Received 22 March 2024; revised 23 April 2024; accepted 6 May 2024. Date of publication 9 May 2024; date of current version 23 May 2024.

Digital Object Identifier 10.1109/OJCOMS.2024.3399015

AI-Driven Ground Robots: Mobile Edge Computing and mmWave Communications at Work

GIUSEPPE BARUFFA [1], ANDREA DETTI [2], LUCA RUGINI [1] (Member, IEEE), FRANCESCO CROCETTI [1], PAOLO BANELLI [1] (Member, IEEE), GABRIELE COSTANTE [1] (Member, IEEE), AND PAOLO VALIGI [1] (Member, IEEE)

[1]Department of Engineering, University of Perugia, 06125 Perugia, Italy

[2]Department of Electronic Engineering, University of Rome "Tor Vergata," 00133 Rome, Italy

CORRESPONDING AUTHOR: G. BARUFFA (e-mail: giuseppe.baruffa@unipg.it)

This work was supported in part by the University of Perugia through the Fondo di Ricerca di Base 2022 Project "Energy-Efficient Networking, Signal Processing and Communications (EFESO)"; in part by the Fondo di Ricerca di Base 2020 Project "Systems, Algorithms, and Architectures for Information Transmission and Processing in Future Telecommunications and Sensor Networks"; in part by the Ministero dell'Istruzione, dell'Università e della Ricerca (MIUR) through the PRIN 2017 Project "Liquid Edge Computing Based on Distributed Machine Learning and Millimeter-Wave Radio Access (LIQUID_EDGE);" and in part by the European Union under the Italian National Recovery and Resilience Plan (NRRP) of NextGenerationEU, Partnership on "Telecommunications of the Future," Program "RESTART" under Grant PE00000001, "Netwin" Project (CUP E83C22004640001).

ABSTRACT The seamless integration of multiple radio access technologies (multi-RAT) and cloud/edge resources is pivotal for advancing future networks, which seek to unify distributed and heterogeneous computing and communication resources into a cohesive continuum system, tailored for mobile applications. Many research projects and focused studies are proposing solutions in this area, the impact of which is undoubtedly increased by moving from theoretical and simulation studies to experimental validations. To this aim, this paper proposes a testbed architecture that combines contemporary communication and cloud technologies to provide microservice-based mobile applications with the ability to offload part of their tasks to cloud/edge data centers connected by multi-RAT cellular networks. The testbed leverages Kubernetes, Istio service mesh, OpenFlow, public 5G networks, and IEEE 802.11ad mmWave (60 GHz) Wi-Fi access points. The architecture is validated through a use case in which a ground robot autonomously follows a moving object by using an artificial intelligence-driven computer vision application. Computationally intensive navigation tasks are offloaded by the robot to microservice instances, which are executed on demand within cloud and edge data centers that the robot can exploit during its journey. The proposed testbed is flexible and can be reused to assess communication and cloud innovations focusing on multi-RAT cloud continuum scenarios.

INDEX TERMS Containerization, ground robot, millimeter wave, mobile edge computing, mobile edge learning, object detection, orchestration, smart city, software defined networking.

I. INTRODUCTION

In THE rapidly evolving landscape of urban development, the concept of smart cities has emerged as a visionary approach to address the complex challenges posed by the growing urban population. Central to the realization of smart cities is the integration of advanced technologies and intelligent systems that can improve urban living, sustainability, and efficiency. Among these technologies, ground robots driven by artificial intelligence (AI) have received significant attention due to their potential to revolutionize various aspects of urban life, from transportation and public safety to environmental monitoring and waste management. However, often (small) ground robots cannot have high computational capability on board, because of lack of space or energy, cost, etc.

Due to the ongoing deployment of 5G networks and other communication technologies, smart cities will be covered by a ubiquitous communication network that will provide low latency and high-speed Internet access [1]. The network infrastructure will be made up of numerous access

© 2024 The Authors. This work is licensed under a Creative Commons Attribution-NonCommercial-NoDerivatives 4.0 License. For more information, see https://creativecommons.org/licenses/by-nc-nd/4.0/

points (AP) or base stations (BS), operating over a wide range of frequency bands, extending from the sub-GHz at 800 MHz to the millimeter wave (mmWave) portion of the radio frequency spectrum at 24-60 GHz, where wireless spectrum is less crowded and a Gigabit-per-user throughput is expected, but radio coverage is much more critical. Along with 5G, IEEE 802.11ad is a Wi-Fi wireless communication standard designed for operation over mmWave links in the V band (at 60 GHz), with multi-GHz bandwidth channels, offering improved performance on short-range static links, thanks to beamforming and multi-gigabit access speeds [2]. IEEE 802.11ad can work indoors, but also outdoors, and can coexist with 5G networks [3], [4], [5].

Such powerful network connectivity allows computing-intensive AI tasks of robots to be offloaded to the data centers of a cloud infrastructure, which can be located at the edge of the infrastructure, i.e., near the robot for low-latency or bandwidth-intensive robotic applications, or in a central part of the cloud for other cases [6], [7], [8]. In fact, mobile edge computing has emerged as a potential solution to a number of problems faced by resource-constrained mobile devices, which require additional computing power to perform essential tasks for their operation [9], [10]. Mobile edge computing solves these problems by offering a proximity-based computing and caching resource that minimizes latency, optimizes bandwidth usage, and increases user density, especially when combined with high-speed radio access such as that offered by Wi-Fi [11] and 5G networks [12]. There are also ongoing standardization proposals, such as the multi-access edge computing (MEC) framework established by ETSI [13]. However, in this paper, we are referring to a more generic Edge Computing deployment not compliant with ETSI MEC specifications.

One of the application fields that could benefit from edge computing is computer vision (CV) for robotics applications. CV and deep machine learning are significantly expanding the capabilities and versatility of robots, enabling them to sense and understand the surrounding environment. In this field, a game changing algorithm is “You only look once” (YOLO) [14], a detection algorithm that excels at identifying and localizing objects within images with the help of convolutional neural networks. The original YOLO algorithm has undergone continuous evolution [15] and is used in various mobile robotics tasks, including real-time obstacle detection [16], [17], object tracking as they move within the robot field of view [18], [19], etc.

Regarding the control applications of ground robots (and more), in recent years, developers are focusing on splitting complex software systems into numerous microservices [20], which interact with each other through application programming interfaces (API), such as HTTP or Google remote procedure calls (gRPC), while they are dropping support for monolithic implementations [21], [22]. Resource allocation for microservice applications is more efficient than for monolithic applications, since only the overloaded microservices can be replicated, rather than the entire application. Microservice applications are more “edge native”, in the sense that it is possible to move only some microservices to the edge, instead of the entire application [23]. Finally, the development effort is simplified, as each microservice may be coded in a different programming language (based on development, integration, and maintenance convenience), and may be packaged as a Linux container, thus forming a self-contained piece of software that can be developed also by a small team [24].

Life cycle management of a microservices-based application is usually managed by Kubernetes [25], a container orchestration [26] framework. The Kubernetes virtualization unit is the POD, which includes a single (or multiple) container running an instance of a single (or multiple) microservice. In a typical configuration, a POD runs an instance of a microservice. Kubernetes executes the PODs within the nodes (servers) of a cluster according to predefined scheduling policies [25]. Specifically, when the microservice load grows, the PODs are replicated, so that there are many instances of the same microservice serving the user requests, according to a load balancing strategy that randomly distributes the load to replicas. Actually, Kubernetes extensions, called *service mesh*, can implement more complex balancing strategies, which allow a more flexible request routing by automatically injecting transparent HTTP/gRPC proxies within the PODs. The proxies intercept any incoming and outgoing service requests to the PODs and apply user-defined routing/balancing strategies.

A. CONTRIBUTION

This paper explores the use of available communication and cloud technologies to implement a testbed where mobile applications offload complex tasks to edge or cloud computing data centers, while they move between cells served by multiple radio access technologies (multi-RAT).

The testbed is modular, and the adopted strategies can be modified to evaluate the effectiveness of new communication and cloud solutions as a part of research projects or focused studies. The testbed enables the integration of laboratory technologies that are not easily obtainable today through public infrastructure, such as edge data centers and mmWave radio access. In addition, the testbed enables the integration of public infrastructure, such as cloud data centers and 5G cellular networks, thus realizing a multi-RAT cloud continuum environment [27].

Although the proposed testbed can be applied to a wide range of mobile applications, as a design methodology, we focus on a multi-RAT cloud continuum system aimed at supporting navigation services of mobile ground robots. These robots need to offload AI-based computer vision tasks to edge and cloud servers to perform object detection, and they move in a radio environment consisting of mmWave Wi-Fi micro-cells at 60 GHz and 5G cells at 3.6 GHz.

The practical implementation problems highlighted by a use case with ground robots are addressed by proposing i) a general multi-RAT cloud/edge architecture that leverages

Kubernetes, Istio service mesh, and Open vSwitch (OvS) technologies, and ii) a design strategy for microservice applications meant for this environment. The developed software tools and applications, installation and operating instructions, configuration settings, and other data, are available in a public repository.[1]

In the following, we describe our system with a top-down approach, starting from the application to the cloud, and finally to the network. Since we used a testbed-driven design methodology, each section starts with the general concepts and, subsequently, dives into the technology used for their implementation.

First, we review some related works in Section II. Then, in Section III, we describe the considered edge computing scenario and the related testbed technologies. We continue in Section IV outlining the design strategy employed for our robot application, which can also be generalized to other types of mobile applications requiring task offloading. Then, Section V presents the multi-RAT cloud/edge testbed architecture used by our robot, and we describe the technologies used by our mobile application to offload complex AI tasks. Performance measurements are discussed in Section VI and, finally, in Section VII we draw conclusions and summarize the lessons learned.

II. RELATED WORKS

The offloading and distribution of complex tasks, either to the edge or cloud, has been explored by several researchers in the last few years. In [28], offloading is used to increase the performance of YOLO object detectors running on the cloud/edge rather than on smartphone hardware; the testbed is made of a dedicated edge server, and measurements have shown that the developed system improves multiple aspects, such as latency, detection accuracy, and battery endurance. In [29], GPU-equipped Kubernetes nodes are used to offload neural network services, such as object detection, with the purpose of defining an improved POD allocation policy. The results have shown increased transmission bandwidth, improved memory management, and more balanced POD allocation. The Edge-V framework proposed in [30] is designed to test high-bandwidth connectivity among vehicles at mmWave frequencies and task offloading for high-burden computational jobs. This framework demonstrated a 60% saving in terms of latency through a testbed made of a single edge node, which can possibly use services available in cloud virtual machines. The offloading solutions proposed in [28], [29], [30] have several interesting advantages, but these solutions were either not conceived for devices without autonomous mobility [28], [29], or not implemented with mobile robots [30].

The authors of [31] explored the trade-off between consumed energy and latency, in hybrid Wi-Fi/cellular mobile edge computing systems: their optimization algorithm offloads computation to minimize both consumed energy and processing latency. Additionally, service caching of frequently-requested computational resources (such as programs, libraries, and databases) has been added in the optimization trade-off [11], thus leading to increased benefits in energy consumption and processing latency. The joint use of Wi-Fi 6 and 5G in an industrial IoT environment is explored in [32], where task offloading is paired with optimal RAT selection, to optimize transmit power, execution delay, and overall costs: reinforcement learning together with game theory is used in a Lyapunov-based optimization framework. The concept of parallel multi-RAT (Wi-Fi, 5G, etc.) usage is studied in [33], where the authors propose an algorithm for optimal multi-RAT traffic offloading, which is capable to equalize the different radio technology communication delays, distribute the capacity in intermediate relay nodes, and avoid packet reordering delays.

For what concerns robotic aspects, the authors of [34] introduced a service platform for robots, to be implemented in the cloud. They designed a distributed computing architecture based on Apache Sparks, with several layers functioning as gateways, services, and algorithms: the results have shown a timing gain with respect to a local implementation, while preserving or improving detection accuracy and resource scaling. Distributed robotic systems often rely on middleware such as the *Robot Operating System* (ROS), whose design principles include distribution and modularity [35]. In [36] a robotic platform derived from ROS is presented, with an architecture based on microservices and implemented with Kubernetes, Docker, and Jenkins. The authors of [36] focused mainly on continuous integration, deployment and delivery, so that the entire process of robotic services provisioning is automated. The investigated application is the parking of autonomous vehicles, divided into numerous microservices dedicated to lane detection, parking detection, sign detection, and so on. Their results have shown that the service invocation time is considerably reduced when compared to that of other monolithic solutions. In [37], [38] the authors investigated a design methodology to containerize and orchestrate ROS-based applications, so that performance and memory footprint overheads are minimized, and integration and verification take place before deployment. Their findings have shown that the additional orchestration overhead may not be negligible if careful splitting of ROS nodes into Docker containers is not taken into account. The implementation of a complete robotic testbed for automated inspection has been presented in [39], where the cloud-edge continuum paradigm has also been evaluated with 5G radio links. Due to the strict latency requirements of robotic applications, the communication handover between different radio access links is an important aspect that has been investigated in [40], [41]. The authors of [40] have shown that orchestrated planning of container migration during the handover phase is crucial to minimize downtime and reduce latency. A WLAN with virtualized APs was used for the radio links in [41] to show that the contextual information

[1] https://github.com/gbaruffa/liquid-edge-microservices-inference-public

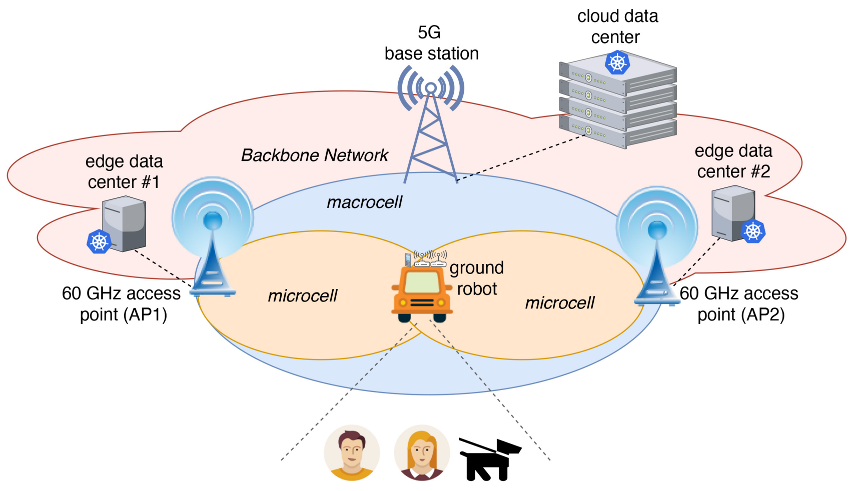

FIGURE 1. Reference scenario.

of the network is essential when migrating the virtual AP functionality.

Although ROS supports deployment solutions based on containerization and orchestration, most applications employ ROS nodes that rely on a stateful paradigm (e.g., mapping or pose estimation methods). Furthermore, ROS requires the presence of a master node that coordinates all the other nodes, and therefore the master node must be active. If it is reinstantiated when switching to a different edge, it could cause the failure of the entire ROS ecosystem.

Although we focus on a ground robot implementation, a similar application could also be developed with unmanned aerial vehicles (UAV). The offload of navigation control in UAV is studied in [42], where a model-predictive control is employed to predict the trajectory of the UAV. The implementation of this task in a local Kubernetes cluster has shown that the latency due to offloading represents a critical point, especially if the edge nodes are resource-bounded, or in case of insufficient link quality. Therefore, the reduction of latency is a critical point not only in ground robotic applications, like the one developed herein, but also, and even more crucial, in UAV contexts, where the real-time constraints are very tight.

In summary, compared to the literature [28], [29], [30], [38], [40], we focus more on cloud-specific aspects of the problem, that is, how to use cloud technologies to create a distributed cloud/edge service platform, where tasks are offloaded. Furthermore, our work provides guidelines to design a microservice application that takes advantage of such a platform. Finally, we remind that we have released all software and hardware project files as open-source (see Footnote 1), encouraging their reuse and extension.

III. REFERENCE SCENARIO

Figure 1 shows the considered scenario. The network infrastructure is made of mmWave Wi-Fi APs and a 5G public network. The cloud/edge infrastructure consists of edge data centers and a cloud data center. Each mmWave AP steers cloud-directed traffic to an edge data center associated with it.

A ground robot connects to the mmWave network through a pool of network interface cards (NICs) managed by a specific connection control strategy. To support connectivity outside the mmWave coverage, the robot uses a 5G NIC (e.g., a mobile phone) to connect with the 5G network.

The ground robot is equipped with a camera and runs a CV application that can benefit from some degree of computational offloading to cloud/edge resources. For example, the robot navigation task may require the detection of objects, people, or faces in the surrounding environment, which can be performed outside the robot. Therefore, video frames captured by the robot are sent to microservices, running in a cloud/edge infrastructure. The microservices perform the required operations and then send the results back to the robot.

We focus on a cloud computing scenario in which mobile applications use microservices run by a cloud provider, according to a function-as-a-service model. This cloud model requires microservice (function) instances to be activated and replicated on demand, according to specific placement and replication policies.

Regarding the placement policy, we consider that when a stream of service requests arrives at an edge data center, it is immediately served if there is a running instance of the microservice; otherwise, a new instance of the microservice is run. During such a *cold start* phase, initial requests are rerouted toward the cloud data center, where an instance is surely running.

Regarding the replication policy, we assume that the number of instances of the same microservice is automatically controlled, so that each one has a CPU consumption below a target value. When there is only a running instance, and it is not used for a configured inactivity timeout, it is eventually removed [43]. We assumed that an inactivity timeout is used for edge data centers to save resources, whereas, for the cloud data center, we always ensured the existence of at

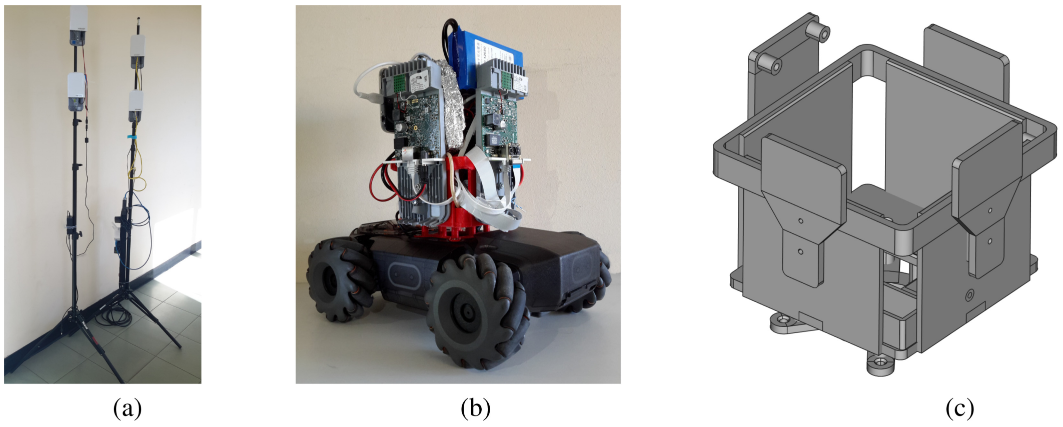

FIGURE 2. a) The two mmWave APs used for the testbed; b) the autonomous ground drone with the mounted cradle box; c) FreeCAD cradle box.

least a microservice instance (i.e., the inactivity timeout is set to infinity).

A. TESTBED ARCHITECTURE AND HARDWARE

We implemented the reference scenario in the testbed, which consists of two APs (Fig. 2a) that implement the IEEE 802.11ad standard [44] at 60 GHz and provide directional mmWave links to the robot. These APs are *MikroTik wAP 60Gx3* devices, which provide azimuth angular coverage of nearly 180 degrees. In addition, we use a public 5G network working at 3.6 GHz.

A cloud data center and two edge data centers are interconnected by a gigabit Ethernet infrastructure. Each edge data center serves a different mmWave AP. The network latency between a mmWave AP and its edge data center is negligible. To emulate the delay between the edge and cloud parts of the scenario, we introduced an artificial 50 ms round-trip time (RTT) delay to all cloud/edge communications. Data centers are made of a cluster of servers managed by a single Kubernetes instance.

The ground robot (Fig. 2b) is a *DJI RoboMaster S1*, which was chosen for its ease of programming and low cost [45]. It has four wheels and has been modified to include our open-source 3D-printed cradle box (Fig. 2c), in which we mounted four 60 GHz Wi-Fi NICs, 90 degrees apart from each other, and a Raspberry Pi 4 board to run our custom software. The Raspberry board is connected to the robot controller via a USB cable. The other USB ports on the Raspberry were used for a 2.4 GHz Wi-Fi dongle, a 5G mobile phone, and to connect a USB3 hub that hosts four USB-Gigabit Ethernet interfaces, which are connected to the 60 GHz Wi-Fi NICs. These NICs are *MikroTik wAP 60G* devices with an azimuth angular coverage of 60 degrees, and beamforming is achieved by the integrated 36-antenna array. Finally, an additional battery is included to power all the devices, while a fail-safe 2.4 GHz IEEE 802.11n link is maintained between the on-board computer and a control computer for debugging and general control purposes. This link is used by the testbed supervisor to start the robot client applications and to access the mmWave stations carried by the robot.

IV. APPLICATION DESIGN

The design of modern mobile applications is usually based on a client-side part of the application that runs on the mobile device, and a server-side part made up of microservices, whose instances are executed by Linux containers in a cloud data center. We noted that there is currently no established technology for live migration of containers between different data centers.[2] Therefore, we assumed a cloud continuum scenario in which microservice instances do not follow the mobile device, but the client must leverage microservice instances that are found (or instantiated) in the data center associated with the radio cell serving the mobile device.

We propose an application design strategy to satisfy this assumption, where only stateless tasks can be offloaded, while stateful tasks must remain in the mobile device. In this way, if the device needs to change data center due to its mobility, the application basic performance is not altered, because instances of the microservices running in the new data center can operate immediately without requesting the state of the application.

A. ROBOT APPLICATION

We applied this design strategy to an application that controls the navigation of the ground robot in the surrounding environment, while managing its network connectivity. Actually, this is a microservice application, whose microservices can implement one or more tasks. The goal of the robot

[2]For instance, the most widely used container runtime, *runC*, uses a technology named CRIU [46] that requires a shared file system between the source and destination hosts, which is feasible in the case of hosts belonging to the same data center. However, this method is unsafe in the case of hosts belonging to different data centers, because for each I/O disk operation there is the risk of transferring data from one data center to another, across a geographical network, consequently introducing delays that dramatically reduce the benefits of local computation given by edge computing. Other commercial solutions, such as Virtuozzo PaaS, support live migration, usually only between regions within the same data center [47].

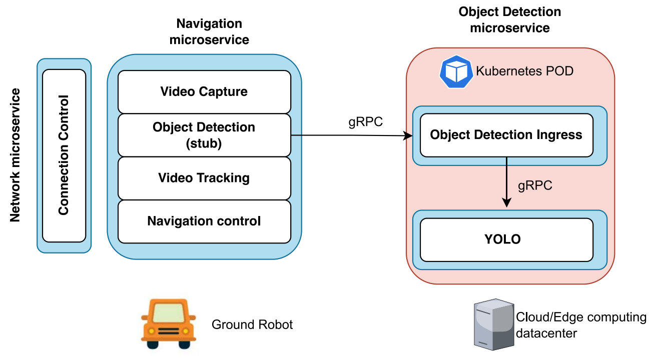

FIGURE 3. Application microservices.

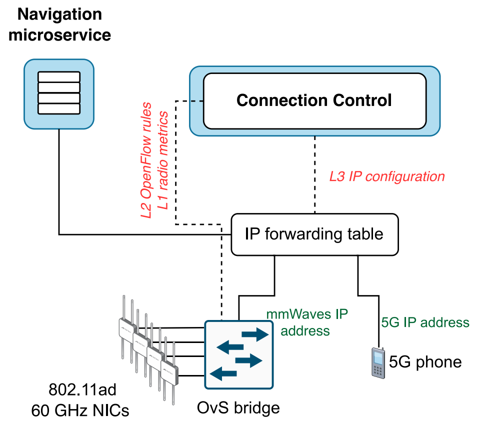

FIGURE 4. Network microservice.

application is to visually track an object of a chosen class (e.g., a person) and follow this object.

We used a microservice application design for both the client-side and server-side parts of the application, as shown in Fig. 3. Specifically, the client-side part is made of two microservices running in the robot, which are used to control the robot movements (*Navigation* microservice), as well as its connection with the fixed network (*Network* microservice). In addition, there is also an object detection task of the *Navigation* microservice, which is offloaded to a server-side *Object Detection* microservice, whose instances can run in cloud or edge data centers.

B. NAVIGATION CONTROL

The *Navigation* microservice performs a sequence of tasks in a continuous loop. A video capture task receives a frame from the robot camera with 640×480 resolution and compresses it using JPEG. Then, the JPEG image is transferred with gRPC to a remote, stateless *Object Detection* task implemented by a couple of Linux containers running in a single Kubernetes POD. Specifically, an ingress container (*Object Detection Ingress*) receives the gRPC request and, in turn, calls a YOLO container [48] that uses OpenCV [49], and specifically its Darknet engine [50].[3] The output of the Darknet engine is a list of bounding boxes corresponding to the detected objects, and each box has a label related to the type of object detected (person, car, etc.). The *Object Detection Ingress* enriches this list with some service metrics, and sends it back to the robot. The received list of labeled bounding boxes is the input to a *Video Tracking* task that implements the tracking of an object, whose class is chosen by the user, in the current video frame. This process is responsible for pairing the positions of the same object, in two consecutive frames, so that only a specific object is tracked. The tracking information is then used by a *Navigation Control* task to steer the robot toward the tracked object.

Note that the stateless nature of the offloaded *Object Detection* task allows the navigation control to use different instances of the *Object Detection* microservice without any operational problems. Instance switching occurs when the application changes the data center that serves it, as the new data center has another instance of the *Object Detection* microservice. In addition, instance switching can also take place when the cloud/edge orchestrator (i.e., Kubernetes) decides to increase the number of replicas of the microservice.

[3]If CUDA GPU acceleration is available and enabled, NVIDIA cuDNN [51] is leveraged to accelerate the detection.

C. NETWORK CONTROL

The *Network* microservice implements a connection control function that periodically checks the link status on each available radio interface, and then selects the best link. The architecture used for connection control is shown in Fig. 4. We used the OvS Linux tool to create an OpenFlow software bridge, to which we added the four IEEE 802.11ad NICs of the robot. The 5G phone interface is not bridged.

The robot has two IP addresses, one for the 5G NIC and another one for the OvS bridge, both received through DHCP mechanisms. Therefore, we are considering a scenario where each RAT is requested to provide the robot with a different IP address.[4] In addition, each mmWave NIC has a local private IP address, to be accessed by the connection control function that retrieves radio metrics, such as the receive signal strength indicator (RSSI).

In order to exchange traffic with the fixed network, the robot uses a single interface at a time. The connection control function configures the 5G phone as the default gateway when the strongest mmWave signal is below a configurable threshold (e.g., -80 dBm); otherwise, the bridge is used as the default gateway route. In this latter case, OpenFlow rules are used to control the traffic exchange between the robot and the bridged mmWave NICs. Specifically, the connection control function injects OpenFlow rules into the OvS bridge to forward (i) outgoing address resolution protocol (ARP) and IP packets pointing to local IP addresses of NICs to

[4]This condition can be enforced both in a self-operated test scenario, i.e., where the RATs are all independently managed, and in a hybrid scenario, where the mmWave RATs are managed by private entities and the 5G ones by a public operator.

the relevant NIC ports, (ii) the remaining outgoing traffic to a single mmWave NIC chosen according to a specific radio policy, and (iii) incoming traffic to the robot TCP/IP stack, without any re-routing. Note that the use of OpenFlow Layer 2 switching control allowed us to keep all mmWave NICs always active, and possibly connected with the same AP, so that the connection control function can continuously access their radio metrics, while avoiding broadcast traffic loops, such as ARP requests. In fact, connection loops can occur when two or more NICs are connected to the same AP.

Concerning the radio selection policy for choosing the best mmWave connection, we have implemented two different logics:

- *best-metric*, with a policy that selects the radio that maximizes a specific metric (e.g., RSSI or signal-to-noise ratio), which can be changed by the user;
- *round-robin*, with a policy that sequentially cycles among all available interfaces, independently of their link status.

The second policy is mostly used for debugging purposes, and hence we mainly focus on the first policy. Although several metrics can be devised, we have chosen a metric $M_{i,k}$ that combines two radio parameters provided by the ith Wi-Fi NIC at time k: the RSSI $R_{i,k}$ (power in dBm rounded to the closest integer) and link quality $Q_{i,k}$ (integer between 0 and 100). Specifically, we employ the product of the two parameters, biasing the RSSI to make it nonnegative, as expressed by

$$M_{i,k} = \left(R_{i,k} + 120\right)Q_{i,k}. \tag{1}$$

Intuitively, we expect that $R_{i,k} + 120$ and $Q_{i,k}$ are positively correlated, therefore the chosen metric in (1) has a quadratic behavior in $R_{i,k}$ or $Q_{i,k}$, while taking both parameters into account. It is worth noting that the RSSI and quality parameters may change between consecutive measurements mainly due to the user mobility in the surrounding environment: to avoid unnecessary switching, we include into the selection policy a hysteresis mechanism, where switching happens only when the best current metric of a device is significantly larger than the metric of the device in use. Therefore, at time k, the device with the best metric is calculated as

$$i_k^+ = \arg\max_i\{M_{i,k}\}, \tag{2}$$

and the value of its metric $M_{i_k^+,k}$ is compared with the metric $M_{i_{k-1}^*,k}$ of the device that has been previously chosen: if the difference between the two metrics exceeds a given threshold Δ_M (set to 400 in our tests), the current best is updated from i_{k-1}^* to $i_k^* = i_k^+$. When $i_{k+1}^* \neq i_k^*$, the switch takes place. This is summarized by the equation

$$i_k^* = \begin{cases} i_{k-1}^*, & M_{i_k^+,k} - M_{i_{k-1}^*,k} \le \Delta_\mathrm{M}, \\ i_k^+, & M_{i_k^+,k} - M_{i_{k-1}^*,k} > \Delta_\mathrm{M}. \end{cases} \tag{3}$$

When a Wi-Fi NIC is not connected to any AP, then it is excluded from the search process in (1)–(3). If there are no connected Wi-Fi NICs, then the 5G NIC supplies the fallback connection to the remote service.

Although our choice for (1)–(3) is not the only possibility, the results in Section VI show that the chosen metric, combined with the hysteresis mechanism, works adequately well for our goals.

V. CLOUD/EDGE ARCHITECTURE AND SERVICES

As shown in Fig. 1, the cloud infrastructure consists of edge data centers and a cloud data center. Edge and cloud data centers are connected by a high-speed backbone network. The data center resources are controlled by Kubernetes, and to simplify orchestration, we assumed a single Kubernetes control plane running in the cloud data center.

We have assumed that each microservice (e.g., the *Object Detection* microservice) has at least one instance running in the cloud data center. However, other instances of the same microservice can run in edge data centers according to a specific placement and replication policy. For instance, in Fig. 1, the *Object Detection* microservice has an instance in any data center.

A. REQUEST ROUTING

Each instance of the *Object Detection* microservice is actually a Kubernetes POD. To enable access to PODs, which are replicas of the same microservice, and to perform load balancing among them, Kubernetes offers a resource called *Service*, which exposes all these PODs through a single endpoint, that is, an URL and an IP address internal to the cluster. The HTTP requests targeting a Service are routed to a randomly chosen POD of the replica set.

This random load balancing is not suitable for edge computing, because there is a risk that a request entering an edge data center will be randomly forwarded to a replica POD running in another data center, thus undermining the latency benefits of edge computing. In general, we argue that Kubernetes Services should not be used for load balancing purposes when the PODs matched to a Service are distributed across multiple data centers, and back-and-forth exchanges between data centers must be avoided due to latency constraints.

Consequently, the problem we face is the following: how can we ensure that service requests (from mobile devices to an edge data center) are served by local microservice instances, if available, rather than randomly forwarded to another data center by Kubernetes?

To solve this “request routing” issue, we moved the load balancing control to a kind of Kubernetes “overlay” layer, namely the Istio service mesh [52]. With a service mesh, the routing of requests to PODs associated with a Service is no longer controlled by the Kubernetes random load balancer, but by higher-level strategies implemented by the service mesh software. Specifically, Istio injects into each POD a *sidecar* transparent proxy, which intercepts any incoming or outgoing gRPC/HTTP request for Kubernetes Services and

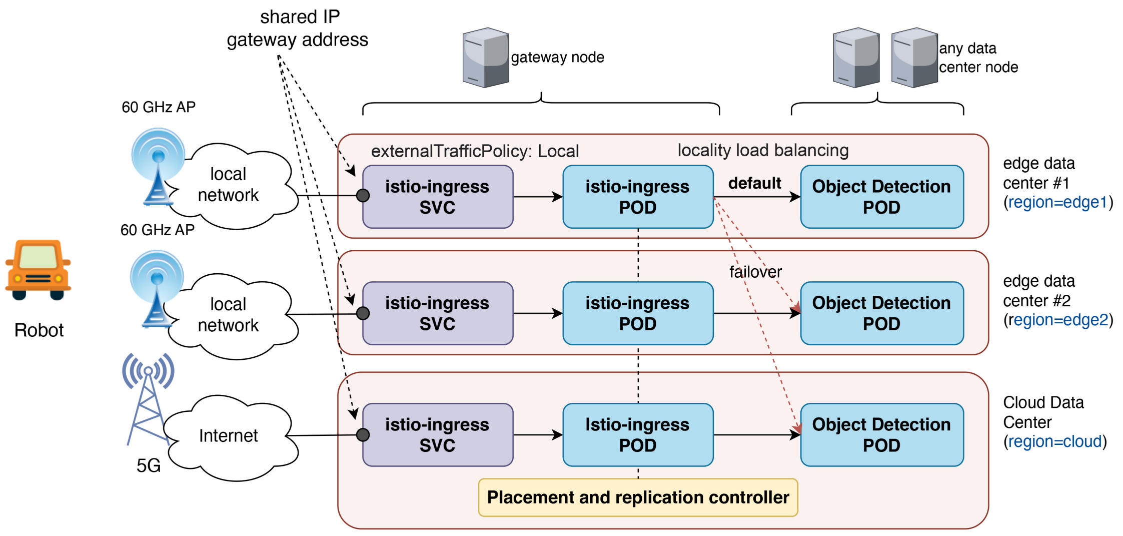

FIGURE 5. Request routing via Istio service mesh.

routes them to specific PODs chosen by Istio load balancing rules rather than by the random Kubernetes one.[5] Thus, we describe the configuration we deployed for the Istio service mesh and the network in the following.[6] Without lack of generality, we focus on a robot application that uses a single back-end microservice. However, the proposed solution can be reused for any type of application that possibly employs multiple microservices.

As shown in Fig. 5, we labeled each node in a data center with a topology identifier, e.g., `region = edge1` for nodes in edge data center #1. Each data center has a *gateway* node that runs an Istio Ingress POD, exposed as a Kubernetes Service (SVC) with *NodePort*. Note that the associated Kubernetes External Traffic Policy rule requires external connections served by the local Istio Ingress POD, instead of being randomly bounced to an Istio Ingress POD running in other data centers. As a result, an Istio Ingress POD running in a gateway node can be contacted using an IP address of the gateway node on a specific default port (e.g., 30844).

The robot application calls an URL resolved, by the DNS, to the public IP address of the gateway node of the cloud data center where Istio Ingress runs. When the robot uses a 60 GHz link, the access point steers traffic to the gateway node of its edge data center, where Istio Ingress runs. The edge data center gateway has two IP addresses, specifically an IP address that uniquely identifies it in the mmWave RAT network, and another one that reuses the IP address of the cloud data center gateway resolved by the DNS.[7] In this way, when the robot changes the service data center, the gRPC persistent connection to the Object Detection microservice is closed, because the POD changed, but it is immediately reopened with the new POD without needing DNS resolution, since the targeted IP address is still valid.[8]

An Istio Ingress implements a reverse proxy that behaves as an API gateway. We configured it to forward *Object Detection* gRPC requests to *Object Detection* PODs according to a sequence of load balancing policies, specifically, Locality and Least Request, which are offered by the Istio service mesh.

The Locality load balancing policy provides that when an Istio Ingress receives a request from the robot, this request is routed to an *Object Detection* POD, which runs in nodes with the same labels of Istio Ingress, i.e., to PODs of the same data center, thus achieving the local processing goals of edge computing. If a local POD implementing the *Object Detection* microservice is either unavailable in the data center or in an error state, Istio uses a fallback approach that forwards the request to the cloud data center. Therefore, this solution is also resilient to possible failures and to temporary unavailabilities of the local POD, for instance, when it is starting up.

In the presence of POD replication, among the pool of PODs selected by the Locality load balancing policy, Istio uses (by default) the Least Request load balancing policy. With this policy, every request is directed to the POD of the pool with the lowest number of outstanding requests,

[5]Testbed results on resource consumption of a sidecar proxy are reported in [53].

[6]The configuration involves three types of Istio resources, named Istio Gateway, Istio Virtual Service and Istio Destination Rule, whose YAML files can be found in our GitHub repository (https://github.com/gbaruffa/liquid-edge-microservices-inference-public). We do not report the details of these files and only provide a conceptual explanation of the resulting configuration.

[7]In a production environment, this "shared" IP address (or cloud data center subnet) could be announced locally via BGP from the edge data center to the mobile network operator access point. Alternatively, traffic steering can be manually configured at the access point, as in our case.

[8]Note that if the robot used the RESTful HTTP API to contact the remote microservice, connection restoration would probably not take place unless a handover occurs during the processing of an object-detection call. Therefore, the change in the service center would occur seamlessly.

with ties resolved at random. This ensures that the most burdened PODs will not receive additional requests. This policy remarkably reduces the request latency, especially in heavy-traffic regimes [54].

B. PLACEMENT AND REPLICATION POLICY

A placement policy is a logic that decides "where" to run microservice instances to achieve a specific goal, such as reducing latency or network traffic. For microservice applications, a key aspect is to take into account the relationships between the microservices. In this paper, we do not delve into the design of placement strategies and we have implemented a *placement and replication controller* with a simple policy that, nevertheless, fits well with our robot application. Specifically, when a request for the *Object Detection* microservice arrives at an edge data center, if no local instance of the microservice exists, a new instance is started with a dedicated Horizontal POD Autoscaler (HPA), to control the number of replicas so that the CPU load of any instance is less than 70%. To activate the new instance, the placement controller runs a new Kubernetes deployment with Node Affinity configured to match the topology identifier of the edge data center nodes. In addition, the new Deployment has a unique label used by the new HPA to select it exclusively. To detect requests arriving at a specific data center, the placement control continuously checks the access logs provided by Istio Ingress. In addition, if the access logs report no requests, after an inactivity timeout (3 minutes) the new edge deployment and related HPA are removed. The latter policy does not apply to the cloud data center, where at least one instance always exists.

C. JOURNEY OF REQUESTS

It is worth summarizing the journey of robot *Object Detection* requests, as detailed in the following:

- At the start of the *Navigation* microservice, the robot establishes a gRPC session with the DNS name of the cloud gateway node. This URL is resolved with an IP address, which is routed to the cloud data center by public Internet in case of 5G communications, or to edge data centers in case of mmWave communications.
- Once the gRPC session is established at the gateway node of a data center, it is handled by the local Istio Ingress POD, and related requests are routed (proxied) to an *Object Detection* POD operating in the same data center, if it exists; otherwise, the requests are temporarily rerouted to the cloud data center while a new local *Object Detection* POD is started.
- After the starting period (also known as *cold-start*), the local *Object Detection* POD will be used by robot requests, thus reducing latency and network traffic.
- When the robot moves in a cell served by another data center, the gRPC session is interrupted, but it is immediately reestablished with the Istio Ingress running in the new data center, without any new DNS resolution.

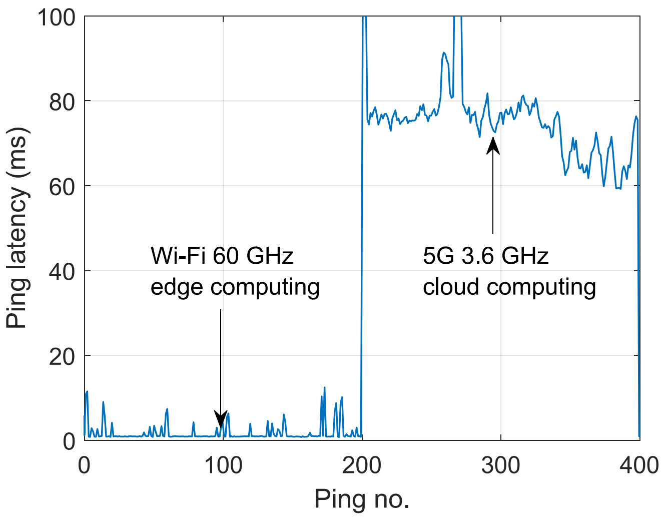

FIGURE 6. RTT between the robot and the data center varying the access link.

D. TESTBED POTENTIAL

The testbed architecture can be reused by any other microservice application, enabling comprehensive exploration of edge computing impact on diverse application-level choices, including the topology of dependency graphs, the use of internal cache, and so forth [23]. Moreover, it facilitates in-depth investigations into various placement and replication strategies, a pivotal aspect often addressed in academic literature through simulations and theoretical analyses [55], [56], [57], [58]. Additionally, it opens possibilities for refining solutions in 5G multi-RAT environments, allowing seamless channel selection optimization [59], [60]. Finally, it supports the comparison of different mmWave RAT technologies, offering valuable insights for their application in edge computing [61].

VI. PERFORMANCE EVALUATION

We carried out two performance evaluation campaigns using the testbed platform and software framework described in the previous sections. First, we analyzed network-level performance and, then, application-level performance. Recall that the testbed consists of two 60 GHz microcell APs (AP1 and AP2) and a third 5G macrocell. The robot has 5 NICs. NICs 1, 2, 3, and 4 are Wi-Fi at 60 GHz, each connecting the robot to either AP1 or AP2, which route the robot traffic to a local edge data center (Fig. 1). NIC 5 is 5G, connecting the robot to a 5G public network, which routes the robot traffic to a remote cloud data center.

A. NETWORK PERFORMANCE

We measured the round-trip time (RTT) and throughput between the robot and the gateway of the serving data center, by varying the radio access technology: we used an IP Ping sequence to measure the RTT, and the Iperf3 tool [62] to measure the throughput.

Fig. 6 shows the measured RTT between the robot and the serving data center, when the robot uses AP1 at 60 GHz

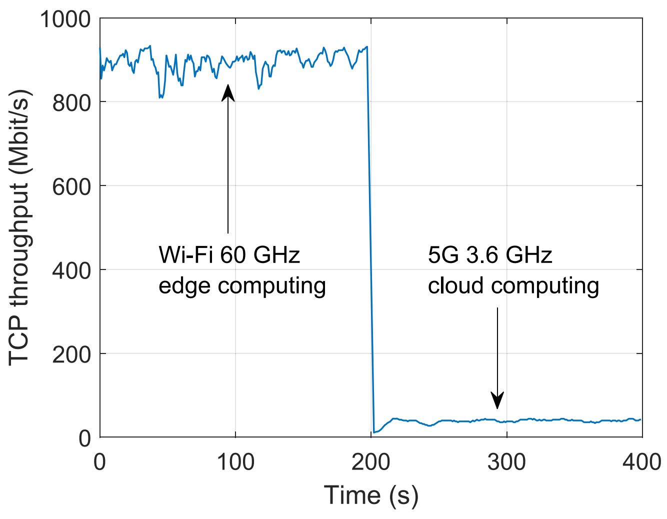

FIGURE 7. Throughput between the robot and the data center varying the access link.

and when the robot uses the 5G network. We note that the RTT is very limited for the 60 GHz AP as the robot uses the edge data center, amounting to a delay (average and standard deviation) of 1.6±2.0 ms. When the robot uses the 5G network, the delay increases to 75.5±18.0 ms, as the robot uses the remote cloud data center and public 5G/Internet.

Similarly, Fig. 7 shows the throughput between the robot and the serving data center as the access links vary. We note that, for 60 GHz connections, the throughput is 895±48 Mbit/s, whereas the throughput drops to 38±8 Mbit/s for the 5G connection, due to the low 5G reference signal received power strength in our laboratory, i.e., about −102 dBm.

The results confirm the successful emulation of a mobile network scenario, where robots can move from cells providing high-speed/low-latency edge computing services to cells with low-speed/high-latency cloud computing services.

B. APPLICATION PERFORMANCE

First, we analyzed the performance of the application under static conditions, that is, when the ground robot does not move in the environment (Section VI-C). Next, we evaluated the performance under dynamic conditions, when the ground robot actively follows a moving target (Section VI-D). We considered the *detection latency* as a performance indicator, defined as the difference between the time instant when the *Navigation* microservice sends a request to the *Object Detection* microservice, and the time it receives the response back. The detection latency is mainly the sum of four delays:

- *transmission delay:* the amount of time the *Navigation* microservice requires for the transmission of a frame to the *Object Detection* microservice;
- *propagation delay:* the network propagation delay between the *Navigation* and *Object Detection* microservices;
- *processing delay:* the processing delay due to the execution of the YOLO algorithm by the *Object Detection* microservice on the received frame;

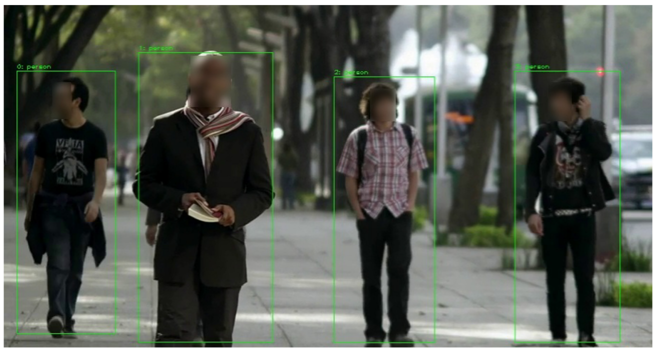

FIGURE 8. Outputs of the *Object Detection* microservice: bounding boxes enclose detected persons.

- *cold-start delay:* temporary delay due to redirecting the robot requests to the cloud data center, while waiting for the activation of a local instance of the *Object Detection* microservice.

C. RESULTS IN STATIC CONDITIONS

For static measurements, we considered a *samplevideo* sequence.[9] in spite of a static image that could be acquired by the robot, and object detection (without tracking) is performed to obtain the bounding boxes of the objects (Fig. 8).

We considered two scenarios. First, a *local* scenario, where the *Navigation* microservice runs in the same machine that also hosts the *Object Detection* microservice. In this case, the best performance is achieved, i.e., minimum latency, since the involved microservices are co-located. Second, a *robot* scenario, where the *Navigation* microservice is executed by the robot and the robot uses different access configurations, namely: i) the AP1 at 60 GHz served by an edge data center, to reproduce a case of edge computing with mmWave link access, ii) the AP1 at 60 GHz served by the cloud data center, to reproduce a case of cloud computing with mmWave link access and, finally, iii) the 5G network served by the cloud data center, to reproduce a case of cloud computing with 5G access. Fig. 9 shows the obtained results. The latency of the local scenario is 22.5±1.9 ms and can be considered an irreducible processing latency. For the non-local cases, where the *Navigation* microservice runs in the robot, the best performance is achieved with 60 GHz and edge computing, which achieves a latency of 51.0±3.9 ms. Indeed, the high bandwidth provided by the mmWave link enables fast transmission of video frames between the robot and the edge data center, which is also close to the serving AP, thus limiting the propagation delay of the network. In the case of 60 GHz and cloud computing, the latency increases to 217.9±78.7 ms, mainly because of the higher RTT between the access point and the data center, where object detection

[9]The *samplevideo* sequence was composed by concatenating nine short sequences from the “4K” section of [63], scaling the frame size to 910×480 pixels and changing the frame rate to 30 fps, for a total of 2528 frames: the clip shows groups of people and vehicles.

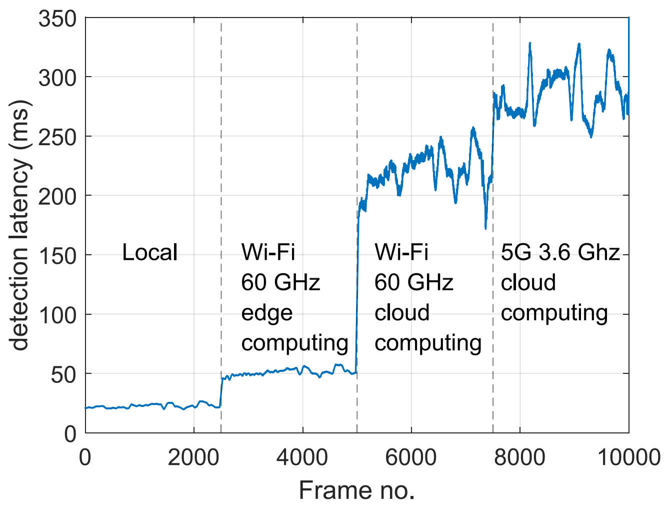

FIGURE 9. Detection latency varying the access link.

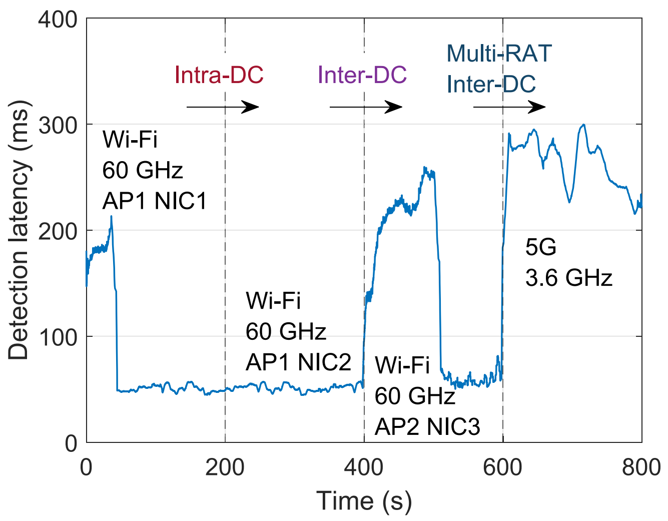

FIGURE 10. Detection latency during forced handovers, with cold-start on edge data centers and without local container image.

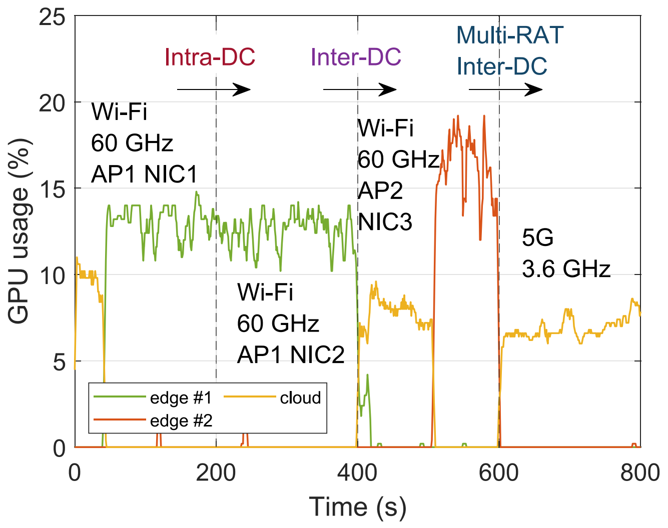

FIGURE 11. GPU usage of different data centers during forced handovers, with cold-start on edge data centers and without local container image.

takes place. This result confirms that, in our scenario, it is necessary to use edge computing to have an actual application-level benefit from mmWave access; otherwise, core network latency and throughput can undermine the benefit of high access speed. Finally, in the case of 5G access, the latency increases further to 284.3±70.8 ms, due to increased network latency and reduced network throughput (see Figs. 6–7).

In the next tests, we simulate the occurrence of a handover between the robot and the network every 200 s. To simulate a handover, we either change the OpenFlow rule configuration of the OvS bridge of the robot to switch between 60 GHz APs, or the default gateway to switch from 60 GHz APs to the 5G network. At the beginning of the test, no instance of the *Object Detection* microservice was running in the edge data center. Consequently, when an edge data center starts receiving requests from the robot, our placement strategy reactively executes a new local instance of the *Object Detection* microservice. The corresponding container image may, or may not, be available on the Kubernetes node scheduled to run the instance, and we measured performance in both cases.

Figs. 10–11 show the detection latency and the GPU utilization of tests when the container image is not available in the edge Kubernetes node (the GPU is used to accelerate YOLO object detection processing). The average delay due to the download of the container images from the public software repository and to the startup process is 90.7±0.7 s, resulting from the average of multiple runs. The robot starts its handover journey from AP1 at 60 GHz using NIC1. The AP1 is served by the edge data center #1, as shown in Fig. 1. The placement policy detects the presence of robot requests for the *Object Detection* microservice on the edge data center #1, and runs an instance of the microservice there, fetching the image from the public software registry. During this cold-start period, requests are rerouted by Istio Locality load balancing to the *Object Detection* instance that runs in the cloud data center. Indeed, the detection latency is rather high (183.4±73.4 ms) due to rerouting, and the GPU on the cloud data center performs the object detection job.[10] After 43 s, the *Object Detection* container image is completely downloaded from the software registry, the container is booted, and the cold-start phase ends. From this point on, the robot requests are directed to that local instance of the *Object Detection* microservice, as confirmed by the reduction of the detection latency (which drops to 50.4±12.4 ms), and the switching of GPU usage from the cloud to edge #1. Note that there is no service interruption during service instance switching from the cloud to the edge, due to the stateless nature of the *Object Detection* microservice. At 200 s, the robot connection is switched to

[10]GPUs have different hardware, so their use for the same job could be different.

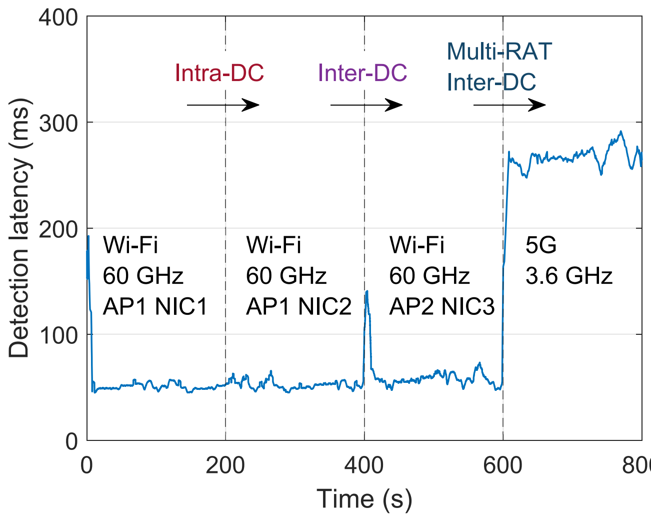

FIGURE 12. Detection latency during forced handovers, with cold-start on edge data centers and with local container image.

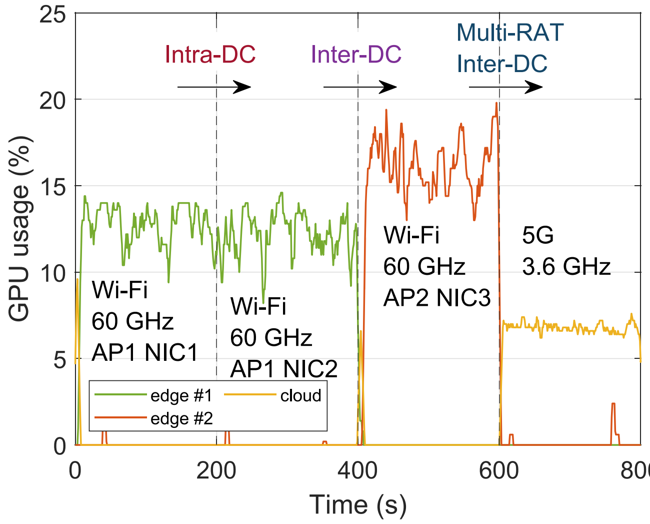

FIGURE 13. GPU usage of different data centers during forced handovers, with cold-start on edge data centers and with local container image.

NIC2 without changing the AP; thus, the robot performs an intra-data center (intra-DC) handover, and the observed performance remains stable (latency is 51.2±4.4 ms). At 400 s, the robot switches the connection to AP2 at 60 GHz using NIC3. The AP is served by edge data center #2, so the robot undergoes an inter-data center (inter-DC) handover. Again, we note a cold-start phase that, however, in this case, lasted much longer (for 108 s, with a latency of 206±93 ms) because of temporary Internet congestion, which slowed down the download of container images from the registry. After the cold-start, the latency decreases to 59.1±28.1 ms. Finally, at 600 s, the robot connects to the 5G network, thus performing a multi-RAT and inter-DC handover. The latency increases to 263.0±36.8 ms, due to the increase in transmission and propagation delay, as already shown in Fig. 9.

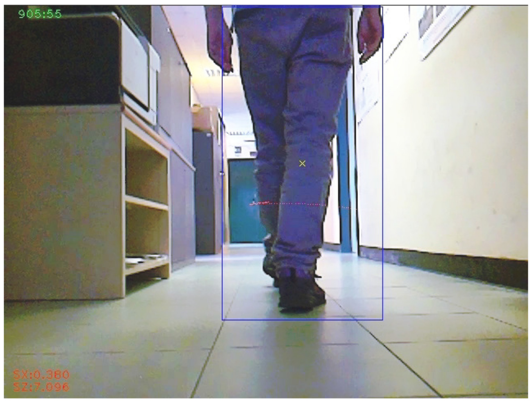

FIGURE 14. Bounding box and centroid marker when tracking a person.

As shown in Figs. 12–13, a similar performance is obtained when an *Object Detection* container image is present on the edge nodes, but with a much shorter cold-start phase, because the registry image fetching is no longer needed. In this case, the nominal startup time of the containers, averaged over multiple runs, is 2.9±0.2 s. However, in practice, there are additional delays due to the other software components of the chain. Specifically, in Fig. 12, there is an initial delay of 6 s, no delay for the intra-DC handover case happening at 200 s, and a delay of 9 s for the inter-DC handover taking place at 400 s. In summary, with respect to the cold-start case, handover delays are reduced from tens of seconds to few seconds. Therefore, the image preloading on edge data center can be useful to mitigate the temporary increase of latency due to the cold-start phase.

D. RESULTS IN DYNAMIC CONDITIONS

In this section, we present a dynamic test with the robot in motion, while tracking a target in a video captured by its camera. The *Navigation* microservice is configured to follow a person walking through our laboratory rooms (Fig. 14). In the test, the tracked person walks at a constant speed along a predetermined round-trip path, completed in about 220 s. During the test, the robot computes its trajectory and position through odometry estimation. This estimation is used to visualize the robot position during its movement and to highlight which radio links are active along the path, as well as their performance. The *Network Control* microservice automatically selects the radio connection according to the policy shown in Section IV-C.

Fig. 15 shows the path of the robot as it tracks the moving person. Different colors identify the NIC used to contact the *Object Detection* microservice. NIC5 is the 5G phone, and other NICs are for Wi-Fi at 60 GHz. Different markers identify the Wi-Fi AP and 5G BS that are used. This figure also shows the walls of the laboratory rooms and the locations of the different APs.

The test begins with the robot and the person to be tracked that are positioned near AP1. Subsequently, the person, and then the following robot, move back and forth toward AP2.

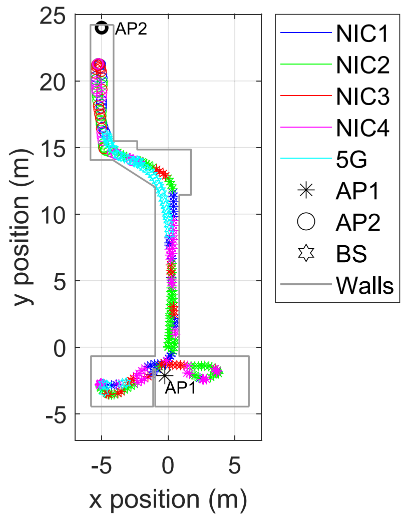

FIGURE 15. Trajectory computed using odometry estimation with used NIC and AP/BS.

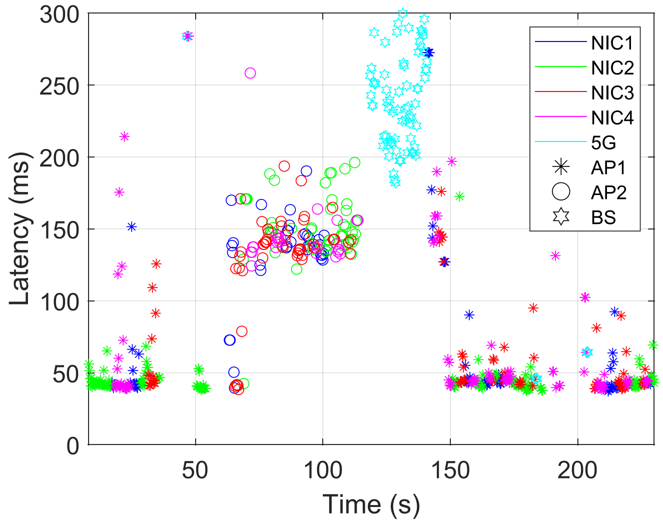

FIGURE 16. Detection delay during robot trajectory with used NIC and AP/BS.

Note that during movement, the robot changes APs and NICs. At the bottom of the map, the robot uses the microcell beamed by AP1, while in the middle, the 60 GHz signal from AP1 drops, and consequently the robot switches to the macrocell covered by a 5G base station through NIC5. Then, the robot connects to AP2 at 60 GHz as it approaches the associated microcell. On the path back to AP1, a similar connection behavior occurs.

Fig. 16 shows the detection delay associated with different positions of the robot over time; the different markers and colors are used to indicate which NIC and AP/BS the robot has been using. As expected, we observe that when the robot is using 60 GHz and edge computing (i.e., AP1 or AP2), the detection delay decreases significantly (latency of 131.9±153.0 ms) with respect to using 5G and cloud computing (latency of 337.9±194.8 ms), which also means the system is capable of processing more frames per second and, thus, performing better tracking jobs. The latency when the robot is connected to AP2 is higher than when connected to AP1, since the GPU of edge data center #2 is slower. When 5G is used, the latency grows and has a greater variability, due to the high variable network delay and throughput that we experienced during the test.

Overall, the results show the effectiveness of our testbed in automatically and efficiently switching between RATs, and properly utilizing the nearest cloud/edge resources to offload mobile robot functions, according to a cloud/edge computing framework [28], [29], [30], [38], [40].

VII. CONCLUSION AND LESSON LEARNED

In this paper, we propose a multi-RAT cloud continuum testbed designed for offloading complex tasks of mobile applications. Specifically, the testbed design has been driven by a cloud-assisted navigation application, used by a mobile robot, which offloads CV tasks to microservices running in cloud/edge data centers reached by mmWave Wi-Fi and 5G access links.

This testbed can be easily replicated in laboratory environments to evaluate the effectiveness of novel cloud and/or communication solutions, specifically developed for multi-RAT cloud continuum environments. Our main findings and recommendations emerged from the testbed development experience are the following.

- *Microservice Design with Stateless Task* - Considering the current absence of an established technology for live migration of Linux containers among different data centers, a microservice-based application design grants a better exploitation of edge computing, because it allows to split the application stateful and stateless activities into different microservices, offloading the stateless ones to cloud/edge resources. In this way, if during the movement of the user device (e.g., a robot), the service data center changes, the functionality of the application is not altered, because instances of the microservices running in the new data center can operate immediately without knowing the state of the application, as they perform only stateless tasks. However, to ensure service continuity, it is necessary to ensure that a mobile device finds the requested microservices already running in the accessed edge data center or, at least, that requests are redirected to a data center where a running instance exists.
- *Shared IP Address for Data Center API Gateway* - To avoid a long service interruption (due to possible invalidation of DNS resolution) when the serving data center changes, it is useful to assign the same shared IP address to all the data center nodes, implementing an API gateway. This IP address should be used by mobile applications to contact cloud/edge microservices. In this way, when the serving data center changes, a new DNS resolution is not required, making data center handovers much faster.

- *Request Routing through Service Mesh* - A Kubernetes cluster may host multiple instances of the same microservice, i.e., replication occurs. Actually, for load balancing purposes, Kubernetes operates on a TCP/IP connection basis. Each incoming connection for a microservice is routed to microservice replicas, chosen at random. Therefore, there is a risk that a connection, originating from a mobile device to a microservice, is not served in its local data center but by an instance running in another data center, undermining the latency reduction we seek with edge computing. Besides, a balancing choice is maintained for the whole duration of the connection, thus making this mechanism ineffective in the case of persistent connections, such as in the case of gRPC. To solve this problem, we can move request routing (or load balancing) to a higher layer, by installing Istio service mesh services in the cluster, with a setup that prefers local microservice instances, and uses remote instances only for fallback purposes. Moreover, Istio performs load balancing on a (HTTP/gRPC) request basis, which is also effective in case of persistent connections.

ACKNOWLEDGMENT

The authors thank Tommaso Tamarindi and Alberto Dionigi for their help in setting up the software tools.

REFERENCES

[1] M. J. Shehab, I. Kassem, A. A. Kutty, M. Kucukvar, N. Onat, and T. Khattab, "5G networks towards smart and sustainable cities: A review of recent developments, applications and future perspectives," *IEEE Access*, vol. 10, pp. 2987–3006, 2022.

[2] P. Wang, B. Di, and L. Song, "Cellular communications over unlicensed mmWave bands with hybrid beamforming," *IEEE Trans. Wireless Commun.*, vol. 21, no. 8, pp. 6064–6078, Aug. 2022.

[3] X. Xu, Q. Chen, H. Jiang, and J. Huang, "Millimeter-wave NR-U and WiGig coexistence: Joint user grouping, beam coordination, and power control," *IEEE Trans. Wireless Commun.*, vol. 21, no. 4, pp. 2352–2367, Apr. 2022.

[4] Y. Kakkad, D. K. Patel, S. Kavaiya, S. Sun, and M. López-Benítez, "Optimal 3GPP fairness parameters in 5G NR unlicensed (NR-U) and WiFi coexistence," *IEEE Trans. Veh. Technol.*, vol. 72, no. 4, pp. 5373–5377, Apr. 2023.

[5] Y. Gao, Z. Zhang, H. Hu, X. Wang, Y. Jin, and X. Chu, "Fair and efficiency coexistence between NR-U and WiGig networks enabled by a matching-based framework with forbidden pairs," *IEEE Trans. Veh. Technol.*, vol. 72, no. 9, pp. 11814–11827, Sep. 2023.

[6] L. Liu, M. Zhao, M. Yu, M. A. Jan, D. Lan, and A. Taherkordi, "Mobility-aware multi-hop task offloading for autonomous driving in vehicular edge computing and networks," *IEEE Trans. Intell. Transp. Syst.*, vol. 24, no. 2, pp. 2169–2182, Feb. 2023.

[7] Z. Ning et al., "Dynamic computation offloading and server deployment for UAV-enabled multi-access edge computing," *IEEE Trans. Mobile Comput.*, vol. 22, no. 5, pp. 2628–2644, May 2023.

[8] X. Dai et al., "Task co-offloading for D2D-assisted mobile edge computing in industrial Internet of Things," *IEEE Trans. Ind. Informat.*, vol. 19, no. 1, pp. 480–490, Jan. 2023.

[9] H. Jiang, X. Dai, Z. Xiao, and A. Iyengar, "Joint task offloading and resource allocation for energy-constrained mobile edge computing," *IEEE Trans. Mobile Comput.*, vol. 22, no. 7, pp. 4000–4015, Jul. 2023.

[10] P. A. Apostolopoulos, G. Fragkos, E. E. Tsiropoulou, and S. Papavassiliou, "Data offloading in UAV-assisted multi-access edge computing systems under resource uncertainty," *IEEE Trans. Mobile Comput.*, vol. 22, no. 1, pp. 175–190, Jan. 2023.

[11] W. Fan et al., "Joint task offloading and service caching for multi-access edge computing in WiFi-cellular heterogeneous networks," *IEEE Trans. Wireless Commun.*, vol. 21, no. 11, pp. 9653–9667, Nov. 2022.

[12] G. Damigos, T. Lindgren, and G. Nikolakopoulos, "Toward 5G edge computing for enabling autonomous aerial vehicles," *IEEE Access*, vol. 11, pp. 3926–3941, 2023.

[13] (ETSI Stand. Co., Sophia Antipolis, France). *Multi-Access Edge Computing (MEC)*. Accessed: Apr. 10, 2024. [Online]. Available: https://www.etsi.org/technologies/multi-access-edge-computing

[14] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, 2016, pp. 779–788.

[15] J. Terven, D.-M. Córdova-Esparza, and J.-A. Romero-González, "A comprehensive review of YOLO architectures in computer vision: From YOLOv1 to YOLOv8 and YOLO-NAS," *Mach. Lear. Knowl. Extr.*, vol. 5, no. 4, pp. 1680–1716, 2023.

[16] D. H. D. Reis, D. Welfer, M. A. D. S. L. Cuadros, and D. F. T. Gamarra, "Mobile robot navigation using an object recognition software with RGBD images and the YOLO algorithm," *Appl. Artif. Intell.*, vol. 33, no. 14, pp. 1290–1305, 2019.

[17] T. Shimoda, S. Koga, and K. Sato, "Autonomous motion control of a mobile robot using marker recognition via deep learning in GPS-denied environments," *J. Robot. Mechatron.*, vol. 35, no. 1, pp. 136–144, 2023.

[18] S. Singh, A. Suri, J. Singh, M. Singh, Nikita, and D. K. Yadav, "Object identification and tracking using YOLO model: A CNN-based approach," in *Proc. Int. Conf. Mach. Learn. Inf. Process. (ICMLIP)*, 2021, pp. 153–160.

[19] J. Zhou, L. Feng, R. Chellali, and H. Zhu, "Detecting and tracking objects in HRI: YOLO networks for the NAO 'I see you' function," in *Proc. 27th IEEE Int. Symp. Robot Human Interact. Commun. (RO-MAN)*, 2018, pp. 479–482.

[20] X. Larrucea, I. Santamaria, R. Colomo-Palacios, and C. Ebert, "Microservices," *IEEE Softw.*, vol. 35, no. 3, pp. 96–100, May/Jun. 2018.

[21] L. Kamiński, M. Kozłowski, D. Sporysz, K. Wolska, P. Zaniewski, and R. Roszczyk, "Comparative review of selected Internet communication protocols," *Found. Comput. Decis. Sci.*, vol. 48, no. 1, pp. 39–56, Mar. 2023.

[22] A. Singleton, "The economics of microservices," *IEEE Cloud Comput.*, vol. 3, no. 5, pp. 16–20, Sep./Oct. 2016.

[23] A. Detti, "Microservices from cloud to edge: An analytical discussion on risks, opportunities and enablers," *IEEE Access*, vol. 11, pp. 49924–49942, 2023.

[24] A. Sill, "The design and architecture of microservices," *IEEE Cloud Comput.*, vol. 3, no. 5, pp. 76–80, Sep./Oct. 2016.

[25] F. Lumpp, F. Fummi, H. D. Patel, and N. Bombieri, "Enabling Kubernetes orchestration of mixed-criticality software for autonomous mobile robots," *IEEE Trans. Robot.*, vol. 40, pp. 540–553, 2024.

[26] Y. Gong, H. Yao, J. Wang, D. Wu, N. Zhang, and F. R. Yu, "Decentralized edge intelligence-driven network resource orchestration mechanism," *IEEE Netw.*, vol. 37, no. 2, pp. 270–276, Mar./Apr. 2023.

[27] S. Moreschini, F. Pecorelli, X. Li, S. Naz, D. Hästbacka, and D. Taibi, "Cloud continuum: The definition," *IEEE Access*, vol. 10, pp. 131876–131886, 2022.

[28] X. Ran, H. Chen, X. Zhu, Z. Liu, and J. Chen, "DeepDecision: A mobile deep learning framework for edge video analytics," in *Proc. IEEE Conf. Comput. Commun. (INFOCOM)*, 2018, pp. 1421–1429.

[29] J. Kim, S. Ullah, and D.-H. Kim, "GPU-based embedded edge server configuration and offloading for a neural network service," *J. Supercomput.*, vol. 77, pp. 8593–8621, Aug. 2021.

[30] F. Raviglione, C. Casetti, and F. Restuccia, "Edge-V: Enabling vehicular edge intelligence in unlicensed spectrum bands," in *Proc. IEEE 97th Veh. Technol. Conf. (VTC)*, 2023, pp. 1–5.

[31] W. Fan, J. Han, L. Yao, F. Wu, and Y. Liu, "Latency-energy optimization for joint WiFi and cellular offloading in mobile edge computing networks," *Comput. Netw.*, vol. 181, Nov. 2020, Art. no. 107570.

[32] F. Zhou, L. Feng, M. Kadoch, P. Yu, W. Li, and Z. Wang, "Multiagent RL aided task offloading and resource management in Wi-Fi 6 and 5G coexisting industrial wireless environment," *IEEE Trans. Ind. Informat.*, vol. 18, no. 5, pp. 2923–2933, May 2022.

[33] A. Ali and K. Ali, "Multiradio parallel offloading in multiaccess edge computing: Optimizing load shares, scheduling, and capacity," *IEEE Internet Things J.*, vol. 11, no. 3, pp. 4047–4062, Feb. 2024.

[34] J. Liu, F. Zhou, L. Yin, and Y. Wang, "A novel cloud platform for service robots," *IEEE Access*, vol. 7, pp. 182951–182961, 2019.

[35] J. Zhang, F. Keramat, X. Yu, D. M. Hernández, J. P. Queralta, and T. Westerlund, "Distributed robotic systems in the edge-cloud continuum with ROS 2: A review on novel architectures and technology readiness," in *Proc. 7th Int. Conf. Fog Mobile Edge Comput. (FMEC)*, 2022, pp. 1–8.

[36] Z. Yin, J. Liu, B. Chen, and C. Chen, "A delivery robot cloud platform based on microservice," *J. Robot.*, vol. 2021, Feb. 2021, Art. no. 6656912.

[37] F. Lumpp, M. Panato, F. Fummi, and N. Bombieri, "A container-based design methodology for robotic applications on Kubernetes edge-cloud architectures," in *Proc. Forum Specif. Des. Lang. (FDL)*, 2021, pp. 1–8.

[38] F. Lumpp, M. Panato, N. Bombieri, and F. Fummi, "A design flow based on Docker and Kubernetes for ROS-based robotic software applications," *ACM Trans. Embed. Comput. Syst.*, vol. 2023, pp. 1–25, Apr. 2023.

[39] A. Motz et al., "Mobile robots enabled by intelligent edge–reference model and testbed setup," in *Proc. 54th Int. Symp. Robot. (ISR Eur.)*, 2022, pp. 1–8.

[40] M. V. Ngo, T. Luo, H. T. Hoang, and T. Q. Ouek, "Coordinated container migration and base station handover in mobile edge computing," in *Proc. IEEE Glob. Commun. Conf. (GLOBECOM)*, 2020, pp. 1–6.

[41] M. Groshev, G. Baldoni, L. Cominardi, A. de la Oliva, and R. Gazda, "Edge robotics: Are we ready? An experimental evaluation of current vision and future directions," *Digit. Commun. Netw.*, vol. 9, no. 1, pp. 166–174, Feb. 2023.

[42] A. S. Seisa, S. G. Satpute, B. Lindqvist, and G. Nikolakopoulos, "An edge-based architecture for offloading model predictive control for UAVs," *Robotics*, vol. 11, no. 4, p. 80, Aug. 2022.

[43] "Fission—A framework for serverless functions on Kubernetes." fission, Accessed: Apr. 11, 2024. [Online]. Available: https://fission.io

[44] *Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications Amendment 3: Enhancements for Very High Throughput in the 60 GHz Band*, IEEE Standard 802.11ad-2012, Dec. 2012.

[45] S. Evripidou, K. Georgiou, L. Doitsidis, A. A. Amanatiadis, Z. Zinonos, and S. A. Chatzichristofis, "Educational robotics: Platforms, competitions and expected learning outcomes," *IEEE Access*, vol. 8, pp. 219534–219562, 2020.

[46] "CRIU—Live migration." criu, Accessed: Apr. 11, 2024. [Online]. Available: https://criu.org/Live_migration

[47] (Virtuozzo Softw. Co., Schaffhausen, Switzerland). *Virtuozzo—Environment Migration Between Regions*. Accessed: Apr. 11, 2024. [Online]. Available: https://www.virtuozzo.com/application-platform-docs/environment-regions-migration/

[48] J. Redmon and A. Farhadi, "YOLOv3: An incremental improvement," 2018, *arXiv:1804.02767*.

[49] J. Howse and J. Minichino, *Learning OpenCV 4 Computer Vision with Python 3: Get to Grips With Tools, Techniques, and Algorithms for Computer Vision and Machine Learning*. Birmingham, U.K.: Packt Publ. Ltd, 2020.

[50] J. Redmon. "Darknet: Open source neural networks in C." Accessed: Jun. 1, 2023. [Online]. Available: http://pjreddie.com/darknet/

[51] S. Chetlur et al., "cuDNN: Efficient primitives for deep learning," 2014, *arXiv:1410.0759*.

[52] O. Sheikh, S. Dikaleh, D. Mistry, D. Pape, and C. Felix, "Modernize digital applications with microservices management using the Istio service mesh," in *Proc. 28th Annu. Int. Conf. Comput. Sci. Softw. Eng. (CASCON)*, 2018, pp. 359–360.

[53] "Istio—Performance and scalability." istio, Accessed: Apr. 11, 2024. [Online]. Available: https://istio.io/latest/docs/ops/deployment/performance-and-scalability

[54] X. Zhou, F. Wu, J. Tan, Y. Sun, and N. Shroff, "Designing low-complexity heavy-traffic delay-optimal load balancing schemes: Theory to algorithms," *Proc. ACM Meas. Anal. Comput. Syst.*, vol. 1, no. 2, pp. 1–30, 2017.

[55] C. T. Joseph and K. Chandrasekaran, "IntMA: Dynamic interaction-aware resource allocation for containerized microservices in cloud environments," *J. Syst. Archit.*, vol. 111, Dec. 2020, Art. no. 101785.

[56] S. Wang, M. Zafer, and K. K. Leung, "Online placement of multi-component applications in edge computing environments," *IEEE Access*, vol. 5, pp. 2514–2533, 2017.

[57] F. Rossi, V. Cardellini, F. Lo Presti, and M. Nardelli, "Geo-distributed efficient deployment of containers with Kubernetes," *Comput. Commun.*, vol. 159, pp. 161–174, Jun. 2020.

[58] W. Lv et al., "Microservice deployment in edge computing based on Deep Q learning," *IEEE Trans. Parallel Distrib. Syst.*, vol. 33, no. 11, pp. 2968–2978, Nov. 2022.

[59] R. Maldonado, C. Rosa, and K. I. Pedersen, "Multi-link techniques for new radio-unlicensed URLLC in hostile environments," in *Proc. IEEE 93rd Veh. Technol. Conf. (VTC)*, 2021, pp. 1–6.

[60] A. Roy, P. Chaporkar, A. Karandikar, and P. Jha, "Online radio access technology selection algorithms in a 5G multi-RAT network," *IEEE Trans. Mobile Comput.*, vol. 22, no. 2, pp. 1110–1128, Feb. 2023.

[61] Y. J. Guo, M. Ansari, R. W. Ziolkowski, and N. J. Fonseca, "Quasi-optical multi-beam antenna technologies for B5G and 6G mmWave and THz networks: A review," *IEEE Open J. Antennas Propag.*, vol. 2, pp. 807–830, 2021.

[62] J. Dugan et al., (ESnet/Lawrence Berkeley Nat. Lab., Berkeley, CA, USA).*iPerf—The Ultimate Speed Test Tool for TCP, UDP and SCTP*. Accessed: Apr. 11, 2024. [Online]. Available: https://iperf.fr/

[63] (Xiph.Org Found. Softw. Co., Somerville, MA, USA). *Xiph.Org Video Test Media*. Accessed: Apr. 11, 2024. [Online]. Available: https://media.xiph.org/video/derf/

GIUSEPPE BARUFFA was born in Perugia, Italy, in 1970. He received the Laurea degree in electronic engineering and the Ph.D. degree in telecommunications from the University of Perugia, Perugia, in 1996 and 2001, respectively. In 2005, he visited the Swiss Federal Polytechnic of Lausanne, Lausanne, Switzerland. Since 2005, he has been an Assistant Professor with the Department of Engineering, University of Perugia. His main research interests include digital television broadcasting, joint source/channel video coding, and LPWAN sensor networks.

ANDREA DETTI is currently a Professor of Wireless Networks and Cloud Computing with the Department of Electronic Engineering, University of Rome "Tor Vergata." He is the coauthor of many papers in journals and conference proceedings and has participated in several EU-funded projects with coordination and research roles. His current research interests include 5G networks, cloud/edge computing, and AI applied to these sectors. He is currently an Associate Editor of IEEE TRANSACTIONS ON NETWORK AND SERVICE MANAGEMENT.

LUCA RUGINI (Member, IEEE) was born in Perugia, Italy, in 1975. He received the Laurea degree (cum laude) in electronic engineering and the Ph.D. degree in telecommunications from the University of Perugia, Perugia, Italy, in 2000 and 2003, respectively. In 2007, he visited the Delft University of Technology, Delft, The Netherlands. He is currently an Assistant Professor with the Department of Engineering, University of Perugia. His research interests lie in the area of signal processing for communications. He has been serving as an Area Editor for *Digital Signal Processing* since 2023. He served as an Associate Editor for the IEEE SIGNAL PROCESSING LETTERS from 2014 to 2018, for *Digital Signal Processing* from 2012 to 2022, and for the *EURASIP Journal on Advances in Signal Processing* from 2015 to 2022.

FRANCESCO CROCETTI received the Ph.D. degree in information engineering from the University of Perugia in 2022. He was an Exchange Visitor with the Agile Robotics and Perception Lab, New York University, USA. He is currently a Postdoctoral Researcher with the Intelligent Systems, Automation, and Robotics Laboratory and a Lecturer of Industrial Robotics with the Department of Engineering, University of Perugia. His research interests are in field of automation, fault diagnosis, robotics, and machine learning.

PAOLO BANELLI (Member, IEEE) received the Laurea degree (cum laude) in electronics engineering and the Ph.D. degree in telecommunications from the University of Perugia, Perugia, Italy, in 1993 and 1998, respectively. Since 2019, he has been a Full Professor with the Department of Engineering, University of Perugia, where he has been an Associate Professor since 2005, and an Assistant Professor since 1998. He was a Visiting Researcher with the University of Minnesota, Minneapolis, MN, USA, in 2001, and a Visiting Professor with Stony Brook University, Stony Brook, NY, USA, from 2019 to 2020. His research interests include signal processing and optimization for wireless communications, graph signal processing and learning, and signal processing for biomedical applications. He served as an Associate Editor for the IEEE TRANSACTIONS ON SIGNAL PROCESSING from 2013 to 2016 and *EURASIP Journal on Advances in Signal Processing* from 2013 to 2019. He has been serving as an Associate Editor for the IEEE OPEN JOURNAL OF SIGNAL PROCESSING since 2020. In 2009, he was the General Co-Chair of the IEEE International Symposium on Signal Processing Advances for Wireless Communications. He was a member of the IEEE Signal Processing for Communications and Networking Technical Committee from 2011 to 2013.

GABRIELE COSTANTE (Member, IEEE) received the Ph.D. degree in robotics from the Department of Engineering, University of Perugia in 2016, where he is currently an Associate Professor with the Intelligent Systems, Automation and Robotics Laboratory and a Lecturer of the courses computer vision and robot perception, and machine learning and data analysis. His main research interests include artificial intelligence, robotics, computer vision, and machine learning.

PAOLO VALIGI (Member, IEEE) received the Ph.D. degree from the University of Rome “Tor Vergata” in 1991. Since 2004, he has been a Full Professor with the Department of Engineering, University of Perugia. He is currently the Head of the Intelligent Systems, Automation and Robotics Laboratory Research Group. His research interests include robotics, nonlinear control, and systems biology.

Open Access funding provided by ’Università degli Studi di Perugia’ within the CRUI CARE Agreement