Groshev, M., Martín, J., Antevski, K., de la Oliva, A. & Bernardos, C. J. (25 June 2021). *COTORRA: Context-Aware testbed for robotic applications* [proceedings]. MobileServerless'21: Proceedings of the 1st Workshop on Serverless mobile networking for 6G Communications, 2021, Virtual WI USA.

DOI: 10.1145/3469263.3469857

COTORRA: COntext-aware Testbed fOR Robotic Applications

Milan Groshev
Universidad Carlos III de Madrid
Spain
mgroshev@pa.uc3m.es

Jorge Martín-Pérez
Universidad Carlos III de Madrid
Spain
jmartinp@it.uc3m.es

Kiril Antevski
Universidad Carlos III de Madrid
Spain
kantevsk@pa.uc3m.es

Antonio de la Oliva
Universidad Carlos III de Madrid
Spain
aoliva@it.uc3m.es

Carlos J. Bernardos
Universidad Carlos III de Madrid
Spain
cjbc@it.uc3m.es

ABSTRACT

Edge computing have received considerable attention as a promising candidate for the evolution of robotic systems. In this work, we propose COTORRA, an Edge driven robotic testbed that combines context information with robot sensor data to validate innovative concepts for robotic systems prior to being applied in a production environment. We have tested COTORRA in a controlled university environment as an easy applicable, serverless, and modular testbed on top of commodity network infrastructure. COTORRA supports pluggable robotic applications. To verify its feasibility and assess its performance, we ran a set of experiments that show how autonomous navigation applications can achieve target latencies bellow 15 ms, and perform an inter-domain Distributed Ledger Technology (DLT) federation within 19 seconds.

CCS CONCEPTS

• **Networks** → **Programmable networks**; **In-network processing**.

KEYWORDS

serverless, Edge, robotic applications, testbed, orchestration, DLT, multi-domain

ACM Reference Format:
Milan Groshev, Jorge Martín-Pérez, Kiril Antevski, Antonio de la Oliva, and Carlos J. Bernardos. 2021. COTORRA: COntext-aware Testbed fOR Robotic Applications. In *1st Workshop on Serverless mobile networking for 6G Communications (MobileServerless'21), June 25, 2021, Virtual, WI, USA.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3469263.3469857

1 INTRODUCTION

Networked robotics combines robotics with communication networks to enhance capabilities of robots. A clear example is Cloud robotics [7], which aims to integrate Cloud computing resources in robotics systems to increase re-configurability as well as to decrease the complexity and cost of robots. Thanks to Edge computing [13], real-time applications as autonomous navigation will run closer to the robots, and the robot-to-cloud communication will be enhanced thanks to improved latency and reliability. Since the robot-to-cloud communication will traverse fewer network links, higher bandwidth rates, and bounded jitter are achievable. Moreover, real-time contextual information is expected to be available in the Edge, allowing dynamic adaptation of application's logic to the actual status of the communication (e.g., radio channel) [2] or other aspects of the environment. Therefore, executing robotics applications in the Edge of the network is considered as a potential evolution of robotic systems.

Most of the existing work on testing platforms for robotic applications focus on Cloud robotics testbeds or platforms [1, 3, 4, 9–11]. In [11], the authors indicate the feasibility and challenges of a real world Cloud robotics implementation over Cloud testbed infrastructure. Besides, [4] suggests a Cloud robotics testbed that can be effective for developing not only robot-related experiments, but also network applications. By providing a Cloud robotics testbed for mobile robots, VC-bots [1] tries to address some important factors such as mobility, traffic scenario, protocol, cloud resources and localization awareness. Furthermore, due to the set of constrains (e.g., cost, time, safety) when performing physical testing of Unamanned Areal Vehicles (UAVs), [10] develops an Cloud-enabled, open-source simulation testbed for UAVs, thus reducing the entry barrier of UAV development and research. Finally, [3] and [9] propose Cloud robotics platforms that address number of implementation-related issues.

Although Edge computing appears as a straightforward extension for offloading time-sensitive robotic applications, such scenario can not be simply supported by migration of the existing Cloud model that adapts virtualization and containerization technologies [15]. The distributed and resource constrained nature of the Edge infrastructure imposes limitations regarding its capability of hosting diverse robotic applications and services since an overloaded Edge server significantly degrades the robotic performance and negates the aforementioned edge advantages. Recently, Serverless architectures, also known as Function-as-a-Service (FaaS), appeared as an alternative that allows developers to deploy applications where the management of the execution environment is delegated to the infrastructure provider. These Serverless applications can be used to run different robotic functionalities (e.g., motion planning, localization), invoked by different types of events (e.g., external user requests), such that the service provider decides

MobileServerless'21, June 25, 2021, Virtual, WI, USA

ACM ISBN 978-1-4503-8603-6/21/06…$15.00
https://doi.org/10.1145/3469263.3469857

on run-time for the resource allocation and execution. As a consequence, provider-managed containers will execute the robotic functionalities, without taking into account the Edge resource utilization or dealing with scalability and load-balancing burden. This can boost the utility of the Edge servers for robotic services, allowing the deployment of more functionality given their limited capabilities and resources, while meeting applications low latency requirements.

However, it is relevant to note that all the existing robotic testbeds or platforms depend on the use of Cloud computing. Unfortunately, no context-aware real-time robotic testbed is explicitly mentioned in the existing literature, despite the need for an analysis on implementation issues, standards and validation for Edge robotic applications. There is a lack of practical experience on context-aware real-time environments in robotics systems. That leads to limited applicability in production systems and final products.

In this work we propose COTORRA, a context-aware testbed architecture that operates at the edge of the network. This testbed offers deployment of serverless Plug-ins. Additionally, to support the testing of time-sensitive applications, COTORRA offers network emulation capabilities. To validate COTORRA, we implemented two applications on top of it: an orchestration and a federation solution for robotic systems. The experiments and results illustrate how COTORRA can be used to enhance robotic applications.

The remainder of this paper is organized as follows: we propose the architecture of our real-time context-aware testbed for robotic applications in Section 2, and its implementation in Section 3. Finally, we experimentally validate the testbed in Section 4 and conclude the paper in Section 5.

2 COTORRA ARCHITECTURE

The COTORRA architecture (see Figure 1) is designed to work with serverless Plug-ins that interact with the hardware infrastructure, and retrieve its context information using the COTORRA Core layer.

It is worth mentioning that the architecture is designed to be compliant with Multi-access Edge Computing (MEC) [12]. In particular, it is possible to implement the Plug-ins and Core layer building blocks as a set of autonomous applications and context-aware services running on top of a standard MEC virtualization infrastructure. This implies that COTORRA contributes to the experimental evaluation of MEC compliant applications.

2.1 COTORRA Core layer

COTORRA Core layer is a set of building blocks, namely, the Robot Middleware, Network Control, and Measurements building block. Every building block has an API to interact with the testbed, and report data to the Plug-ins layer. The following details the functionality of each building block:

- The Robot Middleware building block (*i*) collects, (*ii*) stores and (*iii*) exposes real-time $\{r_i\}_i \subset V(G)$ robots' data; and (*iv*) sends instructions to the robots, e.g., robots' movements. The Robot Middleware periodically collects the robots' Radio Unit (RU) attachment $\phi(r_i, R_i) \in \{0, 1\}$, and a vector of sensor data $\sigma(r_i) \in \mathbb{R}^n$ holding values such as the robot position $\sigma_p(r_i),\ 2 \leq p \leq d$, with d being the space dimension.[1] Using the aforementioned information, the Robot Middleware creates a contextual embedding $\kappa(r_i)$

$$\kappa\colon \{r_i\}_i \to \mathbb{R}^n \qquad (1)$$
$$r_i \mapsto \big(\phi(r_i, R_1), \ldots, \phi(r_i, R_N), \sigma(r_i)\big)$$

that is stored (via the Measurements API) in the Measurements building block, and exposed (via its own API) to the Plug-in layer (see Figure 1). Latter, a Plug-in uses the contextual embedding to elaborate real-time navigation $\sigma_p(r_i)$ (e.g., robot movement) and RU attachment instructions $\phi(r_i, R_i)$, and forwards them to the Robot Middleware API. Finally, Robot Middleware sends the Plug-in instructions to the robots.
- The Network Control building block (*i*) collects, (*ii*) exposes, and (*iii*) stores network and RU context information. One of its main functionalities is to (*iv*) emulate network conditions. As well, it (*v*) maps Virtual Functions (VF) and Virtual Links (VL) in the hardware graph, (*vi*) stores, and (*vii*) exposes the VF/VL mappings. The Network Control collects a vector with RUs' context information $\iota(R_i) \in \mathbb{R}^m,\ R_i \in V(G)$, so as the links' throughput $\lambda(n_1, n_2),\ (n_1, n_2) \in E(G)$, and delay $d(n_1, n_2),\ (n_1, n_2) \in E(G)$ between every robot, RU, switch, and server in the hardware graph G. In addition, the Network Control exposes the real-time RU context information, links' delay, and throughput via its API; and stores the prior data in the Measurements building block using the Measurements API. To emulate network conditions, Network Control (*i*) retrieves $\lambda(\cdot)$ throughput, and $d(\cdot)$ delay embeddings from the Measurements building block; and (*ii*) introduces artificial queuing delay $\psi(n_1, n_2) \geq 0,\ (n_1, n_2) \in E(G)$ so as packet drop rate $\delta(n_1, n_2) \geq 0$. As a result, the links $(n_1, n_2) \in E(G)$ in the hardware graph offer a delay of $\psi(n_1, n_2)d(n_1, n_2)$, and a throughput of $\frac{1}{\delta(n_1,n_2)}\lambda(n_1, n_2)$. The Network Control

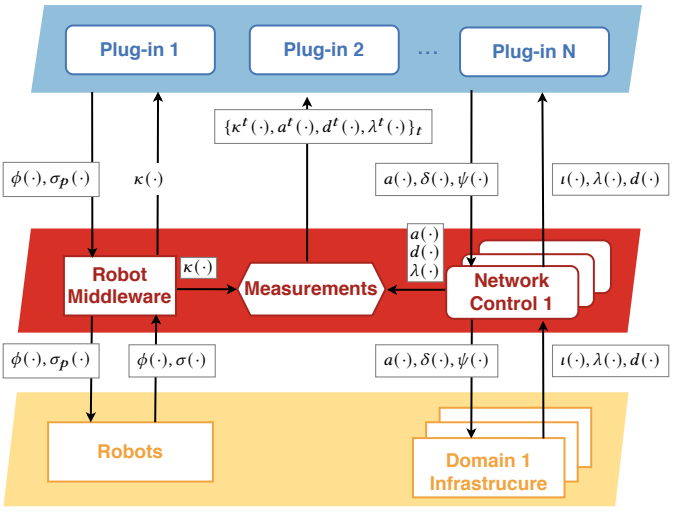


Figure 1: COTORRA exchange of information and architecture: (bottom) Hardware layer; (middle) Core layer building blocks; and (top) Plug-ins layer.

[1] Wheeled robots move on a $d = 2$ space, and UAVs on a $d = 3$ space.

Table 1: Notation table

Symbol	Definition
r_i	robot number $i \in \mathbb{N}$
G	hardware graph
$V(G)$	hardware graph nodes, e.g., a server
$E(G)$	hardware graph edges, e.g., a wireless link
R_i	RU number $i \in \mathbb{N}$
s_i	server number $i \in \mathbb{N}$
w_i	switch number $i \in \mathbb{N}$
$\phi(r_i, R_i)$	whether robot r_i is attached to RU R_i
$\sigma(r_i)$	vector with robot r_i sensor data
$\sigma_p(r_i)$	robot r_i position
$\kappa(r_i)$	robot r_i RU attachment and sensor data
$d(n_1, n_2)$	delay between nodes $n_1, n_2 \in V(G)$
$\iota(R_i)$	context information of RU R_i
$\lambda(n_1, n_2)$	throughput in the link (n_1, n_2)
$\psi(n_1, n_2)$	artificial queuing delay in link (n_1, n_2)
$\delta(n_1, n_2)$	artifitial packet drop rate in link (n_1, n_2)
$a(n)$	set of VFs hosted at node n
$a(n_1, n_2)$	set of VLs hosted at link (n_1, n_2)

API exposes operations to request both, queueing delay, and packet drop.

Additionally, the Network Control building block acts as a Virtual Infrastructure Manager (VIM) [14] that (*i*) maps VFs $\{v_i\}_i$ to hardware nodes $a(n) = \{v_1, \ldots, v_n\}, n \in V(G)$; and (*ii*) maps VLs in the hardware links, that is $a(n_1, n_2) = \{(v_1, v_2), \ldots, (v_{n-1}, v_n)\}$. Both VFs, and VLs mappings are periodically (*h*) stored and (*i*) exposed by the Measurements component (using the Measurements API); and Plug-ins can issue new mappings over the exposed Network Control API.

- The Measurements building block (*i*) stores, and (*ii*) exposes a history of every embedding. The embeddings history is represented as a set

$$\left\{\{\kappa^t(r_i)\}_{r_i}, \{a^t(n)\}_n, \{d^t(n_1, n_2)\}_{n_i}, \{\lambda^t(n_1, n_2)\}_{n_i}\right\}_t \quad (2)$$

with $\kappa^t(r_i)$ denoting the contextual embedding of robot r_i at time t. As shown in (2), the embeddings history stores the VF/VL mappings, network delay, and throughput information. The embeddings history is exposed via the Measurements API to the Plug-in layer (see Figure 1).

2.2 COTORRA Plug-ins layer

COTORRA offers the testing capabilities through the Plug-ins layer. A robotic application in COTORRA is composed of one or more serverless Plug-ins. Each serverless Plug-in is a robotic functionality (e.g., orchestration, navigation), and in some cases it might ask for affinity constraints to be deployed on a specific server/device. For example, since a serverless remote-control Plug-in is latency sensitive, it will ask to be deployed near the edge. Serverless Plug-ins are executed either as containers or virtual machines that implement the APIs to interact with the COTORRA core layer. The Plug-ins receive periodic contextual testbed information (e.g., sensor data $\sigma(\cdot)$, signal strength $\iota(\cdot)$) that the Core layer produces via its APIs. This information can be used by the Plug-ins to develop their logic and define various types of invoking events if the Plug-in is a serverless

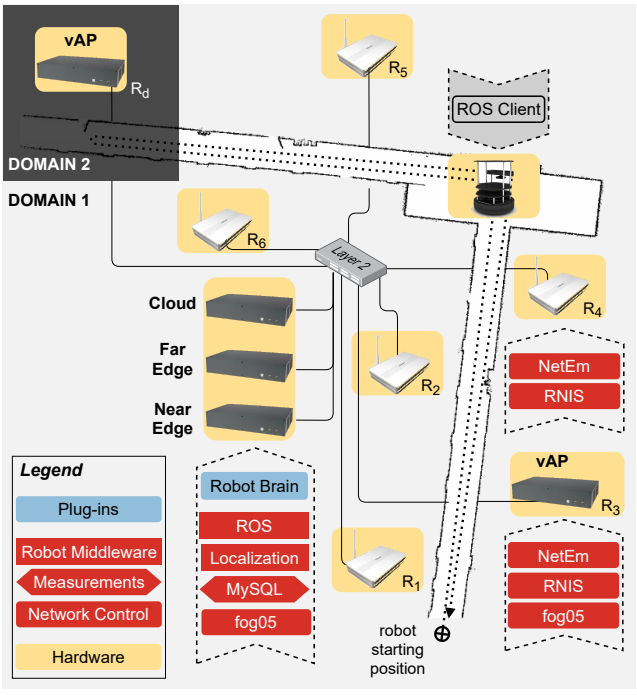


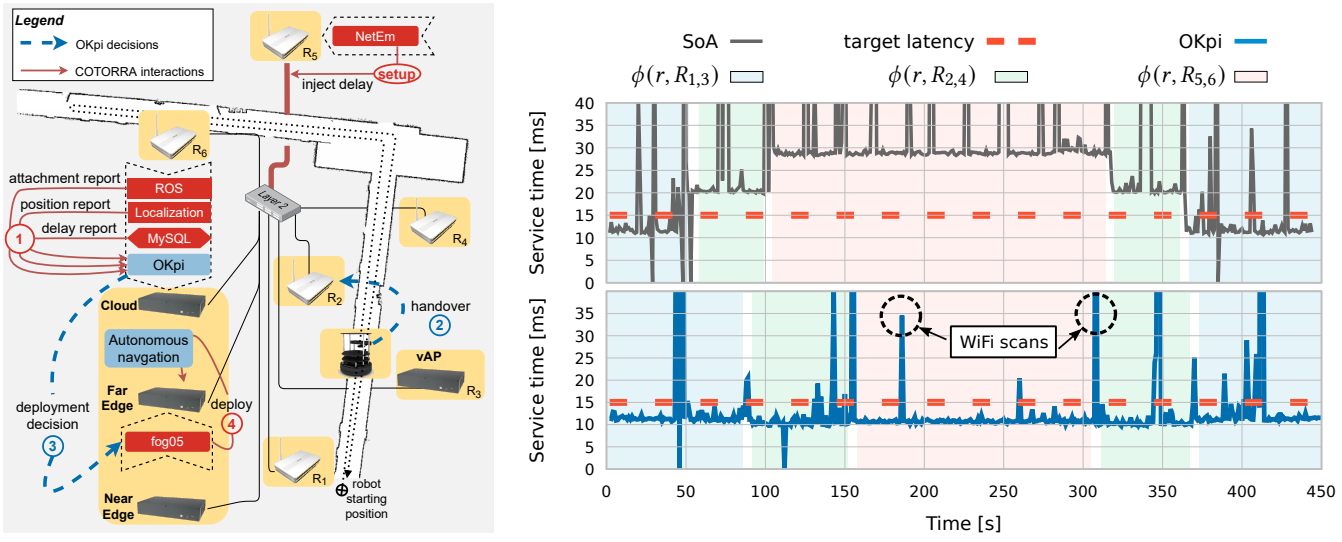
Figure 2: COTORRA testbed in Universidad Carlos III de Madrid facilities.

application. Additionally, Plug-ins can also interact with the testbed by sending to the Core layer instructions such as delay injection $\psi(\cdot)$, or the deployment of a VF v in a server $a(n) = \{v\}$.

3 COTORRA IMPLEMENTATION

Figure 2 shows an implementation of COTORRA testbed based on a mobile robot. This implementation is used to verify the effectiveness of COTORRA in testing innovative mobility features. The COTORRA Hardware layer consists of: (*i*) x5 ASUS WL500G Premium v1 Access Points (APs) running OpenWrt 18.06.2; (*ii*) x5 MiniPC, with x4 vCPUs and 8GB of RAM each. Two MiniPCs are used as APs and the rest as servers. Two of the server MiniPCs simulate Far Edge and Cloud nodes by introducing an artificial queuing delay of 5 and 10 seconds, respectively. The third MiniPC server acts as a local Near Edge node. All APs and MiniPCs are deployed along a corridor of the Universidad Carlos III de Madrid, interconnected with 10Gbps Ethernet connectivity. For the mobile robot, we used a Robotic Operating System (ROS) compatible Kobuki Turtlebot S2 robot equipped with a laptop with 8-GB RAM and 2 CPUs, and a RPLIDAR A2 lidar for 360-degree omnidirectional laser range scanning.

The COTORRA Core layer building blocks are implemented using containers (shown as red rounded boxes on Figure 2). The Robot Middleware building block is implemented using ROS to (*i*) collect and expose the lidar and odometry sensor data $\sigma(r_i)$ to the Plug-ins layer, and (*ii*) send to the robots the navigation instructions $\sigma_p(r_i)$ received from the Plug-ins. In addition, as part of the Robot Middleware building block, we use a Localization service to provide probabilistic robot localization $\mathbb{P}\left(\sigma_p(r_i)\right)$ based on the lidar data. The Network Control building block is implemented using a WLAN



(a) (b)

Figure 3: (a) autonomous robot navigation using COTORRA and OKpi Plug-in; and (b) comparison of the service time experienced by the robot when using a SoA-assisted (top) and OKpi-assisted driving (bottom).

Access Information Service (WAIS) to provide real-time context information (e.g., signal level $\iota(R_i)$, transmission and reception bit rates $\lambda(r_i, R_i)$, etc.) through a REST API. In addition, Network Control emulation is implemented with NetEm, so as to introduce artificial latency and packet loss. And for the Network Control VIM functionalities, we have used Fog05[2] which implements the mapping functionality mentioned in Section 2.1. The Measurements embeddings history (2) is stored and exposed with a a MySQL database. Finally, our COTORRA implementation used a ROS brain as a Plug-in to provide autonomous navigation for the mobile robot through the Robot Middleware building block.

4 EXPERIMENTAL VALIDATION

This Section validates our testbed implementation. For the validation we set up an autonomous driving service consisting of a robot r_i with a driver VF v_r that executes the instructions received from the autonomous navigation Plug-in VF v_d hosted in the Edge. The validation consists in showing that, thanks to the contextual information offered by COTORRA, two different serverless Plug-ins are capable of enhancing the autonomous driving service.

4.1 Service Orchestrator Plug-in

OKpi [6] is a service orchestration algorithm that decides the deployment of VFs and VLs to meet latency constraints. Additionally, OKpi also considers coverage constraints and tells to a robot or end user to which RU it has to attach to experience low latencies. In this experimental validation OKpi is implemented as a COTORRA Plug-in that (*i*) specifies the robot handovers; and (*ii*) migrates the autonomous navigation VF as the robot moves (see Figure 3(a)). The target is to validate that the OKpi Plug-in enhances the autonomous navigation by decreasing the service time below 15 ms.

In the experiment OKpi runs as a COTORRA serverless Plug-in. In particular, it runs as a Python script implementing the Core layer APIs. The OKpi serverless Plug-in listens to the COTORRA Core layer events, which periodically give reports about link delays $d(\cdot)$, robot position $\sigma_p(\cdot)$, and AP attachment $\phi(\cdot)$. Based on these events, OKpi computes what is the best robot-AP attachment, and whether it has to migrate the autonomous driving VF.

As illustrated in Figure 3(a), during the experiment setup the Network Control building block of each AP (implemented with NetEm) injects delay in the links connecting the AP with the Edge and Cloud server. Table 2 specifies the delays injected between APs and servers. These delays follow a normal distribution with average equal to the values reported in Table 2, and a standard deviation of 0.1 times the average value. The goal of the delay injection is to introduce latency heterogeneity. Once the experiment starts, the robot r moves along the hallway, and periodically reports its position $\sigma_p(r)$ to the Localization service, so as its AP attachment $\phi(r, R_i)$ to ROS. Concurrently, the MySQL database stores the network links' delays $d(n_1, n_2)$. All these information is reported as events to the OKpi serverless Plug-in. When OKpi detects that the robot position is leaving the coverage area of an AP, as R_3, it triggers a handover $\phi(r, R_2) = 1$ to AP R_2, and migrates the Autonomous navigation VF v_d to the far Edge server $a(\text{far Edge}) = \{v_d\}$ to keep the service latency below 15 ms (see Table 2). The described handover and migration process, illustrated in Figure 3(a), is repeated throughout the whole robot drive depending on the robot position and AP attachment. Figure 3(b) shows a comparison of what is the service

Table 2: Injected delays in AP-server links for the service orchestration Plug-in experiment

AP	Cloud	Far Edge	Near Edge
R_1, R_3	9 ms	4 ms	3 ms
R_2, R_4	18 ms	8 ms	9 ms
R_5, R_6	27 ms	12 ms	9 ms

[2]Fog05 project: https://fog05.io/ [Accessed: 18 May 2021]

time experienced when the autonomous navigation is enhanced with the OKpi Plug-in, and when it uses a SoA solution consisting in doing handovers based on the signal strength received from periodic WiFi scans (with the autonomous navigation VF hosted in the Cloud). Results show that the OKpi Plug-in enhances the autonomous navigation, and its benefit is three-fold, namely, it (*i*) maintains the service time below 15 ms; (*ii*) reduces the number WiFi scans ; and (*iii*) triggers smarter handovers. Thanks to the OKpi-triggered handovers, the robot does not have to periodically scan the channel to decide the AP attachment. Moreover, OKpi also uses the events with the links' delay information to remain less time attached to the high latency APs – notice in Figure 3(b) how with OKpi the robot is attached for less time to R_5, R_6 than in the SoA solution.

4.2 DLT federation

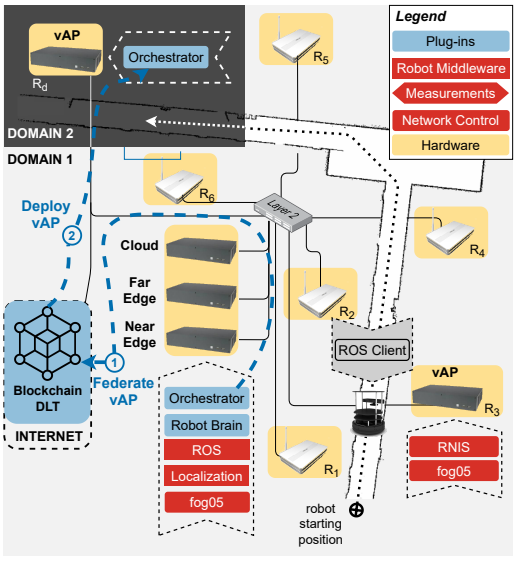
In [8], we designed a federation of a robotic service using a Distributed Ledger Technology - DLT (e.g., Blockchain). Service federation is a set of procedures that enable orchestration of services across multiple administrative domains. We used COTORRA testbed to deploy the DLT solution and perform experimental evaluation of the multi-domain federation. The experiment consists of a moving robot towards out-of-coverage area of Domain 1 (see Figure 2) and extending the connectivity range, on-the-fly, in Domain 2. A serverless Orchestrator Plug-in is deployed in each domain, and a DLT Plug-in is used for the inter-domain interaction.

The mobile robot moves from a starting position (see Figure 4(a)) towards the coverage area of Domain 2. The Robot brain, analyzes the location data from the $\sigma_p(r)$ Localization service, and triggers a federation event. The Domain 1 Orchestrator starts "Federate vAP" (step 1 on Figure 4(a)) through the DLT Plug-in. Both Orchestrators negotiate through the DLT Plug-in, and the "Deploy vAP" event $a(n_{AP}) = \{v_{AP}\}$ is sent to Domain 2 Orchestrator (step 2 on Figure 4(a)). The Domain 2 Orchestrator initiates deployment of vAP on *Rd*. The mobile robot attaches $\phi(r, n_{AP}) = 1$ to the newly deployed vAP in Domain 2 without any (movement or robotic service) interruption.

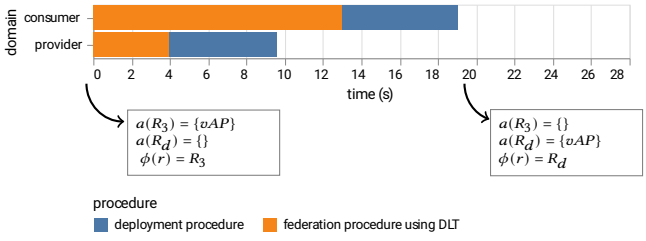
On Figure 4(b), the results show that both Orchestrators negotiate for 4 seconds, and Domain 2 Orchestrator deploys the new vAP for less than 6 seconds. Domain 1 Orchestrator awaits confirmation for 8 seconds and instructs the mobile robot to switch to the new vAP. The whole federation procedure takes around 19 seconds to complete, a tenfold less than a NFV-MANO based solution [5].

5 CONCLUSION

In this paper, we introduce COTORRA, a modular tesbed built to enable and validate innovative mechanisms of serverless applications in robotic system. Additionally, we showed that COTORRA building blocks can be easily implemented over commodity hardware offering support for a rapid prototyping and validation. The network emulation feature in COTORRA can simulate unpredictable network conditions that enable near production robotic environment where new pluggable applications can be tested. We validated this by running an orchestration algorithm that interacts with both the network infrastructure, and mobile robots. Furthermore, using



(a)



(b)

Figure 4: (a) DLT experimental setup; (b) Using COTORRA to measure DLT-negotiation times, and vAPs deployment times in a federated environment

COTORRA we tested a DLT application for the federation of an Edge robotic service.

ACKNOWLEDGMENTS

This work has been partially funded by the EU H2020 5GROWTH Project (grant no. 856709) and by the H2020 collaborative Europe/Taiwan research project 5G-DIVE (grant no. 859881).

REFERENCES

[1] D. Lu et al. 2016. VC-Bots: A Vehicular Cloud Computing Testbed with Mobile Robots. In *Proceedings of the First International Workshop on Internet of Vehicles and Vehicles of Internet* (Paderborn, Germany) *(IoV-Vol '16)*. Association for Computing Machinery, New York, NY, USA, 31–36. https://doi.org/10.1145/2938681.2938683

[2] B. Bangerter et al. 2014. Networks and devices for the 5G era. *IEEE Communications Magazine* 52, 2 (2014), 90–96.

[3] G. Mohanarajah et al. 2015. Rapyuta: A Cloud Robotics Platform. *IEEE Transactions on Automation Science and Engineering* 12, 2 (2015), 481–493. https://doi.org/10.1109/TASE.2014.2329556

[4] G. Portaluri et al. 2019. Open CLORO: An Open Testbed for Cloud Robotics. In *2019 IEEE 24th International Workshop on Computer Aided Modeling and Design of Communication Links and Networks (CAMAD)*. IEEE, Limassol, Cyprus, 1–5. https://doi.org/10.1109/CAMAD.2019.8858492

[5] J.B. Hortiguela et al. 2020. Realizing the network service federation vision: Enabling automated multidomain orchestration of network services. *IEEE Vehicular Technology Magazine* 15, 2 (2020), 48–57.

[6] J. Martín-Pérez et al. 2020. OKpi: All-KPI Network Slicing Through Efficient Resource Allocation. In *IEEE INFOCOM 2020*. 804–813. https://doi.org/10.1109/INFOCOM41043.2020.9155263
[7] J. Wan et al. 2016. Cloud Robotics: Current Status and Open Issues. *IEEE Access* 4 (01 2016), 1–1. https://doi.org/10.1109/ACCESS.2016.2574979
[8] K.Antevski et al. 2020. DLT federation for Edge robotics. In *2020 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN)*. 71–76. https://doi.org/10.1109/NFV-SDN50289.2020.9289887
[9] K. Sugiura et al. 2015. Rospeex: A cloud robotics platform for human-robot spoken dialogues. In *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 6155–6160. https://doi.org/10.1109/IROS.2015.7354254
[10] M. Schmittle et al. 2018. OpenUAV: A UAV Testbed for the CPS and Robotics Community. In *2018 ACM/IEEE 9th International Conference on Cyber-Physical Systems (ICCPS)*. 130–139. https://doi.org/10.1109/ICCPS.2018.00021
[11] R. C. Mello et al. 2019. Cloud Robotics Experimentation Testbeds: a Cloud-Based Navigation Case Study. In *2019 IEEE 4th Colombian Conference on Automatic Control (CCAC)*. 1–6. https://doi.org/10.1109/CCAC.2019.8921387
[12] T. Taleb et al. 2017. On Multi-Access Edge Computing: A Survey of the Emerging 5G Network Edge Cloud Architecture and Orchestration. *IEEE Communications Surveys Tutorials* 19, 3 (2017), 1657–1681.
[13] W. Shi et al. 2016. Edge Computing: Vision and Challenges. *IEEE Internet of Things Journal* 3, 5 (2016), 637–646.
[14] ETSI MEC. 2019. Network Transformation; (Orchestration, Network and Service Management Framework). V1.
[15] Claus Pahl. 2015. Containerization and the PaaS Cloud. *IEEE Cloud Computing* 2, 3 (2015), 24–31. https://doi.org/10.1109/MCC.2015.51