POLITECNICO DI TORINO
Repository ISTITUZIONALE

Benchmarking Different Strategies for Offloading ROS2 Computation to the Edge

*Original*
Benchmarking Different Strategies for Offloading ROS2 Computation to the Edge / Cacciabue, Daniele; Marino, Jacopo; Aglieco, Francesco; Levorato, Marco; Perroni, Domenico; Risso, Fulvio. - (2024), pp. 49-54. (Intervento presentato al convegno 2024 IEEE 10th International Conference on Network Softwarization (NetSoft) tenutosi a St. Louis, MO (USA) nel 24-28 June 2024) [10.1109/NetSoft60951.2024.10588914].

*Availability:*
This version is available at: 11583/2988320 since: 2024-09-12T11:33:45Z

*Publisher:*
IEEE

*Published*
DOI:10.1109/NetSoft60951.2024.10588914

*Terms of use:*

This article is made available under terms and conditions as specified in the corresponding bibliographic description in the repository

*Publisher copyright*
IEEE postprint/Author's Accepted Manuscript

©2024 IEEE. Personal use of this material is permitted. Permission from IEEE must be obtained for all other uses, in any current or future media, including reprinting/republishing this material for advertising or promotional purposes, creating new collecting works, for resale or lists, or reuse of any copyrighted component of this work in other works.

(Article begins on next page)

19 December 2024

# Benchmarking Different Strategies for Offloading ROS2 Computation to the Edge

Daniele Cacciabue*†, Jacopo Marino*†§, Francesco Aglieco¶,
Marco Levorato§, Domenico Perroni‡, and Fulvio Risso†
†Dept. of Control and Computer Engineering, Politecnico di Torino, Torino, Italy
§Dept. of Computer Science, University of California, Irvine, CA, USA
¶Links Foundation, Torino, Italy
‡Italdesign, Torino, Italy

***Abstract*—Mobile robots suffer from inherent limitations due to the tradeoff in the amount of energy consumed by their on-board processing components, and the need to increase their operational time. On the communication side, the volatility of communication links severely hinders the ability of a mobile device to rely on computation offloading. The challenge addressed by this paper is the development of a methodology and framework to effectively migrate the location of a service from a system to another, minimizing downtime and striving to reduce any side-effects that may be perceived by the system. Solving this challenge will pave the way for more effective computation offloading solutions that can cope with the unpredictability of the edge systems. Four different approaches are compared, analyzing their performance via an empirical approach. The insights gathered from data allow the identification of the most promising solution to address the aforementioned challenge.**

***Index Terms*—kubernetes, ros2, liqo, cloud offloading, service switching**

## I. INTRODUCTION

Today's robots (such as rovers, drones and more) are equipped with on-board computing hardware to execute perception tasks that allow them to interact with their surroundings, for instance by identifying obstacles and performing autonomous navigation. However, some Machine Learning (ML) tasks for perception such as object detection, can require a considerable amount of computing power. Properly dimensioning the hardware is highly non-trivial. In fact, one must account for the tasks the robot will need to perform and their requirements, for the expected mission duration and cost.

A very promising approach is to let the robot delegate a subset of its tasks to a separate – less-constrained – system. Offloading computation may, depending on the scenario, effectively grant many advantages, including providing the robot with extended computational capacity, an increased operational time or an improved context awareness. The key challenge to tackle to enable robots with extended computing capabilities is to develop novel methodologies for the robot to communicate with the cloud, and seamlessly integrate its functionalities with it, which is one of the core tenets Mobile Edge Computing (MEC) [1], [2]. Practically speaking, a successful integration of the MEC paradigm in robotics and Internet of Things (IoT) in general can allow for smaller robots to use complex, resource-intensive algorithms such as ML and autonomy algorithms even if their system specifications are severely undersized, resulting in cheaper, simpler robots outperforming their more expensive counterparts.

The main objective of this paper is to develop an effective methodology to offload the execution of stateless ROS2 modules to edge clouds. The main requirement is to guarantee service continuity and transparency, that is, if a ROS2 system is completely or partially moved to the cloud (or vice versa), it must be able to continue working correctly, minimizing service disruption. We performed a preliminary exploration this paradigm in a previous paper [3] and in this paper we analyze four core concepts to implement it. Through the analysis of their performance, we aim to identify the most suitable technology or approach to reach such goal.

We define *switching* as the process of transitioning the active logic of a component from one location to another. The ideal switching is fast and is undetectable by any other entity in the system that interacts with the migrated component. This capability is central to the solutions discussed in this paper, the four approaches will be analyzed according to their distance from ideal switching. In switching, computation *moves* from one location to another. However, the process whose function has changed location may not have moved as well. Especially in mission-critical settings, the old copy of the function might be kept running in the original location, but prevented from communicating with the rest of the system. This can be done to deploy a fallback-mechanism that allows the system to rapidly re-enable the old instance if the remote become unreachable. In cases in which such redundant local copy is not needed, the solution is to add a re-deployment step to the switching procedure. In this case, after having moved the computation, the old logic component is terminated, turning the switching operation into a full-fledged migration. Of the four solutions studied in this paper, two will include a re-deployment step.

The structure of this paper is outlined as follows. Section II introduces the current state of the art, while Section III goes into the technologies employed. The architectures of the solutions proposed are discussed in Section IV, with their corresponding implementations described in Section V.

*Daniele Cacciabue and Jacopo Marino are co-first authors.

The methodology for experimental evaluation and the results obtained are presented in Section VI. The paper concludes with Section VII, outlining possible future directions.

## II. State of the Art

Chen et al. [4] introduced FogROS, a tool for easily deploying robot software components to the cloud via the Robot Operating System (ROS), enhancing computing resources like GPUs with minimal setup. Building on this, Ichnowski et al. [5] presented FogROS2 for ROS2, enabling simple cloud and fog computing integration for robots. This platform allows for shifting heavy computations to the cloud with few script changes, without altering the robot's code, focusing on Virtual Machines (VMs) rather than Kubernetes (K8s) for cloud interactions. Expanding on these concepts, Chen et al. [6] later developed FogROS2-SGC, furthering FogROS2's capabilities to connect robots across different locations securely and efficiently, without requiring code changes. Similarly, Anand et al. [7] introduced an algorithm designed to mitigate the constraints of serverless computing, such as communication and bandwidth issues, employing various work-sharing strategies to enhance cost efficiency and reduce execution time.

Doan et al. [8] proposed a framework that separates MEC application design into processing and state management, using a distributed key-value store to ensure seamless service continuity. This design enables state synchronization across MEC servers and anticipates user handovers between MEC nodes. Its modular, containerized approach enhances its practical applicability in MEC settings.

Machen et al. [9] tackled the challenge of minimizing service downtime and overall migration time in mobile edge clouds. They developed a layered framework that breaks down cloud applications into multiple layers, allowing for the transfer of only the missing layers to the destination. This framework is applicable to both VMs and containers and can be implemented with existing tools.

Chebaane et al. [10] introduced a method for offloading time-sensitive tasks from the initiating application to nearby Fog nodes using Docker containers and Checkpointing. This approach results in a layer-oriented framework that limits offloading to essential steps within just two message exchanges. The strategy assumes that offloading occurs solely from the vehicle hosting the application to an adjacent Fog node.

Some researches have concentrated on the live migration of VMs [9], [11], transferring all VM data during runtime. Conversely, other research has explored the live migration of containers [9], [12], emphasizing the movement of containerized applications without interrupting their operation.

Our approach differentiates from the above solutions because it does not handle the case of stateful migration, moreover, it analyzes specifically the migration of ROS2-based services, during execution. Unlike Machen et al. [9], our approach does not tackle primarily a client-server communication paradigm, but mainly focuses on a client-subscribe one, which ROS2's middleware is based on. This study differentiates from FogROS2 [5] due to the selection of Kubernetes and containers instead of VMs, which allows for a lower resource footprint, easier management and the ability of bringing the the benefits and practices of cloud-native development to robotics.

## III. Technologies

This section explores four technologies, each of which contributes distinctive capabilities to our solution.

### A. ROS2

The Robot Operating System (ROS) [13] includes an extensive array of software libraries and tools aimed at streamlining the creation of robotic applications. It provides a wide range of features, from fundamental drivers to state-of-the-art algorithms, alongside sophisticated tools for developers. In ROS, every Node is designed to fulfill a specific function, such the wheel motors management or sensor data transmission from a laser range-finder. ROS uses publisher/subscriber messaging between Nodes, using the standard called Data Distributed Services (DDS). Nodes could be placed in the same device or they could be distributed across different machines, if the latter are able to communicate using multicast.

### B. Zenoh

Zenoh introduces a Pub/Sub/Query protocol, offering unified abstractions for managing data in motion, data at rest, and computations on an Internet scale. It is optimized for performance across a wide range of hardware and network conditions, from server-grade environments to microcontrollers and networks with limited resources. Moreover, Zenoh supports peer-to-peer, routed, and brokered interactions [14].

### C. Kubernetes

Kubernetes is an open-source platform that automates the deployment, scaling, and management of containerized applications. It is designed to be portable and extensible, facilitating the orchestration of containerized workloads and services through both declarative configuration and automation. Complementing it, the Container Network Interface (CNI) serves as a framework for dynamically configuring networking resources. The CNI defines an interface for network configuration and IP address provisioning, essential for maintaining connectivity in Kubernetes environments.

### D. Liqo

Liqo is an open-source tool that enhances Kubernetes by enabling dynamic multi-cluster configurations across varied infrastructures like on-premise, cloud, and edge. It introduces a virtual node after connecting clusters, representing shared resources from the remote cluster. This process seamlessly expands the local cluster's resource pool, allowing Kubernetes' scheduler to efficiently allocate workloads. Liqo maintains compatibility with standard Kubernetes APIs, ensuring offloaded pods are managed as if they were local [15].

## IV. Switching Architectures

Within the scope of implementing switching capabilities using ROS2, we identify four potential approaches. Solutions are presented side by side, highlighting that the effectiveness of a strategy varies with the scenario, making it impossible to choose a one-size-fits-all option in advance. A key difference in the solutions proposed stands on the fact that two of them switch computation by redeploying the Pod, the other two rely on preemptively deploying multiple copies of the Pod in different locations and only keeping one of them active at the same time. In our discussion we will refer to two Kubernetes clusters: one designated as *robot* and the other as *cloud*.

### A. Switching with Redeployment (SwR)

In addressing the challenge of efficiently allocating ROS2 Nodes within switching architectures, the pivotal issue becomes their placement. This dilemma can be reframed as a scheduling challenge, focusing on identifying the most appropriate Kubernetes node and cluster for hosting the ROS2 Node. The SwR architecture emerges from this concept, entrusting a scheduler with the task of initially deploying the ROS2 Node and subsequently executing a rollout to relocate it to another location as required. Initially, the scheduler operates in a basic capacity, merely transferring the ROS2 Node between locations without incorporating any metrics or contextual analysis. The responsibility for managing the Pod lifecycle is transferred to Kubernetes. Therefore, once the scheduler has assigned the ROS2 Node to a specific cluster node, Kubernetes assumes control over all further processes, adhering to its standard operational procedures.

### B. Switching with Redeployment and Network Policies (SwRNP)

Building upon the SwR model, the SwRNP architecture introduces an additional step following the scheduling phase. After a new ROS2 Node instance is scheduled, enters the *Running* state, and the previous node transitions to *Terminating*, the scheduler deploys a Network Policy (NetPol). This policy blocks all outgoing traffic from the old ROS2 Node, minimizing the overlap of DDS messages between the old and new Nodes. The introduction of NetPol ensures that the outgoing messages from the terminating ROS2 Node do not affect the newly configured Node. After deleting the old ROS2 Node, the NetPol is removed, as it is no longer necessary.

### C. Switching with Network Policy (SwNP)

This solution is not based on redeployment. It starts from the condition in which more than one copy is active at the same time, one in the *robot* cluster and one in the *cloud* cluster. By default, only one of the two instances is active at the same time. This is implemented using a NetPol to block any outgoing communication from the disabled instance, preventing it from communicating. When the switch needs to be performed, the inactive Pod needs to be enabled and the disabled Pod enabled, which allows for effectively switch the active logic from one location to another. Since this solution does not leverage the existence of the Kubernetes scheduler, enabling and disabling of a Pod needs to be performed by a custom-built orchestrator, which communicates with the Kubernetes API server to enforce the NetPol.

### D. ROS2 Lifecycle Node (LCN)

This solution, still not based on redeployment, is based on a ROS2 feature called Lifecycle Nodes. The latter are a special type of Node in ROS that can be managed through four primary states.

- **Unconfigured**: initial state of the Node after instantiation; the node is ready for configuration but not yet operational.
- **Inactive**: state of a Node that is not performing any processing; its main purpose is to provide a state where a Node's behaviour can be changed (re-configured) without requiring it to be in active state.
- **Active**: once ready to be fully functional, the Node transitions to the active state; at this point, it has all the behaviors of a standard ROS Node – it responds to service requests, reads and processes data, and produces output.
- **Finalized**: terminal state; the only transition from here is to be destroyed, which frees up the Node's memory.

A node can be asked to change state using proper service-based interfaces exposed by every Lifecycle Node. A custom-built orchestrator can implement the switching algorithm using such services, suitably activating or deactivating node instances. A deactivated node executes less code, which results in a lower battery consumption compared to the the solutions based on NetPol. This approach can only be used with ROS2 Lifecycle Nodes, making it a less portable solution that relies on ROS2-specific features instead of Kubernetes.

## V. Implementation

The experimental setup was designed to fully utilize the capabilities of Zenoh, Kubernetes, and Liqo. This architecture encompasses two VMs hosting Kubernetes clusters, labeled as the *Robot* and the *Cloud*, and it is depicted in Figure 1. Each VM is configured with 4 CPU cores and 8 GB of RAM, and shares a common virtualized Ethernet network. Within this setup, ROS2 Nodes communicates using DDS middleware, which relies on UDP multicast traffic to implement peer-discovery. Both clusters were configured using K3s, a lightweight Kubernetes distribution, with Cilium serving as the Container Network Interface (CNI). This configuration was specifically chosen due to K3s' default CNI (Flannel) supporting multicast within the cluster. In case of managed Kubernetes, the choice of CNI is up to the cloud provider and not to the single tenant. By taking the most restrictive option, this study ensures that the solution is always applicable, independently of the chosen CNI supporting multicast or not.

The *robot* cluster presents the same architecture as the *cloud* cluster, it employs the same CNI and the same Pod organization. This mirrored architecture across both clusters allows for deployment transparency, since the redeployed Pod does not need to be modified depending on the nature of the CNI where it is running. This simplification allowed for the

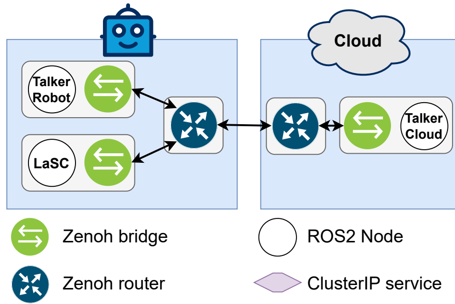



Fig. 1: Architecture interconnecting cloud and robot cluster.

use of a simpler redeployment logic, which does not need to take into account details such as the CNI installed in each cluster. The challenge of integrating the presented solutions in clusters with heterogeneous CNIs is left for future works.

To circumvent the absence of multicast, a Zenoh-based overlay network was devised, which allows discovery traffic to be forwarded to any other ROS2 Node, bypassing limitations due to the CNI. Our approach integrates the Zenoh Plugin ROS2DDS, functioning as a bridge within a sidecar container in each Pod, alongside the main container running the ROS2 Node These bridges connect to the central Zenoh Router within the cluster, facilitating seamless DDS communication between ROS2 Nodes across different clusters.

Liqo was installed and used to establish a peering from the *robot* towards the *cloud* cluster. Liqo enables the *robot* cluster to easily use resources from the *cloud* cluster: it creates a namespace on the *cloud* cluster and it is managed transparently, as if it was local. This greatly simplifies the management of multiple clusters, eliminating the need for exposing Zenoh routers through a Load Balancer, managing IP address changes, and instead relying on using a Fully-Qualified Domain Name (FQDN) for the service in the remote cluster.

### A. Test environment

The basic structure of the test environment consisted of two ROS2 Nodes, namely *talker* and *listener*. The *talker* publishes a message every $100ms$ on a topic, in the form of *location: counter*, where *location* is the Kubernetes cluster on which the Pod is currently scheduled (e.g., *robot*) and *counter* is an increasing number, so each message will have the counter of the previous message increased by 1. The *listener* subscribes to the same topic of the talker and waits for messages. When it receives a message, this is written to a InfluxDB instance which records it together with the relative timestamp.

The *Switch Controller (SC)* is a component responsible for triggering or enacting the switch transition. When the system starts, the SC sets the active talker to be on the robot and it listens on the talker topic to ensure that there is only one instance of talker active at the given time. When such condition is met, it triggers the switching procedure, recording the event in the InfluxDB instance. One important implementation decision regarding this component was to decide a convention regarding the order in which a Node is enabled or disabled. This choice matters for the SwNP and LCN case, because they do not rely on the Kubernetes scheduler to execute their logic. Since the default behaviour used by Kubernetes to migrate a container is to first start a new copy on another cluster node, then to terminate the old copy, the choice was to implement the SC to first enable the currently disabled node, and only afterwards to disable the previously active one. The SC is also able to monitor the state of the Pod, checking when it transitions to *Running* or when the Pod is effectively deleted. The actual implementation resulted in the merging of the *listener* and the SC into the same ROS2 Node, which we will now refer to as *Listener and Switch Controller (LaSC)*. Such Nodes were deployed on the architecture in Figure 1.

## VI. Experimental Evaluation

To assess the duration needed for the architectures to accomplish the switching, we identified three distinct intervals that offer insights into the swiftness of the switching procedure. These intervals illustrate each architecture's capability to either switch ROS2 Nodes or to activate/deactivate them. The intervals are characterized as follows.

- **Time t1:** measured from the moment the switch request is initiated to the point where the new ROS2 talker begins to transmit data. This encompasses the period from issuing the switch command to the instance the listener detects the initial message from the new talker (Figure 2a).
- **Time t2:** defined as the duration from the first message sent by the new talker to the moment the listener ceases to obtain data from the previous talker. It can assume either a positive value (Figure 2b) or negative (Figure 2c).
- **Time t3:** time interval from the last message sent by the old talker to the time when the latter is fully deleted.

### A. Metrics collection

The relevant metrics have been collected using an InfluxDB instance. InfluxDB has been selected as the database of choice because of its time-series structure which allows not only to register events, but also the time at which they have occurred. The gathered data has then be processed offline, which allowed for decoupling the data collection from the data analysis phase.

The LaSC inserts into InfluxDB the events related to (i) the arrival of a new message, (ii) the time at which the switching procedure is started, (iii) the time at which the talker Pods change their state to *Running* or are effectively deleted.

Data was gathered through 200 measurements and it has been organized in two InfluxDB buckets. The *listener metrics* bucket stores information regarding the direction of the switching (i.e. robot-to-cloud or cloud-to-robot), the messages sent by each talker, and the measurement iteration (e.g., 0, 1, 2, etc.). The *switch controller metrics* bucket is dedicated to track the other metrics we need to compute the final times $t1$, $t2$, and $t3$. These metrics are the timestamps related to the switching issued commands, when the new ROS2 Node is running and when the old ROS2 Node has been deleted.

After the metrics have been collected, the data was processed to compute the times $t1$, $t2$, and $t3$ for all of the four architectures. Such data was gathered with the intent of

allowing to easily compare the differences in switching time for each solution. The computed times $t1$, $t2$, and $t3$ have been drawn on boxplot graphs, to easily compare them.

### B. Time $t1$

The results regarding the time $t1$ for the robot-to-cloud and cloud-to-robot case are shown in Figure 3a and Figure 3b respectively. The data shows some slight differences between the two transitions, but the results show similar performance independently of the messages direction.

Unsurprisingly, the two cases with redeployment (SwR and SwRNP) perform worse as far as time $t1$ is concerned, this is due to them requiring time to deploy the new talker Node on the other cluster. On the other hand, the two solutions with redeployment (SwR and SwRNP) and the one with only NetPol (SwNP) show how the LCN solution performs dramatically better, showing a clear downside of using NetPol when the need is to quickly restore the network traffic. Considering $t1$, the most performing solution is the LCN, which, considering the average values, allows for a sensibly faster switching time reduced of at least 85% compared to the other solutions.

### C. Time $t2$

Given that both the default behaviour of the Kubernetes scheduler and the implementation of the SC prioritize the activation of a new node over disabling an old one, it was natural to assume that the new talker would be able to send a message before the old talker was terminated, resulting in strictly-positive samples of $t2$ signalling an overlap condition.

The results depict a different picture and only the SwR case conformed to the expectations. Both overlap and silence have been measured in the SwRNP. In the SwNP case, only one sample of overlap was measured out of 400 measurements comprising both the robot-to-cloud and cloud-to-robot directions, effectively showing how, the previous hypothesis does not apply to this type of switching. The LCN case showed both overlap and silence as well, but in this case, since the activation and deactivation of the Lifecycle Nodes was made asynchronously, some kind of silence was expected, due to the not deterministic nature of such kind of RPC calls.

Due to the presence of both positive and negative values, the time $t2$ was split into two subcategories respectively called $t2_{overlap}$ and $t2_{silence}$. The boxplot graphs in Figure 3, show the relevant silence and overlap statistics. The SwR overlap case (there was no silence in for SwR) has been excluded from the graph as it showed a median value of $(29.9084 \pm 0.5541)s$ for the Robot to Cloud and $(29.412 \pm 0.6144)s$ for the Cloud to Robot. Being this value significantly higher than all the ones from the other solutions, it was not included in Figure 3 with the purpose of better showing the difference in performance among the remaining solutions. The SwNP Overlap boxplot is missing because it consisted of only one event (outlier).

Another unexpected result shown by this graph is the disappointing performance of the SwNP case which was expected to perform similarly to SwRNP. One possible explanation for this result is that Kubernetes takes significantly less time to apply a NetPol than to remove it, thus the SwRNP, which relies on NetPol only to disable the node that is scheduled for termination, is not affected by the time required to remove it.

This hypothesis also explains why, in our measurements, the SwNP case does not show any $t2_{overlap}$, further investigation on this topic will be left for future works, from the graphs shown, the LCN case clearly resulted in being the most performant also with respect to the $t2$ parameter.

### D. Time $t3$

Since $t3$ is the time interval between the last message of a talker to its actual deletion, it is a measure that bears meaning only for the switching cases that include a redeployment step, i.e. SwR and SwRNP. Since LCN and SwNP never delete the old talker but only disable it, they will not be considered further in this subsection. The main result derived by this interval consists in quantifying how, the application of a NetPol on the old talker, allows for a reduction in the amount of unnecessary messages sent on the DDS multicast network. In our metrics, we saw a time of $(1.2455 \pm 0.3043)s$ Robot to Cloud and $(1.2315 \pm 0.2734)s$ Cloud to Robot for the SwR, and $(31.3253 \pm 0.3622)s$ Robot to Cloud and $(31.2824 \pm 0.3457)s$ Cloud to Robot for the SwRNP. This difference is due to the fact that in the SwR case, the Pod is deleted practically immediately after the last message arrives. In the SwNRP case the Pod is first silenced by the NetPol and then deleted afterwards, thus the time $t3$ increases.

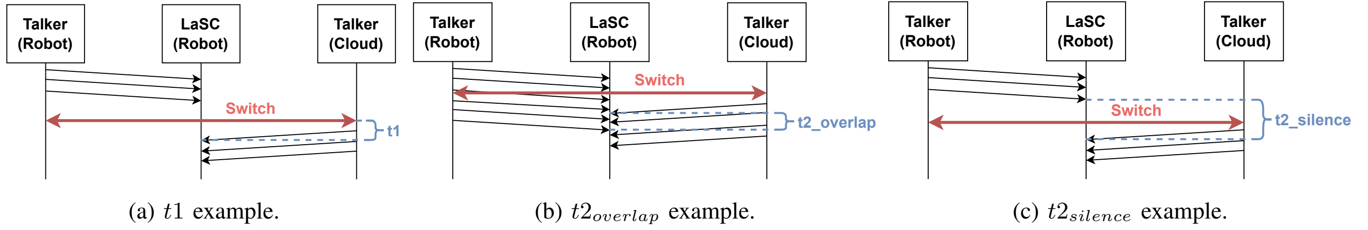
### E. Discussion of results

The results show that the Lifecycle Node architecture outperforms all other options evaluated, which makes it the most promising solution to enable a swift and efficient transition of a ROS2 Node from one cluster to another. In hindsight, given the reached performance, further tests may be required in order to test the maximum publishing frequency that can be supported by the Lifecycle Nodes solution while keeping the $t1$ and $t2$ metrics as low as possible.

As far as the solutions which incorporate a redeployment phase, we hypothesize that the optimal switching with redeployment solution would closely resemble SwRNP, but would utilize Lifecycle Nodes to disable the old instance instead of NetPol. This kind of solution would be able to leverage the Kubernetes scheduler to move the deployment from cluster to cluster and would leverage the very good Lifecycle Nodes performance to minimize any problems regarding the presence of either overlap or silence.

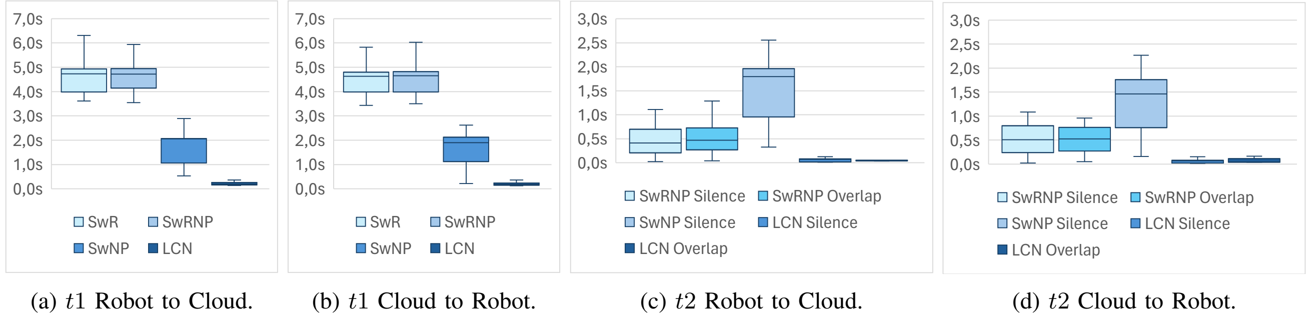
## VII. Conclusions

Our study has made significant strides in evaluating the performance of different methods for switching stateless, ROS2 computation from one Kubernetes cluster to another using a reference architecture based on Liqo, Zenoh and Kubernetes. We also shown how integrating a method to halt communication from the old Pod can be employed in ROS2 use cases to mitigate the issue of duplicate messages, allowing a stateless workload to be effectively redeployed




(a) $t1$ example. (b) $t2_{overlap}$ example. (c) $t2_{silence}$ example.

Fig. 2: Definition of $t1$ and $t2$; $t2$ can be positive ($t2_{overlap}$) or negative ($t2_{silence}$).




(a) $t1$ Robot to Cloud. (b) $t1$ Cloud to Robot. (c) $t2$ Robot to Cloud. (d) $t2$ Cloud to Robot.

Fig. 3: Time $t1$ and $t2$.

without sacrificing business continuity. Our findings provide a measure of switching speed of ROS2 Nodes from one cluster to another using various technologies.

However, several topics warrant further investigation. These include quantifying the time needed by Kubernetes to enforce a NetPol once created and to delete it. By understanding what makes the solution based on NetPol less efficient, it may be possible to make it a suitable competitor to the LCN solution, as it allows for the switching algorithm to be applied to other applications other than ROS2-based ones. The four solutions analyzed in this paper are not meant to be exhaustive and there may be other suitable technologies and ideas to implement the switching operation. The identification and the analysis of them is another research path that deserves to be taken.

## Acknoledgements

This work was partly supported by European Union's Horizon Europe research and innovation programme under grant agreement No 101070473, project FLUIDOS (Flexible, scaLable, secUre, and decentralIseD Operating System).

This research was conducted as part of Daniele Cacciabue and Jacopo Marino Ph.D. programmes, under the financing of the Piano Nazionale di Ripresa e Resilienza (PNRR) and the NextGenerationEU initiative.

## References

[1] S. Wang, J. Xu, N. Zhang, and Y. Liu, "A survey on service migration in mobile edge computing," *IEEE Access*, vol. 6, Apr 2018.

[2] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli, "Fog computing and its role in the internet of things," in *Proceedings of the First Edition of the MCC Workshop on Mobile Cloud Computing*, Helsinki, Finland, Aug. 2012, p. 13–16.

[3] D. Cacciabue, F. Aglieco, D. Perroni, and F. Risso, "Stateless job offloading for mobile robots in kubernetes," in *1st International Workshop on MetaOS for the Cloud-Edge-IoT Continuum (MECC 2024)*, Athens, Greece, Apr. 2024.

[4] Kaiyuan, Chen, Y. Liang, N. Jha, J. Ichnowski, M. Danielczuk, J. Gonzalez, J. Kubiatowicz, and K. Goldberg, "Fogros: An adaptive framework for automating fog robotics deployment," in *2021 IEEE 17th International Conference on Automation Science and Engineering (CASE)*, Lyon, France, Aug. 2021, p. 2035–2042.

[5] J. Ichnowski, K. Chen, K. Dharmarajan, S. Adebola, M. Danielczuk, V. Mayoral-Vilches, N. Jha, H. Zhan, E. LLontop, D. Xu, C. Buscaron, J. Kubiatowicz, I. Stoica, J. Gonzalez, and K. Goldberg, "Fogros2: An adaptive platform for cloud and fog robotics using ros 2," in *2023 IEEE International Conference on Robotics and Automation (ICRA)*, London, UK, Jul. 2023, pp. 5493–5500.

[6] K. Chen, R. Hoque, K. Dharmarajan, E. LLontopl, S. Adebola, J. Ichnowski, J. Kubiatowicz, and K. Goldberg, "Fogros2-sgc: A ros2 cloud robotics platform for secure global connectivity," in *2023 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Detroit, MI, USA, Dec. 2023.

[7] R. Anand, J. Ichnowski, C. Wu, J. M. Hellerstein, J. E. Gonzalez, and K. Goldberg, "Serverless multi-query motion planning for fog robotics," in *2021 IEEE International Conference on Robotics and Automation (ICRA)*, Xi'an, China, May-Jun. 2021, pp. 7457–7463.

[8] T. V. Doan, Z. Fan, G. T. Nguyen, H. Salah, D. You, and F. H. P. Fitzek, "Follow me, if you can: A framework for seamless migration in mobile edge cloud," in *IEEE Conference on Computer Communications (INFOCOM)*, Toronto, ON, Canada, Jul. 2020, pp. 1178–1183.

[9] A. Machen, S. Wang, K. K. Leung, B. J. Ko, and T. Salonidis, "Live service migration in mobile edge clouds," *IEEE Wireless Communications*, vol. 25, no. 1, pp. 140–147, Feb. 2018.

[10] A. Chebaane, S. Spornraft, and A. Khelil, "Container-based task offloading for time-critical fog computing," in *2020 IEEE 3rd 5G World Forum (5GWF)*, Bangalore, India, Sept. 2020, pp. 205–211.

[11] H. Jin, L. Deng, S. Wu, X. Shi, and X. Pan, "Live virtual machine migration with adaptive, memory compression," in *IEEE International Conference on Cluster Computing, ICCC*, Aug.-Sept. 2009, pp. 1–10.

[12] S. Kakakhel, L. Mukkala, T. Westerlund, and J. Plosila, "Virtualization at the network edge: A technology perspective," in *2018 Third International Conference on Fog and Mobile Edge Computing (FMEC)*, Barcelona, Spain, Apr. 2018, pp. 87–92.

[13] S. Macenski, T. Foote, B. Gerkey, C. Lalancette, and W. Woodall, "Robot operating system 2: Design, architecture, and uses in the wild," *Science Robotics*, vol. 7, no. 66, 2022.

[14] A. Corsaro et al., "Zenoh: Unifying communication, storage and computation from the cloud to the microcontroller," in *2023 26th Euromicro Conference on Digital System Design (DSD)*, Sep. 2023, pp. 422–428.

[15] M. Iorio, F. Risso, A. Palesandro, L. Camiciotti, and A. Manzalini, "Computing without borders: The way towards liquid computing," *IEEE Transactions on Cloud Computing*, pp. 1–18, 2022.