SM2: Towards Reliable and Scalable Robotics Middleware Beyond ROS2

SM2: Towards Reliable and Scalable Robotics Middleware Beyond ROS2

Andrea Fasano†, Daniele Cacciabue*, Stefano Galantino*, Jacopo Marino*, Fulvio Risso*
Dept. of Control and Computer Engineering
Politecnico di Torino
Torino, Italy
†andrea17.fasano@gmail.com
*{first}.{last}@polito.it

***Abstract*—Modern robotic systems increasingly rely on modular software frameworks such as Robot Operating System 2 (ROS2) to support real-time, high-throughput communication across distributed components. However, as the demand for specialized hardware capabilities (e.g., GPUs or FPGAs for inference) increases, delivering self-contained robots equipped with all the necessary components is becoming increasingly challenging. The simplest and most adopted solution is to rely on dedicated edge devices with all the necessary components to perform offloading of (part of) the computation. In this regard, ROS2's reliance on Data Distribution Service (DDS) introduces significant performance degradation when networked communication is involved, often impacting even local intra-process exchanges under adverse network conditions. To address this limitation, we present Scalable Middleware 2 (SM2), a lightweight and deterministic middleware for intra- and inter-robot communication. SM2 separates local inter-process communication from network transport, ensuring that local performance remains predictable and unaffected by remote disruptions. Experimental results demonstrate that SM2 achieves comparable throughput to state-of-the-art alternatives such as Zenoh, while requiring significantly fewer system resources. Moreover, SM2 supports higher communication frequencies and lower latency, especially as the number of communicating ROS2 nodes increases. These results highlight SM2 as a scalable and resource-efficient middleware solution for reliable robotic applications.**

***Index Terms*—ROS2, Middleware, Robotics, Inter-Process Communication, Distributed Systems**

I. Introduction

Modern robotic systems rely on complex software architectures composed of numerous interdependent modules that must continuously exchange high-throughput data under real-time constraints. To manage this Inter-Process Communication (IPC), the robotics community has largely standardized on the Robot Operating System (ROS) [1], and more recently ROS2 [2], which offers a flexible, modular framework built on top of the Data Distribution Service (DDS) middleware. However, with the rising need for dedicated acceleration hardware (e.g., GPUs or FPGAs for inference tasks), developing self-contained robots capable of hosting all necessary hardware resources has become increasingly complex. Edge computing has been proposed as a viable solution to alleviate the burden of resource-constrained robots, enabling the offloading of (part of) the logic to dedicated servers equipped with all the required hardware [3], with the goal of optimizing performance or minimizing energy consumption [4]. As a result, the scope of modern robotic applications cannot be constrained within the boundaries of a single device, but rather involves multiple cooperating entities communicating through the network [5].

Although ROS2 introduces significant improvements over its predecessor, it exhibits critical performance limitations in its default DDS-based configuration, particularly when operating across unreliable networks [6], [7]. In particular, simply introducing a remote subscription (a subscriber node running on a different machine and communicating over the network), especially with poor connectivity, can cause severe performance degradation at the publisher, affecting not just remote communication, but also local subscribers that would otherwise operate unaffected. This tight coupling between network conditions and local node performance poses a serious reliability risk in real-world robotics applications where mixed network quality is common.

Recent efforts, such as the integration of Zenoh [8] as an alternative ROS2 middleware, attempt to address these challenges by centralizing communication through dedicated router nodes. While effective in decoupling local and remote communications, these solutions often incur significant CPU and memory overhead, making them less suitable for resource-constrained environments or high-frequency data pipelines. This becomes especially apparent when increasing the number of nodes taking part in communication [9].

To overcome these limitations, we introduce Scalable Middleware 2 (SM2), [1] a lightweight, deterministic software framework designed for fast, robust, and efficient local communication, even under adverse network conditions. Motivated by the need for a practical and scalable IPC solution for robotics, SM2 decouples local IPC from network communication to ensure predictable, uninterrupted module performance, while still supporting external connectivity through an optional, low-overhead gateway. It delivers the ease of use familiar from ROS2, but without the architectural complexity and performance overhead of DDS-based middleware.

The main contributions of this work are as follows. First, we introduce SM2, a lightweight IPC solution allowing ef-

[1]The number 2 does not indicate that this is the second version, but rather that it is positioned as an alternative to ROS2. https://github.com/andrea-fasano/sm2.

ficient intra-robot communication. Second, we present the SM2 Network Gateway, which decouples local communication from network communication and extends the architecture across machines, ensuring predictable performance even under unreliable conditions. Third, we provide an evaluation against ROS2 configurations, demonstrating that SM2 achieves similar throughput with significantly lower resource usage and latency.

II. State of the Art

A significant amount of research has been dedicated to evaluating the performance of ROS2, both in comparison with its predecessor [10], [11] and across different middleware options [7], [12]. ROS2 introduces a modular communication architecture centered on the concept of a ROS Middleware (RMW). The RMW serves as an abstraction layer between the ROS2 API and the underlying transport protocol, enabling support for multiple backends such as Fast DDS, Cyclone DDS, and Zenoh. Each implementation presents distinct trade-offs in latency, resource usage, and scalability.

Thulasiraman *et al.* [13] demonstrated that lossy network conditions can severely degrade the performance of DDS-based communication, particularly when security features are enabled. Similarly, Lee *et al.* [14] found that DDS-based transport struggles to handle large messages over wireless links, attributing the issue to fragmentation, retransmission timing, and buffer bursts. Maruyama *et al.* [10] further observed low throughput and process instability during large-message transmission, both in simulated and real environments.

Despite extensive evidence of performance degradation under non-ideal network conditions, there remains a substantial lack of research on how network communication impacts local performance and latency in controlled environments. This gap is particularly relevant for systems requiring deterministic or low-jitter communication, where DDS's design choices can introduce nontrivial overhead even in reliable networks.

Comparative studies of available middleware have shown that each distribution exhibits unique strengths and weaknesses. DDS-based implementations generally perform worse under adverse network conditions than their counterparts based on alternative protocols. Zhang *et al.* [7] showed that middleware based on Zenoh outperforms DDS-based options in terms of throughput in real robotic scenarios, particularly when relying on a centralized broker, although DDS still achieves lower latency in certain cases. Chovet et al. [6] reached similar conclusions under different real-life testing setups.

In terms of local computation and intra-process communication, Macenski *et al.* [9] reported significant overhead when increasing the number of separate ROS2 processes. Their proposed mitigation, using executors and intra-process communication instead of standard IPC, reduces wasted resources but is not always applicable or advisable across all development and deployment contexts.

Several studies have analyzed DDS-based communication in specialized domains. Paul *et al.* [15] evaluated vendor-specific ROS2 DDS implementations for cooperative driving, showing that latency and reliability depend heavily on data size, connection type, and bridging overhead. Puck et al. [16] demonstrated that with careful real-time tuning (e.g., real-time-safe memory allocation, thread prioritization, and CPU shielding), ROS2 can sustain deterministic communication with sub-millisecond latency, proving its feasibility for real-time control on standard hardware. Kouril *et al.* [17] conducted similar experiments for automated driving, reporting latencies below 2 ms and packet losses under 1%, with Cyclone DDS providing the most consistent performance, sufficient for reliable soft real-time operation but not for strict hard real-time guarantees.

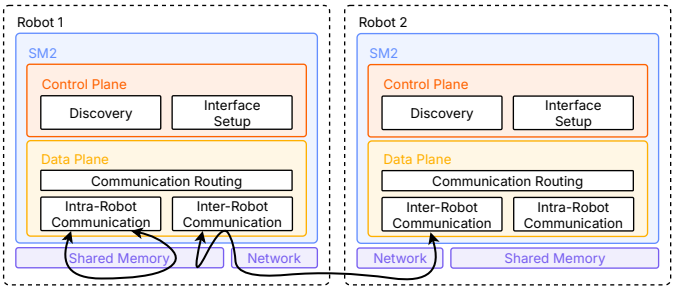

Fig. 1: SM2 logical view depicting the communication architecture and data exchange paths involved in intra-robot and inter-robot communication.

Beyond DDS itself, Time-Sensitive Networking (TSN) has been proposed as a complementary technology to ensure bounded delay and jitter under heavy load. Gutiérrez *et al.* [18] demonstrated TSN's potential for deterministic, low-latency robotic communication, suggesting its integration with DDS or OPC UA as a promising direction. Similarly, San Vicente Gutiérrez *et al.* [19] evaluated ROS2 on a real-time Linux kernel (PREEMPT-RT), finding that careful configuration (thread priorities, memory locking, etc.) can achieve bounded communication latency, though full real-time determinism remains elusive.

Overall, while numerous studies characterize ROS2 and its DDS-based middleware across different setups, there remains no comprehensive solution addressing the intrinsic latency variability and inefficiencies of DDS in both networked and local contexts. This gap highlights the need for alternative communication approaches or optimizations capable of delivering predictable, low-latency performance across heterogeneous and dynamic robotic environments.

III. SM2 Core

The highest priority is to guarantee reliable and deterministic local communication, regardless of external factors. The design of the solution mirrors this requirement by decoupling local and network communication. In the following, we first detail the control plane functionalities offered by SM2 for communication setup, moving then to the data plane functionalities to guarantee reliable communication (see Figure 1).

TABLE I: Comparison of communication frameworks for robotic systems.

Feature	ROS1	ROS2 (DDS)	ROS2 (Zenoh)	SM2
Node Discovery	Centralized (ROS Master)	Decentralized (Multicast)	Centralized Router or Peer-to-Peer	Centralized (Manager Node)
Intra-Robot Communication	TCP/UDP sockets	Shared Memory (optional) or UDP fallback	Shared Memory or TCP/UDP	Dedicated Shared Memory
Inter-Robot Communication	TCP/UDP sockets	DDS over UDP	TCP/UDP via Zenoh Routers	QUIC via Network Gateway
Scalability	Limited by ROS Master bottleneck	Moderate (discovery traffic overhead)	High (router-based)	High (Manager + Network Gateway)

A. Control Plane: Discovery and Interface Setup

The control plane handles discovery, registration, and interface setup within the SM2 domain. The SM2 Library is a C++ IPC library designed for robotics applications with strict real-time and reliability requirements. Similar to ROS, the library allows separate software modules to exchange data over a topic interface (publish/subscribe) or a RPC-like interface (service). At its core, SM2 builds on abstractions of system-level IPC primitives. Control messages are exchanged through the Socket abstraction, primarily derived from Unix Domain Sockets, while bulk data transfer is handled through the SharedMemory abstraction, built on POSIX shared memory.

In ROS1, node discovery is centralized: a process called ROS Master is responsible for discovery and for storing node-related information. However, if the ROS Master fails, it not only disables the discovery of new nodes but also undermines currently running connections. ROS2 overcomes such limitations by performing the discovery process using multicast traffic, which removes the need for a central ROS Master, preventing the single-point-of-failure scenario. The solution fully decouples components, allowing the system to continue working even in the case of one (or more) component failure. However, in modern robotics, a robot is no longer an isolated entity but part of a distributed computing environment. This shift strongly impacts scalability in multi-robot systems, which often rely on lossy networks where multicast delivery can be unreliable. To avoid the multicast delivery problem, in SM2 the registration and setup of communication are centralized, whereas data exchange and feedback occur in a peer-to-peer manner. The central authority overseeing communication setup is the *Manager* node, which is the only publicly accessible node within an SM2 local domain. All other nodes must register with the Manager and interact with it to create interfaces and establish peer-to-peer communication with their peers. A summary of these characteristics is provided in Table I.

This centralized approach has both advantages and disadvantages. The obvious critique to such a design is that the Manager is a single point of failure and, in case of many connected clients, it could become a bottleneck in the local domain. There are, however, multiple considerations that mitigate these potential problems:

- The operations that the Manager needs to conduct are fairly simple both computationally and logically, with appropriate data structures ensuring they can be completed in constant or logarithmic time.
- The Manager follows a strict policy to validate messages from clients, and a deviation from the protocol causes immediate disconnection of the client.
- Manager operations are restricted to changes in the local domain topology (e.g., creation or deletion of interfaces), meaning that the Manager is not involved in the data path for messages exchanged by the clients.
- Even in the case of Manager failure, the already established peer-to-peer connections between clients will keep operating, meaning that a previously set-up connection can continue to transmit data regardless of Manager status.

Having a centralized management of the local domain topology also brings a number of advantages. Since all state information about the local domain is gathered in one place, it becomes straightforward at any moment to determine its exact condition or query it if needed. The presence of a central Manager relieves individual clients of the burden of keeping track of changes, as the Manager assumes responsibility for informing them whenever updates occur. This arrangement not only simplifies the protocol but also avoids unnecessary communication between nodes: they no longer need to interact with peers they have no direct interest in, as all communication setup is routed through the Manager. As a result, the need for a distributed discovery system disappears, since nodes are not required to find each other autonomously. Furthermore, the centralized approach makes it possible to host multiple local domains on the same machine, simply by deciding which Manager the clients connect to.

B. Data Plane: Intra- and Inter-Robot Communication

The data plane is responsible for the actual exchange of data between processes, either within the same robot or across different robots. When two processes have established communication by creating the appropriate interfaces and interacting with the Manager, they communicate over sockets following a strict protocol that depends on the interface type. All control messages are exchanged over sockets and can potentially carry a reference to a shared memory buffer, allowing processes to share variable-size data in bulk. The shared memory buffers are created and written to by one process and then shared with one or multiple other processes. Sharing is a very lightweight operation, effectively implemented by leveraging operating system features (such as CMSG on Linux). The implementation of the buffers themselves also ensures that any unplanned behavior or process crash does not lead to dangling or leaking

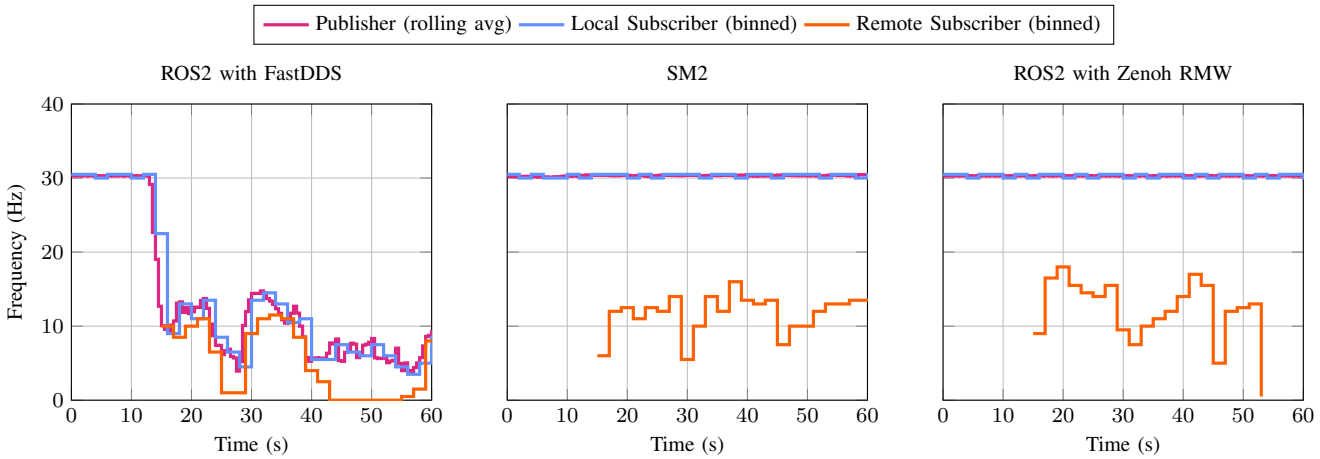

Fig. 2: The graphs show the frequency of sent and received messages in a setup with a publisher and a local subscriber, where a remote subscriber is introduced at time instant 15. Introducing a remote subscriber on an unreliable network using a DDS-based configuration causes a drop in performance, affecting both the local publisher and subscriber frequency. In contrast, SM2 maintains stable local communication performance when the remote subscriber is added. Zenoh RMW mitigates the performance drop observed in DDS-based ROS2 configurations and performs similarly to SM2 in network communication scenarios.

resources. After sharing, multiple processes have access to the same memory area. To prevent any erroneous or malicious operation on shared memory, the SM2 protocol requires that the process sharing the memory seal it beforehand. Sealing is an abstracted operation, based on Linux memory sealing, that prevents anyone from creating a write-access mapping of the memory, ensuring that no process can tamper with the data after it has been finalized.

Building on this abstraction, the SM2 *Network Gateway* (SNG) is a dedicated application that extends communication between SM2 local domains across multiple machines. Acting as a bridge, it can connect to one or more peers and selectively manage what data is exchanged and in which direction. To achieve this, the SNG provides fine-grained, configurable rule-sets that allow or deny the transmission of topics and services, with options for bidirectional, incoming-only, or outgoing-only flows. This prevents unnecessary traffic, conserves network bandwidth, and limits the exposure of sensitive data. The gateway emphasizes both flexibility and security. It automatically discovers peers on the same subnet via IPv6 multicast, supports authentication of peers, and ensures encrypted communication. QUIC was chosen as the transport protocol because it natively provides authentication, confidentiality, and parallel independent streams, thereby avoiding head-of-line blocking and ensuring robust performance under unreliable network conditions. Within each local domain, the SNG dynamically adapts to the current topology, its configuration, and peer requests, minimizing overhead and reducing the interaction burden on local processes. By integrating seamlessly with the SM2 Manager node, it reflects local topology changes and propagates them to remote peers when permitted. This ensures that SM2 retains its predictability and efficiency even in distributed, multi-robot deployments.

IV. Experimental Evaluation

To validate the effectiveness of the approach, and compare the behavior of the two main technologies available in ROS2, we used two laptops, we will call A and B. The two machines are running Ubuntu 20.04 and are connected using 2.4GHz Wi-Fi, which acts as an unreliable connection medium.

The Wi-Fi connection exhibited moderate reliability, with a measured bitrate of 30.8Mbit/s, jitter of 0.234ms, and a packet loss rate of 4%. While the average latency remained below 15ms, occasional spikes of up to 102ms and 1030ms were observed, indicating transient disruptions. These measurements were obtained using iPerf3.

All tested solutions were evaluated using their default configurations.

A. Network Communication Tests

The main driving force behind the development of SM2 was the all-around drop in performance when introducing network communication with ROS2 and DDS-based middleware. For this reason, SM2 was compared with ROS2 (Humble) with FastDDS in a test designed to highlight the impact of the condition of the network on local communication. The test was also conducted with the Zenoh RMW middleware for ROS2 to benchmark SM2 against a state-of-the-art ROS2 configuration, and to gather comparative data.

The network tests are organized as follows:

1) A publisher node running on laptop A publishes large messages (1MB each) at a fixed frequency (30Hz).
2) A subscriber node is also started on A, and the frequency and latency of communication is measured, providing a baseline.

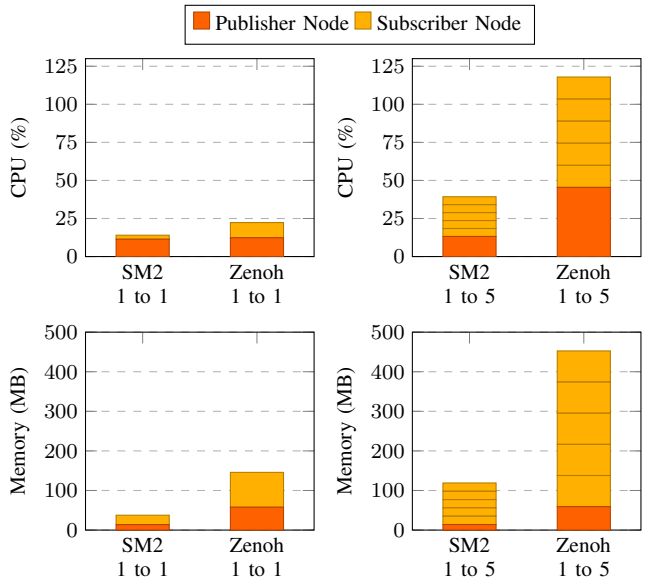

Fig. 3: SM2 and ROS2 + Zenoh compared in terms of usage of CPU and memory. In CPU utilization 100% is one core. In this test, a large message (10MB) would be published at a fixed frequency (10Hz), and one or more processes would subscribe to it. Memory usage and CPU times were recorded from system statistics.

3) Another subscriber is started on B (the machine is connected over Wi-Fi) subscribing to the same topic, after approximately 15s from the beginning of the test.
4) Metrics from all participants in the communication are recorded for later analysis.

Figure 2 shows the difference between ROS2 with DDS-based middleware and the behavior of SM2 and Zenoh, which obtain very similar results. Neither SM2 nor Zenoh show any impact of remote communication on the local side and achieve comparable throughput. FastDDS expectedly suffers from a major performance penalty at the publisher, affecting all participants in the communication, both local and remote. It is interesting to note how even considering only message throughput in network communication, FastDDS manages to deliver far fewer messages, with protracted periods where no messages are delivered at all. The experiment was carried out with both reliable and best-effort QoS settings, which yielded completely analogous results.

Given the glaring issue introduced in DDS-based communication, for the following tests, we decided to limit the following tests only on SM2 and Zenoh to find out how they compare on other metrics.

B. Resource Consumption

Although SM2 and ROS2 with Zenoh RMW behave similarly in the tests designed to highlight the correlation between network communication and drops in local performance, the latter takes a greater toll on system resources. The tests were run locally, using only laptop A. As evidenced by Figure 3,

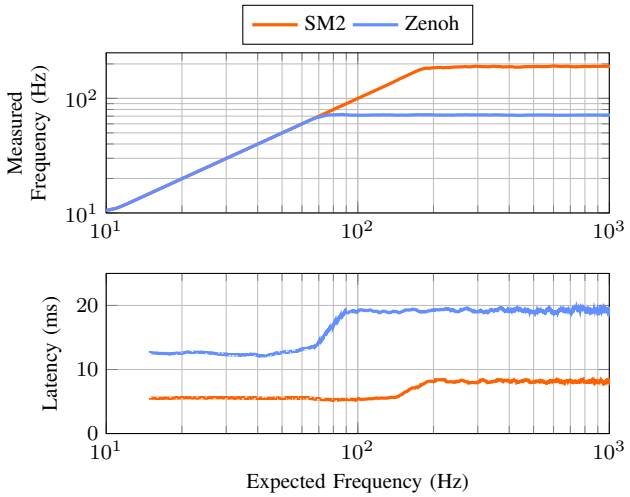

Fig. 4: Latency and achieved frequency at different target publication frequencies, in 1 to 5 local communication. SM2 and ROS2 + Zenoh compared. Message size is 10MB.

when running a publisher (10MB messages) at a frequency of 10Hz, ROS2 with Zenoh RMW uses up to four times as much memory and three times as much CPU time as SM2. This disparity is especially pronounced when the number of subscribers increases (i.e., moving from "1 to 1", one publisher and one subscriber, to "1 to 5", one publisher and 5 subscribers).

This difference in resource usage is significant, as it directly translates to the need for more powerful hardware to deploy similar capabilities. However, even under conditions that should not saturate system resources, it still leads to important differences in performance as we will show in the following subsection.

C. Latency

Starting from the previously described configuration, we were able to understand how Zenoh and SM2 compare in terms of latency. Figure 4 shows the result of a test in which message latency was measured at increasing publication frequencies. This test highlights how SM2 is capable of operating at significantly higher frequencies (with a peak frequency around 200Hz) under the same conditions, with Zenoh only able to manage a peak frequency of around 70Hz. SM2 is also capable of consistently delivering messages at lower latency, with both solutions experiencing an expected latency increase at saturation.

All these performance differences are exacerbated when increasing the number of publishers or subscribers taking part in the communication. Figure 5 shows how the average latency of the messages delivered grows significantly for each added subscriber, and, as for the previous results, SM2 clearly outperforms the Zenoh implementation.

V. Conclusion

This work presented SM2, a lightweight middleware designed to ensure reliable and deterministic inter-process com-

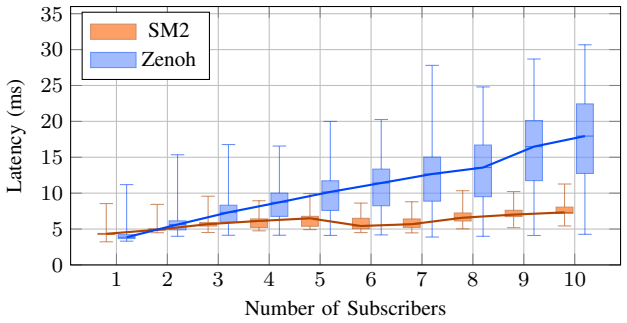

Fig. 5: Comparison of latency with different subscriber counts between SM2 and ROS2 with Zenoh.

munication in robotic systems, even in the presence of adverse or unreliable network conditions. By decoupling local IPC from network transport, SM2 prevents the performance degradation commonly observed in DDS-based ROS2 configurations, where remote subscriptions can negatively impact local communication.

The architecture of SM2 combines a centralized management model with efficient abstractions over system-level IPC primitives, enabling predictable communication while minimizing resource usage. Furthermore, the introduction of the SM2 Network Gateway extends this reliability to inter-machine communication, offering secure, configurable, and low-overhead connectivity through the QUIC protocol.

Experimental evaluation demonstrated that SM2 achieves comparable throughput to state-of-the-art alternatives such as ROS2 with Zenoh RMW, while consuming significantly fewer CPU and memory resources. SM2 reduced CPU by 3× and memory by 4× compared to Zenoh RMW. Additionally, SM2 supports higher message throughput and lower message latencies, particularly as the number of participating nodes increases. These results highlight SM2 as a practical and scalable middleware for robotics, offering usability comparable to ROS2 while delivering predictable performance and reduced overhead.

Future work on SM2 includes expanding platform support and improving serialization compatibility across architectures. The Network Gateway could benefit from enhanced parallelism and per-connection interface handling to better isolate faulty peers.

Acknowledgment

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.

SM2 was developed for the DRAFT team at Politecnico di Torino to participate in the Leonardo Drone Contest.

References

[1] M. Quigley, B. Gerkey, K. Conley, J. Faust, T. Foote, J. Leibs, E. Berger, R. Wheeler, and A. Ng, "Ros: an open-source robot operating system," in *IEEE International Conference on Robotics and Automation*, 2009. [Online]. Available: https://api.semanticscholar.org/CorpusID:6324125

[2] 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, p. eabm6074, 2022.

[3] S. Galantino, J. Marino, F. Risso, S. Braghin, L. Nedoshivina, A. F. Skármeta Gómez, and D. Siracusa, "Edge-to-cloud continuum made real," *Computer*, vol. 59, no. 3, pp. 51–59, 2026.

[4] S. Galantino, F. Risso, V. C. Coroamă, and A. Manzalini, "Assessing the potential energy savings of a fluidified infrastructure," *Computer*, vol. 56, no. 6, pp. 26–34, 2023.

[5] D. Cacciabue, J. Marino, F. Aglieco, M. Levorato, D. Perroni, and F. Risso, "Benchmarking different strategies for offloading ros2 computation to the edge," in *2024 IEEE 10th International Conference on Network Softwarization (NetSoft)*, 2024, pp. 49–54.

[6] L. Chovet, G. Garcia, A. Bera, A. Richard, K. Yoshida, and M. Olivares-Mendez, "Performance comparison of ros2 middlewares for multi-robot mesh networks in planetary exploration," 07 2024.

[7] J. Zhang, X. Yu, S. Ha, J. P. Queralta, and T. Westerlund, "Comparison of middlewares in edge-to-edge and edge-to-cloud communication for distributed ros 2 systems," *Journal of Intelligent & Robotic Systems*, vol. 110, no. 4, p. 162, Nov. 2024. [Online]. Available: https://doi.org/10.1007/s10846-024-02187-z

[8] A. Corsaro, L. Cominardi, O. Hecart, G. Baldoni, J. Avital, J. Loudet, C. Guimares, M. Ilyin, and D. Bannov, "Zenoh: Unifying communication, storage and computation from the cloud to the microcontroller," in *2023 26th Euromicro Conference on Digital System Design (DSD)*. Los Alamitos, CA, USA: IEEE Computer Society, Sep 2023, pp. 422–428. [Online]. Available: https://doi.ieeecomputersociety.org/10.1109/DSD60849.2023.00065

[9] S. Macenski, A. Soragna, M. Carroll, and Z. Ge, "Impact of ros 2 node composition in robotic systems," *IEEE Robotics and Automation Letters*, vol. 8, pp. 3996–4003, 2023. [Online]. Available: https://api.semanticscholar.org/CorpusID:258740945

[10] Y. Maruyama, S. Kato, and T. Azumi, "Exploring the performance of ros2," in *2016 International Conference on Embedded Software (EMSOFT)*, 2016, pp. 1–10.

[11] J. Park, R. Delgado, and B.-W. Choi, "Real-time characteristics of ros 2.0 in multiagent robot systems: An empirical study," *IEEE Access*, vol. 8, pp. 154 637–154 651, 2020. [Online]. Available: https://api.semanticscholar.org/CorpusID:221473691

[12] L. J. Dust, E. Persson, M. Ekström, S. Mubeen, and E. Dean, "Quantitative analysis of communication handling for centralized multi-agent robot systems using ros2," *2022 IEEE 20th International Conference on Industrial Informatics (INDIN)*, pp. 624–629, 2022. [Online]. Available: https://api.semanticscholar.org/CorpusID:254736981

[13] P. Thulasiraman, Z. Chen, B. D. Allen, and B. Bingham, "Evaluation of the robot operating system 2 in lossy unmanned networks," *2020 IEEE International Systems Conference (SysCon)*, pp. 1–8, 2020. [Online]. Available: https://api.semanticscholar.org/CorpusID:228090318

[14] S. Lee, T. Kim, J. Chae, and K.-J. Park, "Optimizing ros 2 communication for wireless robotic systems," *ArXiv*, vol. abs/2508.11366, 2025. [Online]. Available: https://api.semanticscholar.org/CorpusID:280671328

[15] S. Paul, D. Lephuoc, and M. Hauswirth, "Performance evaluation of ros2-dds middleware implementations facilitating cooperative driving in autonomous vehicle," 2024. [Online]. Available: https://arxiv.org/abs/2412.07485

[16] L. Puck, P. Keller, T. Schnell, C. Plasberg, A. Tanev, G. Heppner, A. Roennau, and R. Dillmann, "Performance evaluation of real-time ros2 robotic control in a time-synchronized distributed network," in *2021 IEEE 17th International Conference on Automation Science and Engineering (CASE)*, 2021, pp. 1670–1676.

[17] J. Kouril, B. Schäufele, I. Radusch, and B. Schnor, "Performance evaluation of a ros2 based automated driving system," 11 2024.

[18] C. S. V. Gutiérrez, L. U. S. Juan, I. Z. Ugarte, and V. M. Vilches, "Time-sensitive networking for robotics," 2018. [Online]. Available: https://arxiv.org/abs/1804.07643

[19] C. Gutiérrez, L. Juan, I. Ugarte, and V. Vilches, "Towards a distributed and real-time framework for robots: Evaluation of ros 2.0 communications for real-time robotic applications," 09 2018.