in the implementation. The overall operation follows the constraint and actor programming models [46], [48].

The hardware and software setup for this prototype is described as follows. The API Gateway, the Main Controller Service (MCS), and the microservices were deployed into Raspberry Pi embedded computers, three microcontrollers and a PC computer. The main service (MCS) and most of the nanoservices (PD, BS, UG and MRS) were developed using Python 2. AE nanoservice was developed with Python 3 and the mobile agent part of UG were developed with C++. The MCS and nanoservices communicate among each other using Constrained Application Protocol (CoAP), a specialized lightweight web transfer protocol for constrained nodes and networks in the Internet of Things [49]. The MCS and each nanoservice run CoAP client and CoAP server scripts to enable two-way communications. The hardware testbed setup is illustrated in Figure 7 and the nanoservice deployment on the testbed in Figure 8.

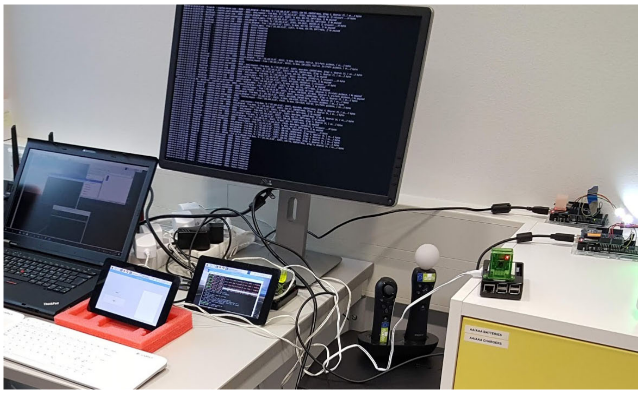


FIGURE 7. Proof-of-concept hardware.

The nanoservices are virtualized as Docker containers using Docker technology [34], [35]. The nanoservices are wrapped into Docker images that can be stored in a service registry and can be deployed when needed. The images come with all the required libraries and configurations, making it quick to set up additional nanoservice instances based on them. Furthermore, we have used mobile agents [42], [45] in the guidance part to demonstrate a mobile nanoservice that can migrate between highly capacity-constrained physical devices. For orchestrating nanoservices, we have used DockerSwarm [33]. Container orchestration has been implemented into Oracle VirtualBox with version 5.2. Command Line

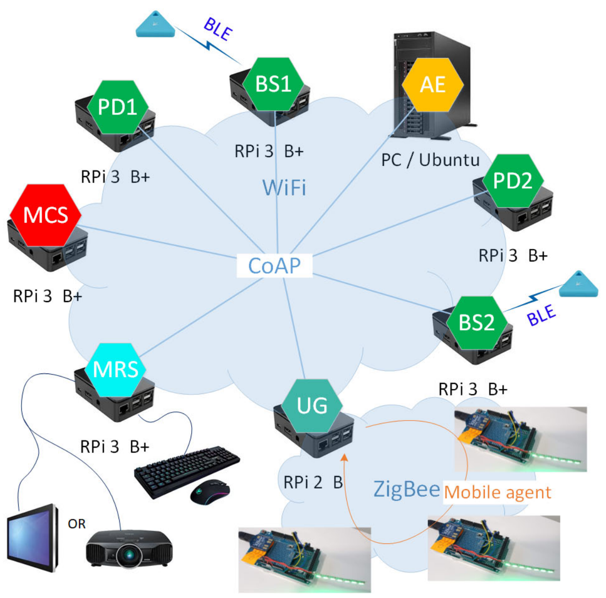


FIGURE 8. Nanoservice deployment on PoC hardware.

Interface (CLI) based Docker Swarm is integrated into the Docker ecosystem with its own API.

Docker containers are deployed to devices from a container image. Docker base image is the basic image on which developers add their own layers and create the final image containing the service or application. To build the nanoservice images, we used the Alpine Linux base image. Alpine is a lightweight Linux distribution, built on Musl-libc library and Busybox Linux distribution for embedded devices. It is smaller and more resource efficient than traditional GNU/Linux distributions. Alpine base-image enables us to run an application requiring only minimal dependencies required for the application. Furthermore, Docker versions from 17.05 facilitate multi-stage-builds[5] to reduce the computational resource consumption even more.

The main controller service (MCS) is deployed on a Raspberry Pi 3 Model B+. MCS contains the main service logic and the main API for external access and maintains the service state. MCS also runs a CoAP server and client processes for communications with nanoservices.

The AE nanoservice is deployed into a PC computer with Ubuntu 18.04 Linux OS. AE was developed using Python 3 in Flask 1.0.2 (Flask Restful 0.3.6) framework. AE works as a server providing a REST API for incoming CoAP authentication request messages. As the authentication database we use a MySQL 8.0.16 database. The authentication responses are sent back using a CoAP response messages.

For PD nanoservice, we used a Passive Infrared Radio (PIR) motion detection sensor expansion board attached to a Raspberry Pi 3 Model B+ computer. When motion is detected, an event handler function of the PD nanoservice detects a voltage rise on the RPi GPIO4 pin connected to the PIR motion sensor and sends a callback function using a CoAP request message to the MCS.

For BS nanoservice, we used a BLE expansion board attached to a Raspberry Pi 3 Model B+. In BLE-ID authentication, the incoming BLE beacons contain a universally unique identifier (UUID) that identifies the device sending it. In our PoC, an off-the-shelf BLE beacon has been configured to broadcast the 16-byte UUID. The BS nanoservice deployed on a Raspberry Pi device scans for the BLE beacon-advertising packets. When detected, PD extracts the UUID from the advertising packet returns it to the requestor (MCS in our scenario) using a CoAP request message.

The UG nanoservice consists of the main controller device, deployed on a Raspberry Pi 2 Model B, and LED controller devices built on top of ATmega 2560 microcontrollers. The UG main service hosts a CoAP server for receiving instructions from the Main Service and uses a CoAP client for deploying mobile agents to the LED controller devices. The operation of the LEDs depends on the task brought into the device by the mobile agents. The communication between the LED devices and the gateway is established using ZigBee wireless mesh networking. The LEDs are capable of showing RGB colors as well as simple animations.

The Meeting Room Service (MRS) nanoservice provides the UI at the meeting room, including the needed input devices, keyboard and mouse in our scenario, and output devices, projector and speakers in our scenario, as well as the needed local programs or an interface to cloud services. In addition, MRS also runs a CoAP server and client processes for communication with the MCS and other nanoservices and a script for controlling the content viewed on a screen based on meeting attendant presence. In our scenario, MRS is deployed on a Raspberry Pi 3 Model B+, which is able to act as a lightweight personal computer needed to successfully run the meeting-related programs.

V. FEASIBILITY ANALYSIS

Since the nanoservices are deployed on local, mostly resource-constrained, IoT devices in our model, it is important that they fit in the available hardware and at the same time provide sufficient performance and tolerate changes in the environment. Although the main focus of this paper is introducing the main concept and not a thorough evaluation and benchmarking of the concept against traditional solutions, during the prototyping work we made evaluations with our PoC implementation regarding some centric performance and resource consumption metrics. In this section, we present and analyze the evaluation results.

A. RESOURCE CONSUMPTION

Low resource consumption is among the highest-priority criteria when analyzing the feasibility of a system to be deployed on an IoT environment. In this section, we evaluate the base image and container sizes, as well as runtime memory consumption that together define the computational and communication requirements for the underlying hardware.

[5]Docker docs - Use multi-stage builds: https://docs.docker.com/develop/develop-images/multistage-build/

TABLE 1. Resource consumption of *nanoEdge* service components using alpine base image.

Component / service	Storage consumption		
	Base Image size	Container size	Runtime consumption
Main Controller Service (MCS)	53.0 MB	184.6 MB (basic) OR 92.2 MB (multi-stage)	231 kB (basic) OR 1.55 MB (multi-stage)
Authentication Engine (AE)	87.0 MB	273.8 MB (basic) OR 137.1 MB (multi-stage) + 443.0 MB (MySQL)	394 kB (basic) OR 2.3 MB (multi-stage) + 7 B (MySQL)
Presence Detection (PD)	53.0 MB	184.8 MB (basic) OR 92.7 MB (multi-stage)	231 kB (basic) OR 1.54 MB (multi-stage)
Bluetooth Scanner (BS)	53.0 MB	273.8 MB (basic) OR 93.7 MB (multi-stage)	239 kB (basic) OR 2.42 MB (multi-stage)
User Guidance (UG)	53.0 MB	191.4 MB (basic) OR 73.8 MB (multi-stage)	17.6 kB (basic) OR 776 kB (multi-stage) + 60 B (mobile agent at ATmega 2560)
Meeting Room Service (MRS)	53.0 MB	193.8 MB (basic) OR 92.2 MB (multi-stage)	231 kB (basic) OR 1.55 MB (multi-stage)

Table 1 demonstrates the base image size and the size of the nanoservice container built on it. The base image size has an effect on network utilization and storage consumption during the container deployment phase, whereas the container size mainly affects the resource consumption on the device after it has been deployed. In addition, the table includes the container runtime size depicting the used writable layer unique per container, i.e. the required space by the dynamic content in addition to the container read-only parts. This is a significant value when several containers based on the same container image are deployed on the same device: whereas the read-only parts of several deployed containers do not reserve more space on a device than one deployed container, each container, however, reserves the runtime part separately, therefore, increasing the resource consumption. We built our nanoservice images using traditional one-stage build and multi-stage build to see how different build methods affect the container size and runtime resource consumption.

As can be seen, the Alpine-based images for our nanoservices are roughly in the class of some tens of Megabytes and the respective container sizes with traditional one-stage build between roughly 184 and 274 Megabytes. With multi-stage builds, we were able to reduce the container size roughly between 74 and 137 Megabytes. In addition, a separate container - sized roughly 443 Megabytes - including the MySQL database service was needed in the AE node.

Furthermore, based on our measurements, the size of the mobile agent used by the UG nanoservice is only around 60 Bytes, and most of it is composed of executable code and the path for the agent. With this approach, the payload size is more than if traditional message patterns were used. However, the advantages come from the added functionality, which reduces the number of messages. With added functionality, we mean that the agent has some properties that the traditional publish-subscribe or client-server models and protocols do not have. The key difference is the autonomous migration from device to device. This reduces the number of round-trip messages sent, and therefore, reduces, e.g. the energy consumption coming from communication.

Altogether, it is clear that the proposed distributed nanoservice architecture brings some extra overhead in the resource consumption at the IoT nodes since they now run services that have traditionally been taken care of by centralized servers. With the evaluations presented in this subsection, however, we have shown that the resource requirements do not grow substantially. Generally, all nanoservices and platform components can be deployed and run on IoT nodes with hardware capacity on a level of a typical Raspberry Pi node with a decent requirement level for computational, memory, storage and communication resources. Furthermore, we demonstrated that the resource requirement level of virtualization can be taken on a whole new level with the mobile agent technology, allowing the simplest virtualized functions to migrate to highly capacity-constrained microcontroller nodes, such as ATmega 2560.

B. SERVICE DEPLOYMENT AND INITIATION PERFORMANCE

In addition to resource consumption, sufficient performance is another key feasibility criteria for a service. We selected two basic performance metrics that have high potential to be negatively affected by the distributed nature of a service. These metrics are 1) service deployment time and 2) service initiation time when initiating a service that has already been deployed on a device.

Service deployment time is the total required time to build and deploy a service into a physical device. The overall service deployment times are visualized in Figure 9. The deployment time with Alpine-based base-images with traditional build was 121s for the MCS, 82s for the PD, 78s for the BS, 165s for the AE, 67s for the UG and 88s for the MRS nanoservice. When using a multi-stage build, the deployment times drop to 52, 54, 56, 65, 59 and 53 seconds, respectively. Regarding the service initiation time, the overall time

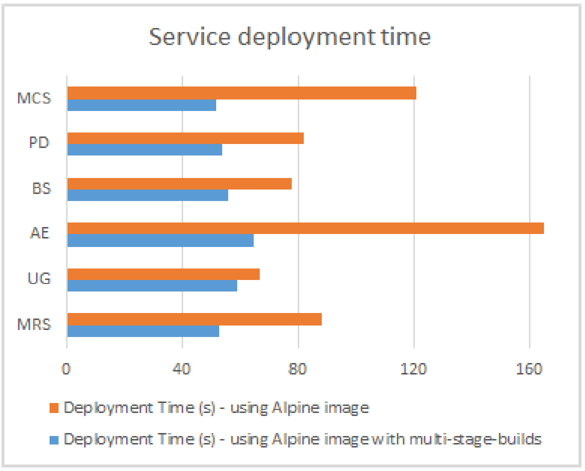


FIGURE 9. Nanoservice deployment time.

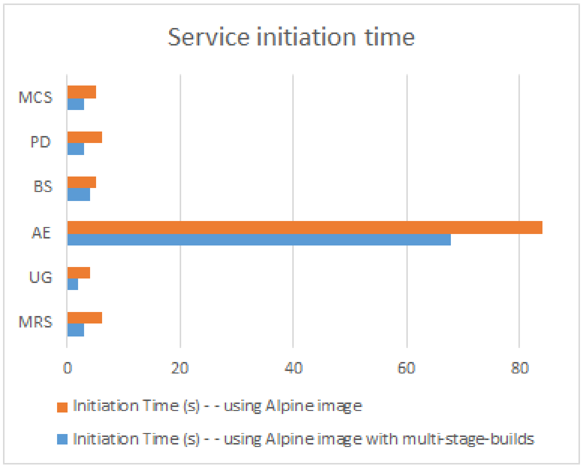


FIGURE 10. Nanoservice initiation time.

with traditional build was 5s for the MCS, 6s for the PD, 5s for the BS, 84s for the AE, 4s for the UG and 6s for the MRS nanoservice. With multi-stage-based container images, the service deployment was very fast, 3s for the MCS, 3s for the PD, 4s for the BS, 68s for the AE, 2s for the UG and 3s for the MRS nanoservice. The overall service initiation times are visualized in Figure 10.

Overall, the time to deploy the service takes roughly 1-3 minutes per nanoservice with traditional Alpine build. With multi-stage build the time drops to roughly in class of one minute for each nanoservice. This reduces considerably the overall service deployment time, which mainly depends on the slowest deployment when the deployment is done concurrently for different devices. In our case, this means the multi-stage build reduces the overall deployment time from 165s to 65s. For most nanoservices and the main service, the time to initiate a deployed service was 4-6 seconds with traditionally built images and 2-4 seconds with images built using multi-stage build. AE nanoservice was an exception: the service initiation time was 84s with traditionally built images and 68 seconds with images built using multi-stage build. The initiation was slower due to the time needed for MySQL module initiation.

Altogether, the achieved overall service deployment time of roughly one minute and initiation time of 2-4 seconds with multi-stage-built container images (except the AE nanoservice containing the MySQL module, that took a bit over a minute) can be considered feasible in our scenario. In practice, the measurement results mean bringing up the service for the first time takes around two minutes (deployment+initiation). The service initiation time of just a few seconds allows scenarios where already deployed nanoservices (except AE nanoservice in our case that takes over a minute to initiate) can be activated during runtime based on the need. This helps, e.g. saving energy of devices hosting non-active nanoservices, since, thanks to short initiation time, they can be kept in idle mode when there is no immediate need.

C. RUNTIME PERFORMANCE AND FUNCTIONALITY

We have already shown in our previous study [33] that the runtime performance is in general not significantly affected by containerization. However, we wanted to see whether the example scenario is feasible with our highly decentralized setup consisting of several wirelessly communicating IoT devices. This particularly concerns the user authentication and guidance performance, which should be fast enough to allow seamless guidance to the meeting room in such a manner that the guided person does not need to stop for waiting for guidance at any point.

In our scenario, the authentication is accomplished in two separate processes that includes BLE scanning and user authentication. Therefore, to evaluate the runtime performance, we measured the total time the system uses for 1) user presence detection, 2) user authentication, and 3) the latencies related to user guidance. For presence detection, we measured the time between a user entering the area under surveillance and the time when the MCS received a notification of user presence. Each measurement was performed three times and the average was taken. The observed times for user presence detection varied from 1 to 3 seconds, the average was roughly 2 seconds. To evaluate the BLE-ID authentication performance, we measured the maximum response time of the service to discover and authenticate the user under average network conditions. Each experiment was performed three times and the average was taken. The total duration for BLE beacon discovery, retrieval of the data and authentication took 3 seconds on average, the results varying between 2–4 seconds. We achieved 100 % success rate when the authenticated person carried the BLE beacon and had a previously created user profile. In total, the time to detect the presence and authenticate a user took 7 seconds at maximum.

According to measurements, the creation of a mobile agent and route planning is very quick in our scenario, in the order of tens of milliseconds. Also, the latency of the agent migration from device to device is almost instantaneous to the human eye. For the demo, the migration latency has been artificially increased in order to allow some time for the LED animations to run before the agent jumps to the next device. The total traverse time depends on the time the agent spends in the device multiplied by the number of devices in the path. Currently, the delay has been set to 2.5 seconds per device, and with two devices, the total traverse time is 5 seconds.

Although it is obvious that the distributed virtualized architecture causes some degree of extra overhead in the communication and computation, our observations show that the distributed architecture does not seem to cause such a significant penalty for the functionality and performance that would be detrimental to the user experience. With the average 1 m/s walking speed, the maximum authentication time (7s) would mean that after detection, the user would walk 7 meters before the guidance would start.

D. DISCUSSION AND FUTURE WORK

According to the evaluations, we can clearly state that the proposed local edge computing model based on virtualized distributed nanoservices is feasible with regard to resource consumption and performance. We were able to demonstrate that local services can be deployed and run in a distributed manner on a set of single-board-computers (SBC), such as Raspberry Pi, with feasible performance using Docker-based nanoservices, and that simple functions can be deployed to even more resource-constrained microcontroller devices, such as ATMega 2560, using mobile agent technology.

The significance of the proposed model becomes clear when taking a glance at the potential of using local computational resources in service provisioning. The typical devices found at our everyday surroundings with computational and networking capacity comparable with Raspberry Pi devices used in our PoC and therefore potential devices for hosting nanoservices include e.g. smart TVs, broadband routers, home/building automation central units, car computers, and surveillance cameras, just to mention a few. We should neither forget the more traditional computing devices, such as PC computers, tablets, laptops and smartphones providing very high computational and networking capacity that may be available for nanoservice deployment. Furthermore, typical devices found at our everyday surroundings containing a microcontroller with a computational capacity comparable to the ATMega 2560 used in our PoC include devices such as fridges, freezers, ovens, air ventilation and conditioning devices, heating controllers, different sensors, as well as smart locks, lights, switches, and power plugs. A growing number of these devices can communicate using low-power wireless radios such as BLE, ZigBee or other similar technologies, making them also potential host nodes for functions encapsulated in mobile agents.

As discussed in Section II, many concepts and implementations exist in the context of edge-centric networking in IoT application area. The closest work with regard to the similarity of the concept is proposed by Haubenwaller and Vandikas [47]. The work focuses as well on an approach where IoT processing tasks are split into smaller tasks and deployed on local IoT devices. However, their evaluation focuses completely on performance, ignoring the resource-efficiency. Direct performance comparison is also hard to make since their evaluation scenario is built on a different platform (Akka) and a different use case scenario. Furthermore, in contrast to our model, their deployment is static and cannot be changed during runtime. A comparative study with related work is, however, an interesting avenue for future work as new publications with similar concepts emerge.

The envisioned three-tier cloud computing architecture includes cloud servers at the core of the Internet, the MEC hosts at access networks, and the nanoservices at local networks. In this architecture, different services and their parts can be deployed in an optimal way based on the service requirements and available computational and network capacity provided by the underlying architecture. Since we see local nanoservice systems as an inherent building block of this three-tier architecture, we see it is important to direct the work towards integration with the other tiers.

Since the current PoC implementation contains only the minimal functionality needed to evaluate the feasibility of the proposed nanoEdge concept, the future work includes developing several additional functions. To allow external and inter-service access, a well-defined nanoEdge API needs to be developed. Related to this, we see a clear need for providing certain service components on higher layers of the three-tier model. For example, deploying the MySQL database at MEC in our example scenario would remove the need for a PC computer that was used for running the authentication engine (AE) nanoservice, including the database.

To optimize the service deployment, our nanoEdge concept defines that the API gateway keeps track on all available physical resources within proximity, and passes this information to the orchestrator, making it able to deploy nanoservice images from a service registry to different nodes based on their capabilities. However, this resource-matching functionality was not implemented in the PoC and therefore remains as future work.

For simplicity, our PoC example scenario was implemented with one nanoservice per node policy. However, the proposed nanoEdge model promotes deploying several nanoservices to a single node if the device provides enough hardware resources for it. Therefore, as future work, it would be useful to measure how deployment of several containers on a same device would affect the computational resource requirements, service deployment and initiation time, and runtime performance.

As important future work, we also see utilizing intelligent AI/ML algorithms to optimize the service orchestration considering all layers of the three-tier cloud computing architecture. Edge computing on local and access network levels bring two new computational tiers to cloud computing, between the datacenter and end-devices. By e.g. moving some functions from datacenters to MEC or local edge hosts, cloud systems would better serve applications requiring low latency and high reliability while saving computational and networking resources at core networks and datacenters.

Finally, an important avenue for further research is to study how to establish trust between nanoservice providers and users with e.g. distributed trust systems such as Blockchain. This is important to prevent e.g. distribution of maliciously operating nanoservice images in global service registries.

VI. CONCLUSION

In this article we propose a novel service model, called nanoEdge, where nodes collaboratively provide for the services the needed management, processing, storage, interfaces, and security functions, without relying too much on centralized servers. The main contributions of this article are the introduction of the novel nanoEdge conceptual model, its proof-of-concept (PoC) implementation, and a feasibility evaluation. With the evaluation, we demonstrate that a virtualized local service can be deployed and run in a distributed manner with decent performance using Docker-based nanoservices deployed to SBC devices and that the simplest virtual functions can be deployed to even more resource-constrained microcontroller devices using mobile agent technology.

We see the proposed model as a significant part of future IoE scenarios where users interact with local and global digital services mostly through smart surroundings. To support this development, service architectures need to provide virtualized, on-demand, service composition based on the hardware and software resources near the user. With the help of suitable service architectures, such as our nanoEdge, the realization of this ubiquitous vision may be even closer than we imagine. Our everyday surroundings already now include a plethora of devices with hardware and networking capacity comparable with the devices used in our PoC and therefore potential devices for hosting nanoservices.

The future work includes updating the PoC with features that are currently missing, such as an API for inter-service and external access and a resource-matching algorithm with the cloud-based nanoservice registry for universal service optimization. Further measurements are also needed, e.g. to study how several containers deployed on the same device would affect the computational resource requirements and runtime performance. The proposed model also enables developing intelligent AI/ML algorithms to optimize the service orchestration considering the features and capacity of all three layers of the three-tier cloud computing architecture. Finally, an important avenue for further research is to study how to establish trust between different stakeholders using distributed trust systems such as Blockchain.

ACKNOWLEDGMENT

The authors would like to thank Mr. Jude Okwuibe for his assistance in writing the article. This work has been performed under the framework of MEC-AI, Industrial Edge, WiFiUS Massive IoT and 6Genesis Flagship projects.

REFERENCES

[1] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, and M. Zaharia, "A view of cloud computing," *Commun. ACM*, vol. 53, no. 4, pp. 50–58, Apr. 2010. doi: 10.1145/1721654.1721672.

[2] J. Zhou, T. Leppänen, E. Harjula, M. Ylianttila, T. Ojala, C. Yu, H. Jin, and L. T. Yang, "Cloudthings: A common architecture for integrating the Internet of Things with cloud computing," in *Proc. IEEE 17th Int. Conf. Comput. Supported Cooperat. Work Design (CSCWD)*, Whistler, BC, Canada, Jun. 2013, pp. 651–657.

[3] M. H. Miraz, M. Ali, P. S. Excell, and R. Picking, "A review on Internet of Things (IoT), Internet of everything (IoE) and Internet of nano Things (IoNT)," in *Proc. Internet Technol. Appl. (ITA)*, Wrexham, U.K., Sep. 2015, pp. 219–224.

[4] K. Crisler, M. Anneroth, A. Aftelak, and P. Pulil, "The human perspective of the wireless world," *Comput. Commun.*, vol. 26, no. 1, pp. 11–18, Jan. 2003. doi: 10.1016/S1403-3664(02)00114-7.

[5] I. Ahmad, T. Kumar, M. Liyanage, M. Ylianttila, T. Koskela, T. Braysy, A. Anttonen, V. Pentikinen, J.-P. Soininen, and J. Huusko, "Towards gadget-free Internet services: A roadmap of the naked world," *Telematics Inform.*, vol. 35, no. 1, pp. 82–92, Apr. 2018. doi: 10.1016/j.tele.2017.09.020.

[6] J. Gubbi, R. Buyya, S. Marusic, and M. Palaniswami, "Internet of Things (IoT): A vision, architectural elements, and future directions," *Future Gener. Comput. Syst.*, vol. 29, no. 7, pp. 1645–1660, Sep. 2013. doi: 10.1016/j.future.2013.01.010.

[7] L. Atzori, A. Iera, and G. Morabito, "The Internet of Things: A survey," *Comput. Netw.*, vol. 54, no. 15, pp. 2787–2805, Oct. 2010. doi: 10.1016/j.comnet.2010.05.010.

[8] J. A. Stankovich, "Research directions for the Internet of Things," *IEEE Internet Things J.*, vol. 1, no. 1, pp. 3–9, Feb. 2014. doi: 10.1109/JIOT.2014.2312291.

[9] P. Porambage, C. Schmitt, P. Kumar, A. Gurtov, and M. Ylianttila, "PauthKey: A pervasive authentication Protocol and key establishment scheme for wireless sensor networks in distributed IoT applications," *Int. J. Distrib. Sensor Netw.*, vol. 10, no. 7, Jul. 2014, Art. no. 357430. doi: 10.1155/2014/357430.

[10] I. F. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci, "Wireless sensor networks: A survey," *Comput. Netw.*, vol. 38, no. 4, pp. 393–422, Mar. 2002. doi: 10.1016/S1389-1286(01)00302-4.

[11] M. R. Palattella, M. Dohler, A. Grieco, G. Rizzo, J. Torsner, T. Engel, and L. Ladid, "Internet of Things in the 5G era: Enablers, architecture, and business models," *IEEE J. Sel. Areas Commun.*, vol. 34, no. 3, pp. 510–527, Mar. 2016. doi: 10.1109/JSAC.2016.2525418.

[12] Y. Chao Hu, M. Patel, D. Sabella, N. Sprecher, and V. Young, "Mobile edge computing a key technology towards 5G," ETSI, Sophia Antipolis, France, ETSI White Paper 11, Sep. 2015. [Online]. Available: https://www.etsi.org/images/files/ETSIWhitePapers/etsi_wp11_mec_a_key _technology_towards_5g.pdf

[13] T. X. Tran, A. Hajisami, P. Pandey, and D. Pompili, "Collaborative mobile edge computing in 5G networks: New paradigms, scenarios, and challenges," *IEEE Commun. Mag.*, vol. 55, no. 4, pp. 54–61, Apr. 2017. doi: 10.1109/MCOM.2017.1600863.

[14] J. Aikio, V. Pentikäinen, J. Häikiö, J. Häkkilä, and A. Colley. (2016). *On the Road to Digital Paradise: The Naked Approach*. [Online]. Available: https://nakedapproach.demoshelsinki.fi/wp-content/uploads/sites/3/2016/08/NA-concept-book.pdf

[15] T. Kumar, P. Porambage, I. Ahmad, M. Liyanage, E. Harjula, and M. Ylianttila, "Securing gadget-free digital services," *Computer*, vol. 51, no. 11, pp. 66–77, Nov. 2018. doi: 10.1109/MC.2018.2876017.

[16] I. Khan, F. Belqasmi, R. Glitho, N. Crespi, M. Morrow, and P. Polakos, "Wireless sensor network virtualization: A survey," *IEEE Commun. Surveys Tuts.*, vol. 18, no. 1, pp. 553–576, 1st Quart., 2015. doi: 10.1109/COMST.2015.2412971.

[17] P. Lubomski, A. Kalinowski, and H. Krawczyk, "Multi-level virtualization and its impact on system performance in cloud computing," in *Proc. Int. Conf. Comput. Netw.*, P. Gaj, A. Kwiecień, and P. Stera, Eds. Cham, Switzerland: Springer, 2016, pp. 247–259.

[18] S. Choy, B. Wong, G. Simon, and C. Rosenberg, "The brewing storm in cloud gaming: A measurement study on cloud to end-user latency," in *Proc. 11th Annu. Workshop Netw. Syst. Support Games (NetGames)*, Venice, Italy, Nov. 2012, Art. no. 2.

[19] M. Liyanage, I. Ahmad, A. B. Abro, A. Gurtov, and M. Ylianttila, *A Comprehensive Guide to 5G Security*. Hoboken, NJ, USA: Wiley, 2018.

[20] A.-R. Sadeghi, C. Wachsmann, and M. Waidner, ''Security and privacy challenges in industrial Internet of Things,'' in *Proc. 52nd ACM/EDAC/IEEE Design Autom. Conf. (DAC)*, San Francisco, CA, USA, Jun. 2015, pp. 1–6.

[21] P. Kasinathan, C. Pastrone, M. A. Spirito, and M. Vinkovits, ''Denial-of-service detection in 6LoWPAN based Internet of Things,'' in *Proc. IEEE 9th Int. Conf. Wireless Mobile Comput., Netw. Commun. (WiMob)*, Lyon, France, Oct. 2013, pp. 600–607.

[22] K. Kang, Z.-B. Pang, and C. Wang, ''Security and privacy mechanism for health Internet of Things,'' *J. China Universities Posts Telecommun.*, vol. 20, no. 2, pp. 64–68, Dec. 2013. doi: 10.1016/S1005-8885(13)60219-8.

[23] M. Liyanage, J. Salo, A. Braeken, T. Kumar, S. Seneviratne, and M. Ylianttila, ''5G Privacy: Scenarios and solutions,'' in *Proc. IEEE 5G World Forum (5GWF)*, Santa Clara, CA, USA, Jul. 2018, pp. 197–203.

[24] V. Ramani, T. Kumar, A. Bracken, M. Liyanage, and M. Ylianttila, ''Secure and efficient data accessibility in blockchain based healthcare systems,'' in *Proc. IEEE Global Commun. Conf. (GLOBECOM)*, Abu Dhabi, UAE, Dec. 2018, pp. 206–212.

[25] P. G. Lopez, A. Montresor, D. Epema, A. Datta, T. Higashino, A. Iamnitchi, M. Barcellos, P. Felber, and E. Riviere, ''Edge-centric computing: Vision and challenges,'' *ACM SIGCOMM Comput. Commun. Rev.*, vol. 45, no. 5, pp. 37–42, Oct. 2015. doi: 10.1145/2831347.2831354.

[26] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu, ''Edge computing: Vision and challenges,'' *IEEE Internet Things J.*, vol. 3, no. 5, pp. 637–646, Oct. 2016. doi: 10.1109/JIOT.2016.2579198.

[27] A. Reznik, R. Arora, M. Cannon, L. Cominardi, W. Featherstone, R. Frazao, F. Giust, S. Kekki, A. Li, D. Sabella, C. Turyagyenda, and Z. Zheng, ''Developing software for multi-access edge computing,'' ETSI, Sophia Antipolis, France, ETSI White Paper 20, Sep. 2017. [Online]. Available: https://www.etsi.org/images/files/ETSIWhitePapers/etsi_wp20_MEC_SoftwareDevelopment_FINAL.pdf

[28] P. Mach and Z. Becvar, ''Mobile edge computing: A survey on architecture and computation offloading,'' *IEEE Commun. Surveys Tuts.*, vol. 19, no. 3, pp. 1628–1656, 3rd Quart., 2017. doi: 10.1109/COMST.2017.2682318.

[29] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli, ''Fog computing and its role in the Internet of Things,'' in *Proc. 1st Workshop Mobile Cloud Comput. (MCC)*, Helsinki, Finland, Aug. 2012, pp. 13–16.

[30] F. Bonomi, R. Milito, P. Natarajan, and J. Zhu, ''Fog computing: A platform for Internet of Things and analytics,'' in *Big Data and Internet of Things: A Roadmap for Smart Environments*, N. Bessis, and C. Dobre, Eds. Berlin, Germany: Springer, 2014, pp. 169–186.

[31] P. Porambage, J. Okwuibe, M. Liyanage, M. Ylianttila, and T. Taleb, ''Survey on multi-access edge computing for Internet of Things realization,'' *IEEE Commun. Surveys Tuts.*, vol. 20, no. 4, pp. 2961–2991, 1st Quart., 2018. doi: 10.1109/COMST.2018.2849509.

[32] R. Morabito, ''Virtualization on Internet of Things edge devices with container technologies: A performance evaluation,'' *IEEE Access*, vol. 5, pp. 8835–8850, 2017. doi: 10.1109/ACCESS.2017.2704444.

[33] E. Harjula, T. Mekonnen, M. Komu, P. Porambage, T. Kauppinen, and J. Kjällman, and M. Ylianttila, ''Energy efficiency in wireless multimedia sensor networking: Architecture, management and security,'' in *Greening Video Distribution Networks*, A. Popescu, Ed. Cham, Switzerland: Springer, 2018, pp. 133–157.

[34] D. A. Celesti, D. Mulfari, M. Fazio, M. Villari, and A. Puliafito, ''Exploring container virtualization in IoT clouds,'' in *Proc. 2nd Smart Comput. (SMARTCOMP)*, Saint Louis, MO, USA, May 2016, pp. 1–6.

[35] C. M. Aderaldo, N. C. Mendonça, C. Pahl, and P. Jamshidi, ''Benchmark requirements for microservices architecture research,'' in *Proc. 1st Int. Workshop Establishing Community-Wide Infrastruct. Archit.-Based Softw. Eng. (ECASE)*, Buenos Aires, Argentina, May 2017, pp. 8–13.

[36] C. Esposito, A. Castiglione, and K.-R. Choo, ''Challenges in delivering software in the cloud as microservices,'' *IEEE Cloud Comput.*, vol. 3, no. 5, pp. 10–14, Sep./Oct. 2016. doi: 10.1109/MCC.2016.105.

[37] M. Villamizar, O. Garcés, H. Castro, M. Verano, L. Salamanca, R. Casallas, and S. Gil, ''Evaluating the monolithic and the microservice architecture pattern to deploy Web applications in the cloud,'' in *Proc. 10th Comput. Colombian Conf. (CCC)*, Bogotá, Colombia, Sep. 2015, pp. 583–590.

[38] C. Pahl and P. Jamshidi, ''Microservices: A systematic mapping study,'' in *Proc. 6th Int. Conf. Cloud Comput. Services Sci.*, Rome, Italy, Apr. 2015, pp. 1–10.

[39] M. Amaral, J. Polo, D. Carrera, I. Mohomed, M. Unuvar, and M. Steinder, ''Performance evaluation of microservices architectures using containers,'' in *Proc. 14th Int. Symp. Netw. Comput. Appl. (NCA)*, Cambridge, MA, USA, Sep. 2015, pp. 27–34.

[40] B. Butzin, F. Golatowski, and D. Timmermann, ''Microservices approach for the Internet of Things,'' in *Proc. 21st Int. Conf. Emerg. Technol. Factory Autom. (ETFA)*, Berlin, Germany, Sep. 2016, pp. 1–6.

[41] P. Angin and B. Bhargava, ''An agent-based optimization framework for mobile-cloud computing,'' *J. Wireless Mobile Netw., Ubiquitous Comput., Dependable Appl.*, vol. 4, no. 2, pp. 1–17, Jun. 2013. doi: 10.22667/JOWUA.2013.06.31.001.

[42] T. Leppänen, J. Riekki, M. Liu, E. Harjula, and T. Ojala, ''Mobile agents-based smart objects for the Internet of Things,'' in *Internet Things Based Smart Objects*, G. Fortino and P. Trunfio Eds. Cham, Switzerland: Springer, 2014, pp. 29–48.

[43] E. Harjula, T. Leppänen, T. Ojala, and M. Ylianttila, ''ADHT: Agent-based DHT architecture for constrained devices,'' in *Proc. IEEE Global Commun. Conf. (GLOBECOM)*, Austin, TX, USA, Dec. 2014, pp. 2763–2769.

[44] T. Leppänen, A. Heikkinen, A. Karhu, E. Harjula, J. Riekki, and T. Koskela, ''Augmented reality Web applications with mobile agents in the Internet of Things,'' in*Proc. 8th Int. Conf. Next Gener. Mobile Apps, Services Technol. (NGMAST)*, Oxford, U.K., Sep. 2014, pp. 54–59.

[45] T. Leppänen, I. S. Milara, J. Yang, J. Kataja, and J. Riekki, ''Enabling user-centered interactions in the Internet of Things,'' in *Proc. IEEE Int. Conf. Syst., Man, Cybern. (SMC)*, Budapest, Hungary, Oct. 2016, pp. 1537–1543.

[46] C. Hewitt, P. Bishop, and R. Steiger, ''A universal modular ACTOR formalism for artificial intelligence,'' in *Proc. 3rd Int. Joint Conf. Artif. Intell. (IJCAI)*, Stanford, CA, USA, Aug. 1973, pp. 235–245.

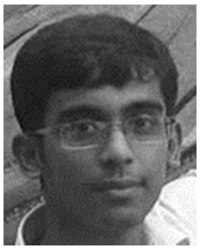
[47] A. M. Haubenwaller and K. Vandikas, ''Computations on the edge in the Internet of Things,'' *Procedia Comput. Sci.*, vol. 52, pp. 29–34, Jan. 2015. doi: 10.1016/j.procs.2015.05.011.

[48] F. Rossi, P. Van Beek, and T. Walsh, *Handbook of Constraint Programming*. Amsterdam, The Netherlands: Elsevier, 2006.

[49] Z. Shelby, K. Hartke, and C. Bormann, *The Constrained Application Protocol (CoAP)*, document IETF RFC7252, 2014. [Online]. Available: https://tools.ietf.org/html/rfc7252

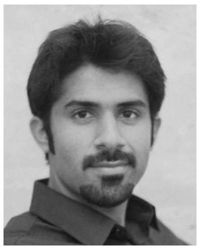
ERKKI HARJULA received the M.Sc. degree in computer engineering and the D.Sc. degree in communications engineering from the University of Oulu, Finland, in 2007 and 2016, respectively, where he is currently a Postdoctoral Researcher and a Project Manager with the Centre for Wireless Communications Research Group. He was with the Center for Internet Excellence, University of Oulu, from 2013 to 2015, and the MediaTeam Research Group, University of Oulu, from 2000 to 2014. He has also visited Columbia University, New York, NY, USA, from 2008 to 2009, as a Researcher. He is a coauthor of over 50 international peer-reviewed scientific articles on the mobile and IoT systems, edge computing, distributed systems, and energy efficiency.

PEKKA KARHULA received the M.Sc. degree in information technology from the University of Jyväskylä, Finland, in 2016. He is currently pursuing the Ph.D. degree with the University of Oulu. He is also a Research Scientist with VTT Technical Research Centre of Finland Ltd., where his current research topics include 5G and beyond communications. He was a Visiting Scholar with Columbia University, New York, NY, USA, for nine months, in 2018 and 2019. His research interests include the IoT, wearables, code mobility, and edge computing.



JOHIRUL ISLAM received the B.Sc. degree in information and communication technology from Mawlana Bhashani Science and Technology University, Bangladesh, in 2014, and the M.Sc. degree in wireless communication engineering from the University of Oulu, Finland, in 2019. He is currently a Research Assistant with the Center for Wireless Communication Research Group, University of Oulu. His research interests include the IoT, cloud and edge computing, and virtualization technologies for intelligent environment.

TEEMU LEPPÄNEN received the D.Sc. degree from the University of Oulu, Finland, in 2018, where he is currently a Research Scientist with the Center for Ubiquitous Computing. His research interests include the IoT edge computing real-world infrastructures with a focus on distributed AI methods that target energy efficiency and human–machine interactions. He visited the Institute of Industrial Science, The University of Tokyo, Japan, from 2012 to 2013. He has authored over 40 peer-reviewed articles in scientific journals, research books, and international conferences, where he has received three best paper awards.



AHSAN MANZOOR received the B.Sc. degree in computer software engineering from the Ghulam Ishaq Khan Institute, Pakistan, in 2014, and the M.Sc. degree from the University of Oulu, Finland, in 2017, where he is currently pursuing the Ph.D. degree. He was a Research Assistant with the Centre for Wireless Communications, University of Oulu. He is also a Blockchain Research Developer with Rovio Entertainment, Espoo, Finland. His research interests include Blockchain, the Internet of Things, and ubiquitous computing.

MADHUSANKA LIYANAGE received the D.Sc. degree in communication engineering from the University of Oulu, Oulu, Finland. From 2011 to 2012, he was a Research Scientist with the I3S Laboratory, and Inria, Sophia Antipolis, France. He was a Visiting Research Fellow with the Computer Science and Engineering Department, The University of Oxford, Data61, CSIRO, Sydney Australia, Infolabs21, Lancaster University, U.K., and the Computer Science and Engineering Department, The University of New South Wales, from 2016 to 2018. He is currently a Marie Curie Fellow with the School of Computer Science, University College Dublin, Ireland. He is also an Adjunct Professor with the University of Oulu. He is a coauthor of over 50 publications, including two edited books with Wiley. His research interests include SDN, the IoT, Blockchain, mobile, and virtual network security. He is also a member of ICT.

JAGMOHAN CHAUHAN received the Ph.D. degree in electrical engineering and telecommunications from The University of New South Wales (UNSW). He is currently a Postdoctoral Researcher with the Mobile Systems Group, University of Cambridge. His research interests include mobile sensing, mobile health, and applied machine learning.

TANESH KUMAR received the B.E. degree in computer engineering from the National University of Sciences and Technology (E&ME), Pakistan, in 2012, and the M.Sc. degree in computer science from South Asian University, New Delhi, India, in 2014. He is currently pursuing the Ph.D. degree with the Centre for Wireless Communications Research Group, University of Oulu, Finland, where he is also a Research Scientist. He has coauthored over 40 peer-reviewed scientific articles. His current research interests include security and privacy in the IoT/smart environments, edge computing, and blockchain.

IJAZ AHMAD received the B.Sc. degree in computer systems engineering from the University of Engineering and Technology, Peshawar, Pakistan, and the M.Sc. and D.Sc. degrees in wireless communications engineering from the University of Oulu, Finland, in 2012 and 2018, respectively. He was a Researcher and a Postdoctoral Researcher with the Center for Wireless Communications, Oulu, Finland. He is currently a Research Scientist with the VTT Technical Research Centre of Finland. He has contributed over 40 publications, including high impact factor journal articles, conference papers, and book chapters. His research interests include SDN, SDN-based mobile networks, wireless networks, network security, and network load balancing.

MIKA YLIANTTILA (SM'08) received the Ph.D. degree in communications engineering from the University of Oulu, in 2005. He was the Director of the Center for Internet Excellence, from 2012 to 2015, the Vice Director of the MediaTeam Oulu Research Group, from 2009 to 2011, and a Professor (pro tem) of computer science and engineering and the Director of the Information Networks Study Program, from 2005 to 2010. He is currently a full-time Associate Professor (tenure track) with the Centre for Wireless Communications (CWC), Faculty of Information Technology and Electrical Engineering (ITEE), University of Oulu, Finland. He is leading a research team and is the Director of Communications Engineering Doctoral Degree Program. He has coauthored more than 150 international peer-reviewed articles. His research interests include edge computing, network security, network virtualization, and software-defined networking. He is also an Editor of *Wireless Networks* journal.