Edge-based Microservices Architecture for Internet of Things: Mobility Analysis Case Study

Teemu Leppänen[1], Claudio Savaglio[2], Lauri Lovén[1], Tommi Järvenpää[1], Rouhollah Ehsani[1], Ella Peltonen[1], Giancarlo Fortino[2] and Jukka Riekki[1]

[1]Center for Ubiquitous Computing, University of Oulu, Finland

[2]Department of Informatics, Modeling, Electronics and Systems, University of Calabria, Italy

Email: {first.last}@oulu.fi, {first.last}@unical.it

***Abstract*—In this paper, we describe how the microservices paradigm can be used to design and implement distributed edge services for Internet of Things applications. As a case study, traditionally monolithic user mobility analysis service is developed, with distributed and extendable microservices, for the standardized ETSI MEC system reference architecture. In each of the edge system three tiers, microservices implement the service logic with components for movement trace analysis, movement prediction and visualization of the results. The distributed service is implemented with Docker containers and evaluated on real-world settings with low capacity edge servers and real user mobility data. The results show that the edge promise of low latency can be met in such as implementation. The integration of a software development technology with a standardized edge system provides solid background for further development.**

***Index Terms*—Multi-access Edge Computing, Service computing, 5G, Mobility, Container**

I. Introduction

Edge computing is seen as the next step towards future Internet of Things (IoT) systems [1]. Edge computing leverages cloud resources, i.e. application-specific computations and data, to the network infrastructure devices in close proximity of the data producing IoT devices. This approach reduces both the volume of data transmitted in the core network and the application execution latencies, as the data can be processed already at the edge. These benefits, with the capability of offloading mobile applications from the device to the edge, improve the overall Quality of Service (QoS) experienced by the mobile users.

Still, the operational environment of the IoT devices is dynamic, with intermittent connectivity and low bandwidth, where edge service availability can not be guaranteed. This makes the orchestration and management of such distributed resources across the networks challenging, where the heterogeneous system components, including the IoT devices, have varying capabilities for computation and communication. Moreover, user mobility and resulting unpredictable actions are another reason for inherent system dynamicity. With location- and context-awareness, edge systems can better control resource orchestration in such an environment and optimize the application execution at the edge.

To deploy IoT applications and application-specific services to the edge, virtualization technologies have appeared [2]. The idea is to distribute self-contained applications as virtual machines (VM) to the servers at the edge. However, such VMs have typically large overhead, i.e. include a guest operating system, and therefore resource consuming to deploy, instantiate, maintain and relocate. Recently, lightweight virtualization technologies, i.e. containers, allow partitioning of applications into low overhead packages, e.g. without the operating system. This further enables deployment of application packages to the edge and IoT devices with low computational capabilities [2].

However, partitioning of monolithic cloud applications into distributed packages and their deployment into hierarchical IoT system architectures is not straightforward. First, the application-specific requirements, e.g. performance and QoS, have to be met with the available resources in the multi-tenant edge infrastructure. Second, novel modeling, design and programming paradigms are needed, that unify the different properties of IoT system architecture layers for distributed software development. A recent approach, microservices [3, 4], modularize the applications and services at the individual process level, that can be developed in isolation and deployed as containers to the edge. Benefits of microservices are seen in isolation and maintainability in their development and autonomy and scalability in their operation. The low overhead increases scalability and further facilitates the optimization of package deployments for service provisioning, e.g. to provide low latency in response to the user mobility.

In this paper, we present a case study of user mobility analysis as an edge service, implemented with distributed microservices that complement each other as the service logic. The presented microservices-based service design and implementation particularly follows the Multi-access Edge Computing (MEC) specifications, currently under standardization by the European Telecommunications Standards Institute (ETSI). The integration of a software development solution with a standardized edge system provides a solid background for further developments. The presented microservices implementation is then evaluated in real world settings.

The rest of the paper is organized as follows. In Section 2, the background in microservices for edge computing, user mobility analysis and service design is presented. The microservice-based design and implementation of the service logic are presented in Section 3. The resulting microservice implemented is evaluated in Section 4. In Section 5, we discuss the findings of the paper. Section 6 concludes the paper.

II. Background

In this work, we utilize the ETSI MEC reference architecture [5] to exemplify an edge computing system. MEC is currently under standardization by ETSI. The standards cover the system architecture and components, the functionality of the components and their interactions and provide a set of application programming interfaces (API). The APIs realize capabilities for top-down MEC system orchestration and application life-cycle management and the bottom-up retrieval of information of the system state and resource usage. In MEC, the APIs and interactions netween system components follow the Representational State Transfer (REST) architectural principles. The standardization does not address implementation details of the system components. No implementation of the reference architecture exists yet, but a set of the standardized APIs is under development.

The ETSI MEC system reference architecture is illustrated in Figure 1, with the MEC system components depicted in white, the user and application-specific components in gray and the microservices, as described in this paper, in black. Application requests from the user equipment (UE) are received through a portal that authenticates the requests. The system Orchestrator is the sole authority for system orchestration and resource provisioning for the applications. The Orchestrator is expected to have complete real-time view over the system state. With the help of system services, it validates the UE requests with regard to available resources and services in hosts and manages the request handling, e.g. application instantiation, relocation or termination. The Platform Managers manage the operation of a set of edge hosts under their control. With the Orchestrator, the managers configure the hosts in the platform to fulfill the application requests, e.g. launching the virtualized application packages and their required services in the selected hosts. The actual package deployment, e.g. microservices, is handled by the virtualization infrastructure with components in the platforms and hosts. The platform managers are responsible for collecting and sending information of their state to the Orchestrator that uses it in the system orchestration. The edge hosts are responsible for managing the execution of the assigned applications and services, with the real-time knowledge of local environment characteristics, which they receive through the MEC system APIs. Examples of such APIs include the Location API, that can be used for device location tracking, and Radio Network Information API that provides information about the measured network conditions.

A. Microservices

Server-side applications have been traditionally developed as monoliths [3] that are difficult to maintain and evolve, due to the internal complexities of the software. Their deployment is resource consuming, which limits their feasibility and scalability as a solution for IoT systems. Microservices have emerged to address these issues through the modularization of the application logic into a set of independent processes, which can be deployed as loosely-coupled service components into the hosts [3]. The resulting distributed architecture becomes more autonomous and flexible, as these components can be developed and deployed independently. Scalability can be increased through a variety of deployment options for small-scale components. Therefore, microservices are a promising paradigm for design, implementation and deployment for distributed IoT services [4]. Challenges in their operation include the orchestration of a set of microservices and in handling possible down times and cascading of faults. However, these issues can be addressed with design patterns [4].

Currently, there is no established practise for microservice development, as the paradigm has emerged from good practises in software business [4]. Each microservice can be developed in isolation with a variety of programming paradigms and available technologies that address the particular problem. For distributed services, microservices have adopted the service-oriented architecture paradigm, while communicating through lightweight APIs, e.g. the REST paradigm [4].

Containers have emerged as an implementation technology for microservices, providing a lightweight alternative to virtualize cloud applications particularly at the edge, where VM deployment is resource consuming [2]. A container is a self-contained application-specific functionality, and possibly its data, in an executable form that can be deployed as a single package. Typically, containers include Web servers to facilitate interactions and collaboration with their execution environment. The performance of containers has been shown similar to the bare-metal application execution in servers, where the instantiation of containers is faster than the corresponding VMs [6]. In this, work we utilize Docker containers to build the microservices for the edge system.

ETSI MEC [7] sees microservices as a tool for partitioning monolithic applications into a set of loosely-coupled distributed components. Such an architecture is expected to facilitate dynamic tailoring during the application execution. However, increased modularization may also increase the orchestration and system management load. In MEC, UE's connect through gateways to the microservices running in the edge host.

B. Service modeling and design

According to the different definitions in the state-of-the-art, an IoT service can be seen as an interface enabling contextualized interactions among system components. To exploit the potential and to speed up development in IoT systems, service modeling as a fundamental step and has been faced from different perspectives. As reported in [8], IoT services have been modeled according to workflow specifications, business processes, and modeling languages and ontologies. While the first three are operational approaches, aiming to support the service validation, verification and simulation, the last one provides high-level descriptive representations of both functional and non-functional properties of services. Further, software agent-based computing has been reported for the

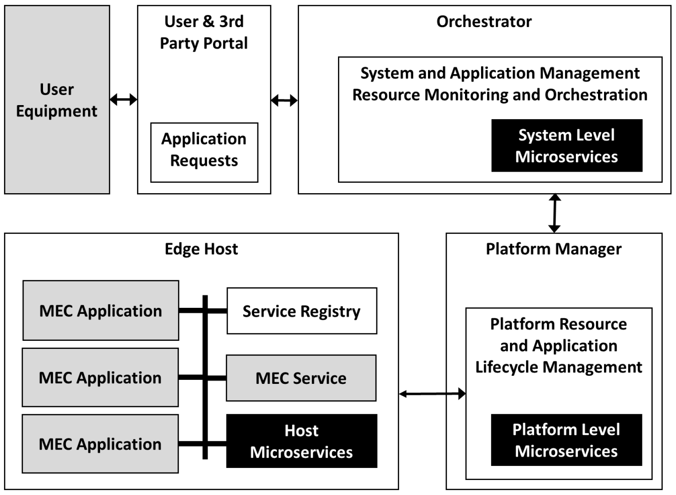


Fig. 1: Simplified MEC system architecture with hierarchical microservice deployment.

modeling and design of MEC and, in general, edge services in [9, 10].

In particular, the opportunistic IoT service model [11], provides 1) a service profile that specifies functionality and its static and dynamic properties, e.g. description, category, dependencies, QoS parameters and provisioning rules and constraints, and 2) a service model that details the inputs and their preconditions, outputs and effects of the processes that implement the service logic. With this information, a service can be accurately described, automatically composed, discovered and consumed. Moreover, such a comprehensive, flexible, context-aware and technology-agnostic modeling fits different IoT application scenarios and fosters service design and implementation independently from a specific computing paradigm, as exemplified for edge and aggregate computing in [11, 12].

C. *User mobility analysis*

Mobility analysis states, straightforwardly, the tracking and prediction of user movement trajectories to optimize the system operation [13, 14]. Applications for mobility analysis include location-based services for IoT and edge computing, where the analysis aims to find patterns and point-of-interests in the user trajectories.

Traditionally, mobility prediction is often modeled as the connectivity probability according to the user movement [13]. Such large-scale data analysis of combined user device connections and applications, e.g. social media check-ins [14], requires both wide data sets and back-end services in clouds with possible privacy violations. With currently available location-aware UEs, accurate tracking services have become available, but such approaches have been found resource consuming [15].

In edge computing, mobility prediction aims for optimizing both the communication and edge application performance in the opportunistic IoT environment in large scale [13]. For example, edge-based location- and context-aware crowdsensing solutions have been developed through microservices [16], mobility enhanced Web-integrated smart objects [10] and with autonomous software agents [17].

III. User mobility analysis for MEC

As a case study, we design and implement a microservice-based distributed Mobility MicroService (MMS) as a MEC service. The purpose of MMS is to provide refined information about user trajectories with accuracy of one second, based on UE connections to the edge system. In MEC, the trajectories are identified with the information provided by the Location API.

The three-tier architecture of the MMS is illustrated in the Figure 1, where the depicted microservices represent the MMS instances. At the system level, mobility information is beneficial for the orchestration of the whole MEC system, aiming to maintain system and application performance with regard to the current UE locations and available resources. On the platform level, mobility information is useful for managing the deployment and execution of applications and services in its hosts, e.g. application component placement and relocation in response to the UE mobility. At the hosts, the information is used to maintain the expected QoS for the UEs.

Such a service architecture requires data processing in all tiers, which distributes the service execution load across the system and increases service responsiveness for different purposes. This is useful to address the opportunistic nature of edge systems with a service resolution metric, e.g. accuracy vs latency [18]. Placement of an MMS instance to the host level facilitates low latency, but with limited accuracy due to locality, close to the UEs and their corresponding edge applications. It is apparent that system and platform level MMS needs to process much more data than the host level MMS, which limits the possible component placement scenarios.

Each MMS instance collects UE session data from the infrastructure devices, e.g. base stations or access points (AP), under the control of its host. The geographical locations of the infrastructure devices are known, thus can be used as coarse estimations of the locations of the UEs. Each instance maintains a multidimensional data array, which contains the number of detected UE movement actions, e.g. handovers between origin and destination APs. The array is further divided on each observed time scale (minute, hour, day of week, day, month, year). The resulting array is thus 8-dimensional (O $\times$ D $\times$ min $\times$ h $\times$ wd $\times$ d $\times$ m $\times$ y), where the cell values are cumulative count of movement actions in the time scale. As an array, the data are computationally straightforward to process along the dimensions, e.g. to summarize movement actions in each AP, host, geographical area, etc., on different time scales. Although the actual data analysis algorithm is the same in all tiers, the number of dimensions, that each instance is capable of handling, varies according to resources and its position in the system hierarchy.

A. Microservices-based design of MMS

Today, microservices are typically designed by domain experts, with focus on the distribution and role-based encapsulation of the functionality. The preferred design and implementation model of microservices is containerization, where each functionality of the service is developed as a single container [4], leading to a distributed microservices architecture. The distributed service logic is expected to follow the service-oriented architecture principles, providing loose coupling and isolation. For edge computing, microservices need to be implemented and configured to operate atop the available virtualization technologies in compliance with the edge system services. Due to the heterogeneity of software and hardware platforms, and to extend their interoperability, a set of adapters are typically needed for integration to the edge system. The collaboration between microservices is commonly based on Web technologies [4].

The MMS instance architecture is modularized into four microservices, as illustrated in Figure 2, namely Web User Interface, Mobility Trace Analysis, Mobility Prediction and Visualization. The Web-based user interface (UI) operates as the front-end to submit trajectory or prediction queries and visualize the results in a Web browser. The Mobility Trace Analysis service collects data from the UEs and updates its mobility data accordingly. The Mobility Prediction service further analyses the data to calculate the probabilities of user movements based on the given parameters, e.g. a set of APs, by following the traditional mobility modeling [13]. The Visualization microservice compiles the map for the visualization of the data analysis results.

These MMS microservice roles are purposedly designed in a way that each requires different computational and communication resources, including interactions, data analysis and collaboration with an external Web service. Moreover, such a design exemplifies an extendable microservice in MEC, where the functionality of each microservice complements each other.

For the modeling of distributed MMS for MEC, we envision the opportunistic service modeling [8, 11] that is based on a service profile, detailing the service functionality and its properties, and service model, that details the service pro-

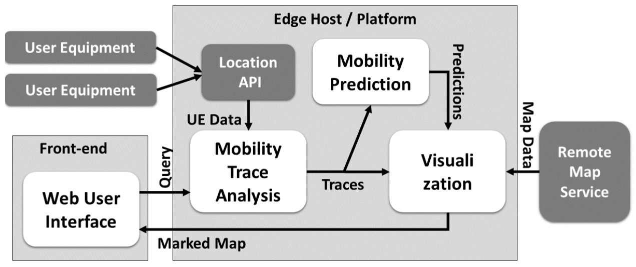


Fig. 2: MMS service instance architecture.

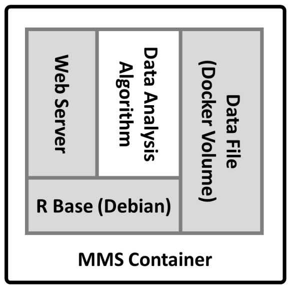


Fig. 3: MMS container internal architecture.

cesses as microservices. Such model is suitable for supporting microservice-based edge service implementation, due to the conceptual alignment of the service profile and model and the MEC service and application contextual information as defined in the MEC specifications. However, the details of MEC specifications are out of the scope of the paper.

Thus, for each MMS microservice, a MEC service configuration is created that describes the service content, the interfaces it exposes, its operational rules, the required other services and the requirements for computational power, data storage and networking. According to the configuration and available edge resources, the Orchestrator instructs the deployment and instantiation of each MMS microservice instance as an independent system component in the MEC hosts.

B. Distributed MMS implementation

Each microservice is realized as a single container, which internal architecture is illustrated in Figure 3. Alike all services on the MEC system, the microservices are expected to comply with the REST principles in their interactions with the MEC system services and each other, through a Web server. Data processing algorithms are needed to provide the service content. A file system is needed storage and sharing of data between the microservices.

The four MMS microservices are implemented as follows. Their architecture is depicted in Figure 3. A R base, based on Debian, is needed to deploy additional R software. The front-end container hosts the Web User Interface that is implemented with Shiny Web server for R. The mobility data analysis is done in the two analysis containers with R scripts that access the data with Data.Table R package. Docker Volume is mounted as a distributed file system, enabling seamless sharing of the data between microservices.

The Mobility Trace Analysis microservice identifies sets of unique movement pairs, i.e. origin and destination APs, and calculates the total amount of such actions in the data set during the given time period. The resulting data is sliced up to sets of 800 movement actions to speed up request handling with parallelized processing through the service logic. The visualization container uses Google Maps as the remote Map Service, requiring two specific R packages for map handling (ggmap and ggplot). The size of the front-end container is 1.63 GB and the size of the three other containers is 1.1 GB each. The biggest individual component is the R base (667 MB).

To replace the MEC Location API, we utilize the mobility data set provided in [19]. The data set contains the complete session data, with one second accuracy, of connected UEs in 1300 WiFi APs in the public city-wide panOULU WiFi network in Oulu, Finland, collected during the years 2007-2015. From the complete data set, the UE session data of 218 APs in the expanded city centre area (2km x 2km), during February 3rd 2015, was selected for the MMS implementation.

To utilize the microservices with MEC, service descriptions and configurations from the design phase are needed. MEC service information contains the assigned service URI, its descriptor, possible dependencies to other services, functional requirements such as latency, computational power, memory, data storage and data traffic rules. We omit this information from the implementation presented in this paper, due to lack of a real MEC implementation. This information is then used by the Orchestrator to assign and instruct on the deployment of the service into available MEC hosts, with the assistance of the platform manager.

The expected placement of the MMS microservices, as illustrated in Figure 2, is the following. Mobility Trace Analysis, Prediction and Visualization microservices operate on edge system components with sufficient computational capacity and need an access to the MEC Location API and to a remote Map Service. The Web UI microservice placement is left open, but it can be deployed for example on edge hosts, remote servers or to an external cloud platform for global access.

IV. Evaluation

To illustrate the MMS service results, Figure 4 shows the user movement patterns with corresponding probabilities from an APs to other APs at one hop distance, as identified from the example data set, during 8am - 9am in February 3rd 2015.

To evaluate the feasibility of the distributed MMS, in comparison with a monolithic single-container version, we collected the MMS service request latencies in both implementations where the functionality is the same. A service request in a MMS instance calculates the movement actions and the resulting predictions in a selected set of APs, corresponding to different geographical areas in a city and visualizes the results as a Web service content in a map that is fetched from a remote map service. The selected areas roughly correspond to the three-tiered MMS instance deployment in a MEC system. As

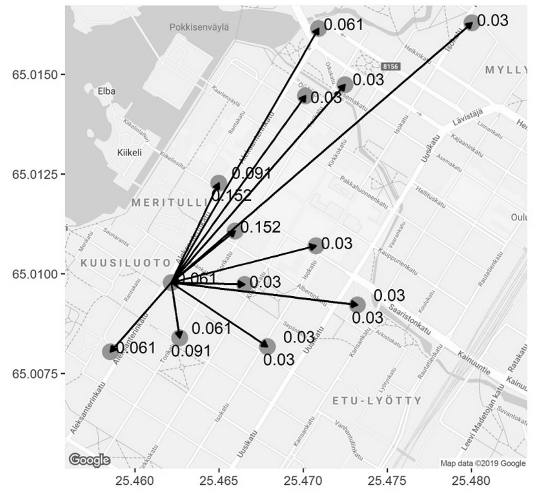


Fig. 4: Visualization of a movement prediction.

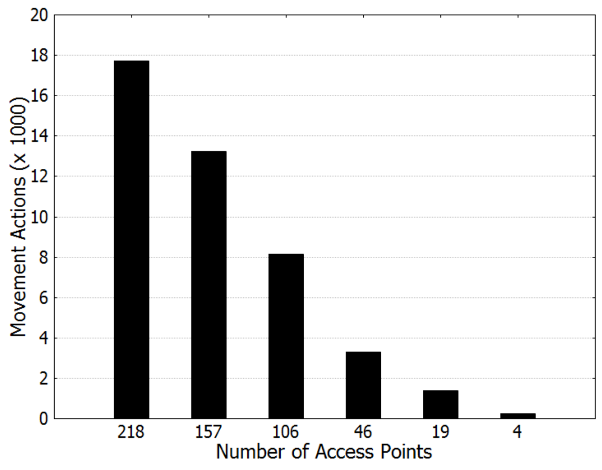
shown in Figure 5a, an area with more than 150 APs represents the MEC system that covers a city centre (about 2km x 2km). An area with 50-100 APs represents a MEC platform that covers several blocks. An area with up to 20 APs represents the area covered by a single MEC host, corresponding to a block in the city.

The MMS containers were instantiated on two computers for comparison of service request handling latencies: (1) a desktop PC (Intel i5-7600K CPU, 3.79 GHz, 16 GB RAM) and (2) an Asus R558U laptop (Intel i5-7200U CPU, 2.71 GHz, 8 GB RAM). The measured latencies are shown in Figure 5b. It can be observed that at MEC host level, such a distributed MMS handles service requests, in a small geographical area, with latencies up to a few seconds, e.g. one second in desktop PC-based host and three seconds in a Laptop-based host. The latency difference between the distributed and monolithic implementations is observed to be from a half a second to two seconds in the desktop PC, depending on the size of the controlled area. In the Laptop versions, the latency difference is about 1-3 seconds in all areas.

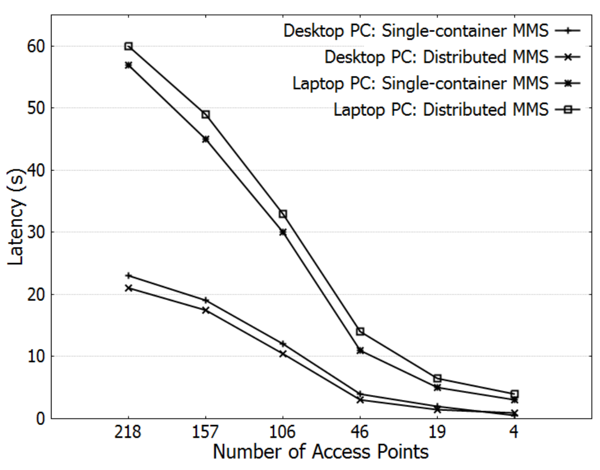
V. Discussion

In this paper, the microservices paradigm is utilized to build a modularized and distributed user mobility analysis service on an edge computing platform, exemplified by ETSI MEC. For seamless integration, the microservices are designed and implemented according to the MEC practises, e.g. comply with MEC service configurations, follow REST principles and use the MEC Location API. As demonstrated, the existing IoT service modeling and design approaches provide formalized methods and tools for realizing microservice-based edge applications. With microservices, the main design factors include computational, data and network resource requirements, dependencies on other services, deployment costs and overall distributed maintainability [4].

Container-based microservices have been demonstrated to be a feasible lightweight option for edge application imple-



(a) APs and corresponding actions in a city area



(b) Different MMS execution latencies

Fig. 5: Evaluation results of microservice-based MMS.

mentation. There is no clear definition of how "small" a microservice functionality should be [4], but each implements a function that is integrated into a complete service logic that is extendable with minimal effort in MEC. Moreover, each of the presented MMS service components can be developed and maintained in isolation and deployed independently.

The presented latency evaluation shows that such a distributed service implementation meets the low latency requirements of edge applications, even with low capacity edge hosts. The MMS implementation relied on R based software components, where some operational overhead is expected in comparison with a bare-metal implementation. Nevertheless, such measurement results are largely application- and implementation-specific, further depending on the skills of the programmers and availability edge system resources. For example, an industry edge host typically consists of server racks with tens of cores and hundreds of gigabytes of memory.

Regarding the interactions between the microservices, the latency measurements show that the selected virtualization technology does not introduce significant management overhead latency in such an application. However, network delays are to be expected when the operation is distributed across edge hosts, where typical edge infrastructure is required to have a high-speed Internet connection between the components.

VI. Conclusion

This paper demonstrated the design and implementation of an edge service logic with distributed microservices for an edge system. The integration of microservices paradigm, a lightweight virtualization technology and a standardized edge computing system provides a solid background for further developments of distributed IoT applications and services for edge computing. With the presented evaluation, the contributions of this paper support the idea of modularization of edge service logic with microservices that are deployed to the edge system components, even with low resources.

The presented user mobility analysis service is being developed with an open source-based 5G edge testbed [20] at the premises of the University of Oulu, Finland. Moreover, the development framework provided by opportunistic edge service modelling [11], large-scale edge simulation platform [12] and the real-world 5G testbed facilitates future developments of distributed microservice prototypes and their detailed evaluation.

Acknowledgements

This research is financially supported by the AI-Enhanced Mobile Edge Computing -project, funded by the Future Makers program of Jane and Aatos Erkko Foundation and Technology Industries of Finland Centennial Foundation, and by the Academy of Finland 6Genesis Flagship (grant 318927).

References

[1] Mahadev Satyanarayanan. The emergence of edge computing. *Computer*, 50(1):30–39, 2017.

[2] Claus Pahl and Brian Lee. Containers and clusters for edge cloud architectures–a technology review. In *3rd International Conference on Future Internet of Things and Cloud*, pages 379–386. IEEE, 2015.

[3] Nicola Dragoni, Saverio Giallorenzo, Alberto Lluch Lafuente, Manuel Mazzara, Fabrizio Montesi, Ruslan Mustafin, and Larisa Safina. Microservices: yesterday, today, and tomorrow. In *Present and ulterior software engineering*, pages 195–216. Springer, 2017.

[4] Björn Butzin, Frank Golatowski, and Dirk Timmermann. Microservices approach for the internet of things. In *21st International Conference on Emerging Technologies and Factory Automation*, pages 1–6. IEEE, 2016.

[5] ETSI. Multi-access Edge Computing (MEC); Framework and Reference Architecture. *ETSI GS MEC 003*, 2019.

[6] Marcelo Amaral, Jorda Polo, David Carrera, Iqbal Mohomed, Merve Unuvar, and Malgorzata Steinder. Performance evaluation of microservices architectures using

containers. In *14th International Symposium on Network Computing and Applications*, pages 27–34. IEEE, 2015.

[7] Alex Reznik et al. Developing software for multi-access edge computing. *ETSI White Paper 20*, 2017.

[8] Giancarlo Fortino, Claudio Savaglio, and Mengchu Zhou. Toward opportunistic services for the industrial internet of things. In *13th IEEE Conference on Automation Science and Engineering*, pages 825–830, 2017.

[9] Teemu Leppänen. Distributed artificial intelligence with multi-agent systems for mec. In *28th International Conference on Computer Communications and Networks (ICCCN 2019) Workshops, 1st Edge of Things: Enabling Internet of Things Ecosystems through the Edge Computing*. IEEE, 2019. [To appear].

[10] Teemu Leppänen, Claudio Savaglio, Lauri Lovén, Wilma Russo, Giuseppe Di Fatta, Jukka Riekki, and Giancarlo Fortino. Developing agent-based smart objects for iot edge computing: Mobile crowdsensing use case. In *11th International Conference on Internet and Distributed Computing Systems*, pages 235–247. Springer, 2018.

[11] Roberto Casadei, Giancarlo Fortino, Danilo Pianini, Wilma Russo, Claudio Savaglio, and Mirko Viroli. Modelling and simulation of opportunistic iot services with aggregate computing. *Future Generation Computer Systems*, 91:252–262, 2019.

[12] Claudio Savaglio, Giuseppe Campisano, Giuseppe Di Fatta, and Fortino Giancarlo. Iot services deployment over edge vs cloud systems: a simulation-based analysis. In *IEEE INFOCOM WKSHPS: Hot Topics in Social and mobile connected Smart objects*, 2019. [To appear].

[13] Yuyi Mao, Changsheng You, Jun Zhang, Kaibin Huang, and Khaled B Letaief. A survey on mobile edge computing: The communication perspective. *IEEE Communications Surveys & Tutorials*, 19(4):2322–2358, 2017.

[14] Zhiyuan Cheng, James Caverlee, Kyumin Lee, and Daniel Z Sui. Exploring millions of footprints in location sharing services. In *5th International AAAI Conference on Weblogs and Social Media*, 2011.

[15] Yohan Chon, Elmurod Talipov, Hyojeong Shin, and Hojung Cha. Mobility prediction-based smartphone energy optimization for everyday location monitoring. In *9th Conference on Embedded Networked Sensor Systems*, pages 82–95. ACM, 2011.

[16] Silvia Mirri, Catia Prandi, Paola Salomoni, Franco Callegati, Andrea Melis, and Marco Prandini. A service-oriented approach to crowdsensing for accessible smart mobility scenarios. *Mobile Information Systems*, article no. 2821680, 2016.

[17] Teemu Leppänen, José Álvarez Lacasia, Yoshito Tobe, Kaoru Sezaki, and Jukka Riekki. Mobile crowdsensing with mobile agents. *Autonomous Agents and Multi-Agent Systems*, 31(1):1–35, 2017.

[18] Min Chen, Wei Li, Giancarlo Fortino, Yixue Hao, Long Hu, and Iztok Humar. A dynamic service migration mechanism in edge cognitive computing. *ACM Transactions on Internet Technology*, 19(2):30, 2019.

[19] Vassilis Kostakos, Timo Ojala, and Tomi Juntunen. Traffic in the smart city: Exploring city-wide sensing for traffic control center augmentation. *IEEE Internet Computing*, 17(6):22–29, 2013.

[20] Juuso Haavisto, Muhammad Arif, Lauri Lovén, Teemu Leppänen, and Jukka Riekki. Open-source RANs in Practice: an Over-the-air Deployment for 5G MEC. In *28th European Conference on Networks and Communications*, 2019. [To appear].