Comparison of DDS, MQTT, and Zenoh in Edge-to-Edge/Cloud Communication with ROS 2

Jiaqiang Zhang[†], Xianjia Yu[†], Sier Ha[†], Jorge Peña Queralta[†], Tomi Westerlund[†]

[†]Turku Intelligent Embedded and Robotic Systems (TIERS) Lab, University of Turku, Finland.
Emails: [1]{jiaqiang.zhang, xianjia.yu, sierha, jopequ, tovewe}@utu.fi

***Abstract*—With the development of IoT and edge computing, there is a need for efficient and reliable middleware to handle the communication among Edge devices or between Edge and Cloud. Meanwhile, ROS 2 is more commonly used in robotic systems, but there is no comparison study of middleware using ROS Messages. In this study, we compared the middlewares that are commonly used in ROS 2 systems, including DDS, Zenoh, and MQTT. In order to evaluate the performance of the middleware in Edge-to-Edge and Edge-to-Cloud scenarios, we conducted the experiments in a multi-host environment and compared the latency and throughput of the middlewares with different types and sizes of ROS Messages in three network setups including Ethernet, Wi-Fi, and 4G. Additionally, we implemented different middlewares on a real robot platform, TurtleBot 4, and sent commands from a host to the robot to run a square-shaped trajectory. With the Optitrack Motion Capture system, we recorded the robot trajectories and analyzed the drift error. The results showed that CycloneDDS performs better under Ethernet, and Zenoh performs better under Wifi and 4G. In the actual robot test, Zenoh's trajectory drift error was the smallest.**

***Index Terms*—Cloud computing; Edge computing; ROS 2; ROS Middleware; DDS; MQTT; Zenoh;**

I. Introduction

The rise of the Internet of Things (IoT) and edge computing has led to the need for efficient and reliable communication protocols that can handle the massive amounts of data generated by distributed systems. In addition, these systems often include heterogeneous devices and communication protocols, which can make data sharing and communication a challenging task. At the same time, the Robot Operating System (ROS) has become the *de-facto* standard in autonomous robotic systems, and recently its revamped version, ROS 2. ROS 2 changed the middleware allowing robots to better exploit the computational resources of the cloud-edge continuum [1].

The selection of a middleware solution for ROS nodes can have a significant impact on the overall performance and scalability of the system. In this study, we evaluate and compare three middlewares in edge-to-edge and edge-to-cloud communication scenarios. The middlewares are Data Distribution Service (DDS), Message Queuing Telemetry Transport (MQTT), and Zenoh. DDS [2] is a data-centric middleware solution that emphasizes efficient and reliable data transfer between devices. DDS provides a rich set of Quality of Service (QoS) policies that allow applications to specify the reliability, latency, and throughput requirements of their data. MQTT [3] is a lightweight messaging protocol that uses a publish/subscribe model for data transfer. Zenoh [4] is an open-source protocol and suite of tools for data sharing and communication in distributed systems. Zenoh aims to provide a unified approach to data sharing and communication, regardless of the underlying hardware, network topology, or programming languages used.

In this paper, we evaluate the performance of these middlewares in terms of latency and throughput under different network conditions, including Ethernet, Wi-Fi, and 4G. The experiment setup is shown in Fig. 1. Additionally, we compare the performance of middleware on an actual robot platform, as shown in Fig. 4. To the best of our knowledge, extant literature lacks a comprehensive examination of middleware performance in conjunction with ROS Messages. Furthermore, the research landscape has not yet delved into the intricacies of assessing middleware performance within the context of tangible, real-world robotic systems. The present study, therefore, serves to bridge this evident gap in the scholarly discourse by conducting a meticulous comparative analysis of middleware functionalities utilizing ROS Messages and actualized experimentation on physical robotic platforms.

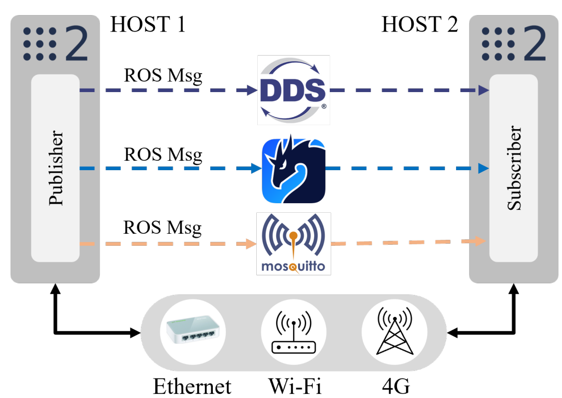


Fig. 1: Experiment setup.

The rest of the paper is organized as follows: Section II provides an overview of the background and related work, discussing the key concepts of the middleware used in ROS systems and presenting the existing literature on the performance study of middleware used in robotics and IoT. Section

III describes the experimental setup and methodology used to evaluate the performance of these middlewares. Section IV presents the experiment results of DDS, MQTT, and Zenoh, including latency and throughput. We analyze the findings as well. Section V discusses CPU usage and the impact of the security feature on performance. Finally, Section VI concludes the paper.

II. Background and Related Work

Through this section, we give a brief introduction to ROS 2 and the most used communication protocols, as well as the existing study on the performance of middleware.

A. ROS 2 and SROS 2

Robot Operating System (ROS) has emerged as a popular middleware for developing robotic systems due to its modular architecture, open-source nature, and large community support. ROS provides a framework for communication between various modules of a robotic system using message passing and service calls. ROS 2, the latest version of ROS, was released in 2015, with an emphasis on real-time systems, large-scale deployments, and distributed systems. ROS 2 has improved modularity and supports different middleware for communication.

SROS 2, short for Secure ROS 2, is a security framework designed to enhance the data protection and communication security of robotic systems built on ROS 2. SROS 2 provides a set of tools, and libraries for providing security, authentication, and access control for ROS 2 entities [5]. In [6], the authors investigate the trade-off between security and performance in ROS 2 middleware. It provides guidelines for selecting appropriate security mechanisms and highlights the importance of a "defense-in-depth" approach. The authors conduct experiments to measure the impact of different security mechanisms on communication latency and throughput.

B. Middleware

ROS Middleware (RMW) is the abstraction layer responsible for facilitating communication between different components of a robotic system [7]. It provides a set of APIs and protocols for sending and receiving data between publishers and subscribers. The use of RMW allows for decoupling of the communication protocol from the application code, enabling interoperability between different components and platforms. The most used middleware in ROS 2 systems are DDS, MQTT, and Zenoh.

1) DDS: DDS is a publish-subscribe middleware that provides a data-centric communication model for distributed systems. It is designed for real-time systems and supports QoS settings to optimize communication performance. As the default RMW of ROS 2, DDS provides a decentralized architecture shown in Fig. 2, where nodes can communicate directly with each other without relying on a centralized broker. This architecture ensures reliability, fault tolerance, and scalability.

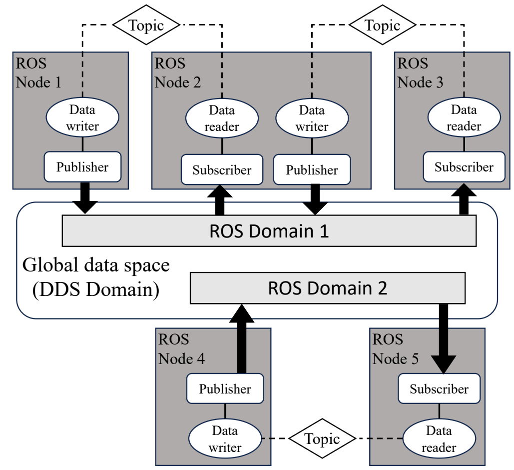


Fig. 2: DDS architecture in ROS 2 system

2) MQTT: MQTT is a lightweight publish-subscribe protocol designed for constrained devices and low-bandwidth network environments and has almost become a *de-facto* standard in the field of Internet of Things (IoT) [8, 9]. The architecture of MQTT is shown in Fig. 3. It supports QoS levels to ensure message delivery and allows for a wide range of applications, including sensors and IoT devices.

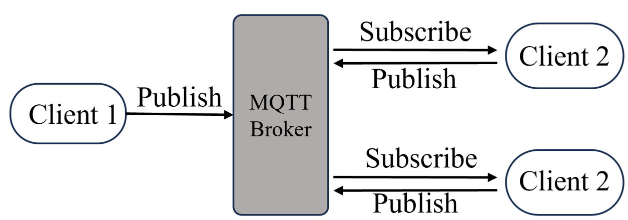


Fig. 3: MQTT architecture

3) Zenoh: Zenoh is a new type of middleware that provides a unified data space for distributed systems. It is designed to provide seamless communication between edge devices, cloud systems, and hybrid environments. Zenoh provides a data-centric communication model and supports different data formats, including structured and unstructured data. It also provides a decentralized architecture and supports fault tolerance and scalability.

DDS, MQTT, and Zenoh are three different middleware options with distinct features. DDS is a data-centric middleware that offers a decentralized architecture, enabling direct peer-to-peer communication and extensive QoS settings for optimizing reliability, latency, and throughput. MQTT, on the other hand, is a lightweight publish-subscribe protocol designed for resource-constrained devices and networks, with a client-server architecture and low network overhead. It is suitable for IoT applications. Zenoh provides a unified data space, allowing seamless communication across edge devices, cloud systems, and hybrid environments.

C. Performance Tools

In this part, we present the tools or packages that are developed to assess the performance of the ROS 2 system.

- The performance_test [10] tool tests latency and other performance metrics of various middleware implementations that support a pub/sub pattern. It is used to simulate the non-functional performance of the application.
- The ros2_tracing [11] is a tool that provides tracing capabilities within ROS 2. It enables the collection of detailed runtime data, including message passing, data flow, and timing characteristics.
- The ros-network-analysis [12] is a ROS package that provides tools to analyze the wireless network such as the signal quality, latency, throughput, link utilization, connection rates, error metrics, etc., between two ROS nodes, computers, or machines.
- CARET (Chain-Aware ROS Evaluation Tool) [13] provides a customizable and analyzable framework for capturing and analyzing real-time execution traces in ROS-based systems. It allows developers to gain a deep understanding of the timing behavior and performance characteristics of their real-time applications, facilitating the development and optimization of reliable and efficient robotic systems.
- The ros2_latency_evaluation [14] provides tools and scripts for measuring and analyzing latency in ROS 2 communication between multiple nodes. It includes code that sets up the necessary infrastructure to perform latency measurements, collects latency data during the execution of ROS 2 systems, and provides analysis scripts to process and visualize the collected data.
- The Autoware_Perf [15] can measure the callback latency, node latency, and communication latency in ROS 2 applications. In addition, Autoware_Perf calculates the end-to-end latency by a convolutional integral as an estimated value.

D. Related Work on the Performance of Middleware

Several studies have investigated the performance and characteristics of different middleware solutions in the context of robotic systems and communication frameworks. These studies shed light on various aspects such as latency, throughput, queuing systems, dataflow architectures, and message analysis in distributed environments.

[16] provides a qualitative and conceptual overview of IoT protocols and service-oriented middleware, offering insights into their roles, capabilities, and challenges in the Internet of Things domain. Examining a decentralized serverless architecture, [17] addresses the challenges of dataflow in the cloud-to-edge continuum. The research presents an architectural solution that can provide insights into optimizing communication and data processing in the context of distributed systems. [18] focuses on message flow analysis in distributed ROS 2 systems, addressing complex causal relationships. The paper's insights into message propagation and interactions contribute to understanding the behavior of ROS Middleware in complex scenarios. L. Puck et al. investigated the real-time performance of ROS 2-based robotic control within a time-synchronized distributed network [19]. In [20], the authors use a standardized metric and reproducible experimental environment to provide a fair comparison among five message queuing systems, including Kafka, RabbitMQ, RocketMQ, ActiveMQ, and Pulsar. The latency characteristics of multi-node systems within ROS 2 are examined in [14]. It highlights the criticality of low latency in robotics and presents an in-depth analysis of factors affecting latency in ROS 2. The work sheds light on the significance of optimizing communication mechanisms for real-time applications. A comprehensive performance study on Zenoh, MQTT, Kafka, and DDS is provided in [21]. By evaluating throughput and latency, the study contributes to understanding the trade-offs and capabilities of different middleware implementations, thus aiding in informed middleware selection.

III. Experiment Methodology

In this section, we outline the experimental methodology employed to evaluate and compare the performance of DDS, MQTT, and Zenoh in Edge-to-Edge and Edge-to-Cloud communication scenarios. We describe the experimental setup, test scenarios, the equipment utilized for testing, and performance metrics used for comparison.

A. Experimental Setup

Our experiments were conducted in a networked environment consisting of multiple robotic systems. As shown in Fig.1, the setup includes a combination of devices and communication channels, including Ethernet, Wi-Fi, and 4G connectivity. Each middleware (DDS, MQTT, and Zenoh) was configured and integrated into the ROS 2 framework, enabling communication between the robotic systems. Notably, the RMW is DDS in all experiments. In the experiments of Zenoh and MQTT, we enabled "Localhost Only" and set bridge between DDS and Zenoh or MQTT, so that DDS will publish or subscribe ROS Messages only to the local host. In other words, for Zenoh or MQTT, the communication within one host used DDS, and communication between two hosts used Zenoh or MQTT.

1) Hardware: The experiments were conducted in multi-machine scenarios. All devices involved in the experiments run Ubuntu 20.04. The device information is shown in Table I.

TABLE I: Hardware Setup

Devices	System	PROCESSOR	Memory
Host 1	Ubuntu 20.04	AMD® Ryzen 7 5800h	16GB
Host 2	Ubuntu 20.04	Intel® Core i3-1215U	16GB
Router/Broker	Ubuntu 20.04	Intel® Core i7-9700K	64GB

In order to evaluate the performance of different middleware implementations in a real-world scenario, we set the middlewares between a TurtleBot 4 robot and a laptop. As shown Fig. 4, the TurtleBot 4 and the laptop both connect to the same Wi-Fi local network. The laptop sends commands to the TurtleBot 4 to execute a square-shaped trajectory. The TurtleBot 4 will run four square circles in 96 seconds. The OptiTrack motion capture (MOCAP) system records the actual trajectory of the TurtleBot 4 during the execution of the square shape.

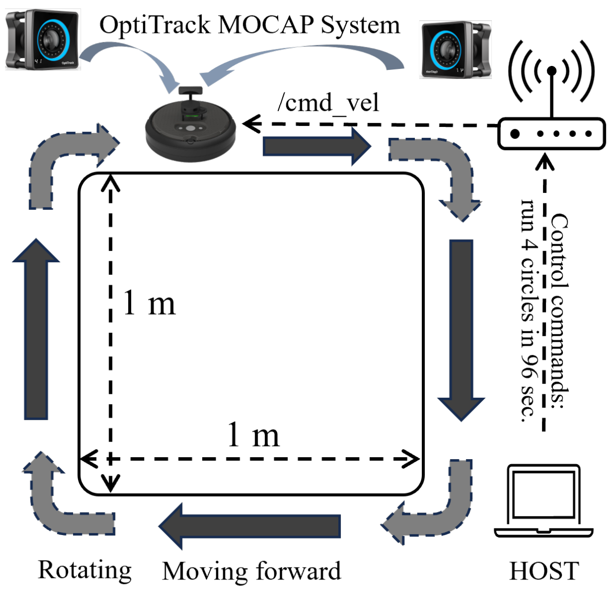


Fig. 4: Real-world setup

2) Network: In order to assess the performance of the middlewares in different network conditions, we set three different network environments, as shown in Fig. 5. The first setup utilized an Ethernet local network, representing a high-bandwidth and low-latency environment for Edge-to-Edge communication. The LAN Switch used in Ethernet Local Network is Unifi Flex Mini. The second setup involved a local network using Wi-Fi connectivity, simulating Edge-to-Edge communication within a confined area. The Wi-Fi Router is Huawei B593. Lastly, we leveraged the Zerotier virtualized network to 4G setup, which emulates Edge-to-Cloud communication scenarios. In this scenario, for HOST 1 & 2, we used HUAWEI 4G Dongle E3372 as 4G module, and for broker, we used Netgear 4G LTE Mobile Router as 4G module.

3) ROS Message: To comprehensively evaluate the performance of the middlewares, we employed various types and sizes of ROS Messages in our experiments, including Array1k, Array4k, Array16k, Array64k, Array256k, Array1m, Array2m, PointCloud512k, PointCloud1m, and PointCloud2m, where “k” and “m” are “Kilobyte” and “Megabyte”. The ROS messages were published and subscribed with a frequency of 10 Hz. By using different sizes and types of ROS Messages, we aimed to assess the middleware’s ability to handle messages of varying complexities and data sizes, simulating real-world scenarios encountered in robotic systems.

By designing our experimental setup to encompass different hardware configurations, network conditions, and message types, we ensured a comprehensive evaluation of the performance of DDS, MQTT, and Zenoh in Edge-to-Edge and Edge-to-Cloud communication scenarios. The chosen setups allowed us to capture the nuances and implications of each middleware’s performance under various conditions, facilitating a robust comparison and analysis of their capabilities.

B. *Performance Metrics and Tool*

We focus on two key performance metrics: latency and throughput. The performance tool selected for this research is performance_test [10].

The primary objective of this work is to compare the performance of different middlewares in Edge-to-Edge and Edge-to-Cloud environments. The performance_test tool is specifically designed for benchmarking and performance evaluation, making it well-suited for this purpose. It provides standardized methods for generating synthetic message traffic, measuring latency and throughput, and facilitating direct performance comparisons between different middlewares. In addition, the performance_test supports SROS 2 with a simple setup.

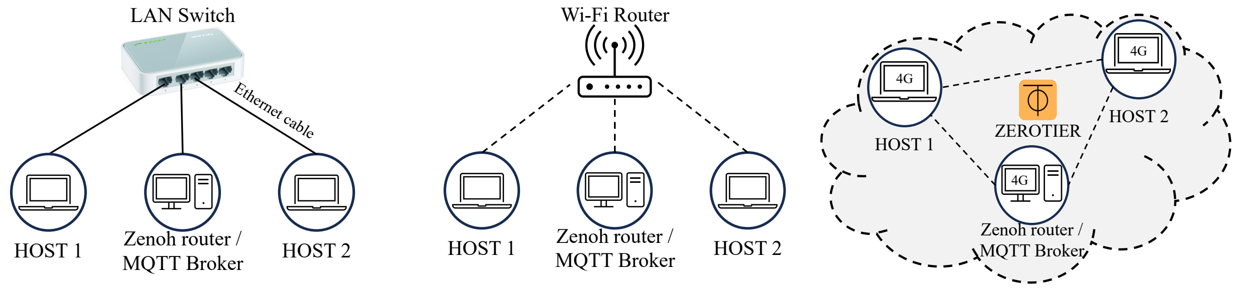
IV. Experiment Results

In this section, we present the results of our experiments comparing the performance of DDS, MQTT, and Zenoh in different network conditions and on an actual robot platform.

A. *Latency*

Table II presents the mean latency results obtained from the experiments conducted in different network setups, including Ethernet, Wi-Fi, and 4G. In each network setup, we measured the mean latency for different middleware implementations and various ROS message types. The results provide insights into the latency performance of each middleware under different network conditions and message sizes.

It is evident that the mean latency increases as the message size grows in all network scenarios. The results also demonstrate the impact of different network setups on the latency performance of middleware. Each middleware performs better under Ethernet, which has the highest bandwidth, than Wi-Fi and 4G conditions. Notably, CycloneDDS performs best in Ethernet setup, while Zenoh performs best in Wi-Fi and 4G setups. This is caused by the DDS discovery mechanism. DDS leverages UDP multicast features to broadcast messages in the transport layer. It enables fast peer discovery and QoS-based low-latency data transmission in wired local networks where bandwidth and packet loss rate are promised. On the other hand, in wireless networks such as Wi-Fi or 4G, UDP multicast could cause the flooding effect, resulting in poorer performance [21, 22]. These findings highlight the importance of considering network conditions and message characteristics when selecting an appropriate middleware for Edge-to-Edge and Edge-to-Cloud communication in ROS-based robotic systems.



(a) Ethernet Local Network (b) Wi-Fi Local Network (c) 4G (Zerotier Virtualized Network)

Fig. 5: Network setups. The router or broker is for Zenoh and MQTT; DDS does not need an individual broker.

TABLE II: Mean latency with different network setup

(a) Mean latency (ms) with Ethernet

ROS Message	FastDDS	CycloneDDS	Zenoh-TCP	Zenoh-broker	MQTT-nobroker	MQTT-broker
Array1k	13.63	**1.29**	1.98	1.78	89.74	91.01
Array4k	16.92	**1.30**	1.97	1.91	86.59	3.87
Array16k	26.62	**1.55**	2.36	2.24	86.98	3.58
Array64k	4.79	3.35	3.35	**3.27**	3.51	14.77
Array256k	43.63	**5.37**	7.42	7.42	10.75	48.50
Array1m	180.09	**19.28**	32.88	42.69	136.60	5545.04
Array2m	-	**37.97**	70.97	70.61	312.89	7468.62
PointCloud512k	66.55	**11.16**	20.51	18.47	18.45	87.07
PointCloud1m	545.27	**21.95**	37.42	41.24	94.92	1020.70
PointCloud2m	604.80	**39.94**	76.78	77.12	540.25	5625.40

(b) Mean latency with Wi-Fi

ROS Message	FastDDS	CycloneDDS	Zenoh-TCP	Zenoh-broker	MQTT-nobroker	MQTT-broker
Array1k	178.40	51.40	14.82	**12.58**	92.93	173.62
Array4k	178.74	39.84	**32.34**	54.11	101.19	73.64
Array16k	2811.88	56.73	**48.30**	139.63	101.77	125.69
Array64k	3238.00	606.50	452.51	326.96	**278.81**	4382.25
Array256k	12450.00	1574.14	3021.77	2323.91	13538.75	19788.57
Array1m	-	-	**5960.16**	7599.04	20950.00	-
Array2m	-	-	**9386.79**	11337.42	-	-
PointCloud512k	-	-	**5164.90**	5407.28	11307.95	-
PointCloud1m	-	**3748.00**	5326.69	5657.96	24115.00	-
PointCloud2m	-	-	9283.26	**8805.96**	-	-

(c) Mean latency with 4G

ROS Message	FastDDS	CycloneDDS	Zenoh-TCP	Zenoh-broker	MQTT-nobroker	MQTT-broker
Array1k	104.69	**58.80**	72.11	134.02	279.17	679.91
Array4k	126.50	103.75	**102.86**	98.05	228.84	537.87
Array16k	3680.47	**116.95**	150.81	145.42	220.82	530.20
Array64k	4802.26	**163.29**	3349.90	434.19	541.76	2222.11
Array256k	-	-	**7458.61**	7556.25	10736.73	19987.27
Array1m	-	-	**10086.88**	11811.52	-	-
Array2m	-	-	15365.29	**10870.59**	-	-
PointCloud512k	-	-	7357.50	**6591.85**	-	-
PointCloud1m	-	-	13254.00	**10333.13**	-	-
PointCloud2m	-	-	14309.50	**11900.27**	-	-

B. Throughput

Fig. 6 presents the throughput results.The results showed that the bandwidth of the network has the largest impact on throughput. Additionally, as explained in IV-A, CycloneDDS performed as well as Zenoh in Ethernet setup, but in Wi-Fi and 4G scenarios, Zenoh showed better performance.

C. Performance on Actual Robot Platform

Fig. 7 shows the moving trajectory of the TurtleBot 4 over 96 s. Fig. 8 shows the drift error between the last and the first circle. Zenoh demonstrated the closest adherence to the square trajectory, exhibiting minimal drift and accurately reproducing the desired path. This result highlights Zenoh's robustness and capability to maintain precise position and movement control on a real robot platform. On the other hand, the trajectories of CycloneDDS, FastDDS, and MQTT showed varying degrees of drift over time.

V. Discussion

Throughout the paper, we have provided a comprehensive comparison of DDS, MQTT, and Zenoh by giving the experiment results and their analysis. The evaluation of latency and

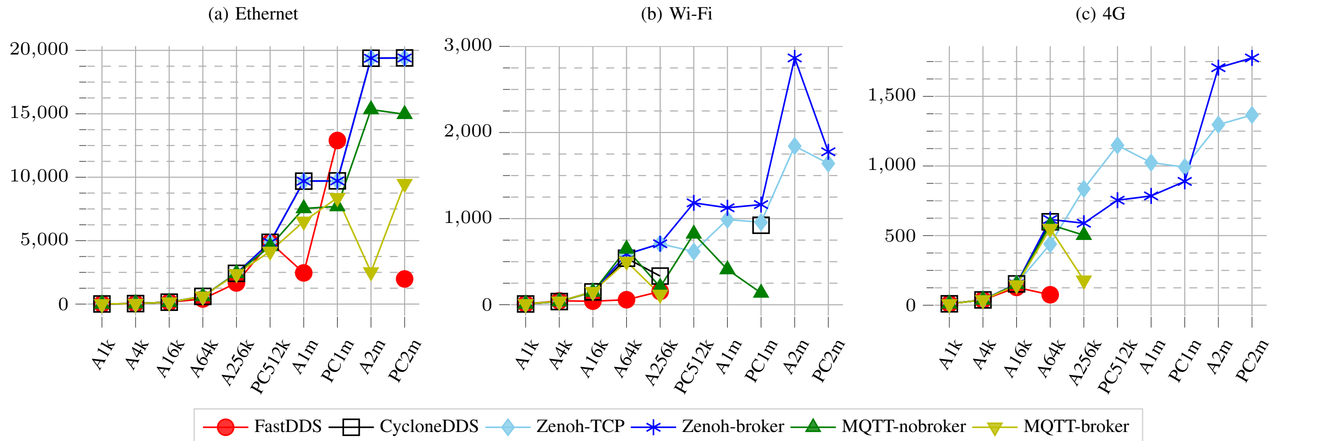


Fig. 6: Throughput (Kilobytes/sec) with different network setup. In Subfig (a), since the data difference is very small, the lines of CycloneDDS, Zenoh-TCP, and Zenoh-broker overlap.

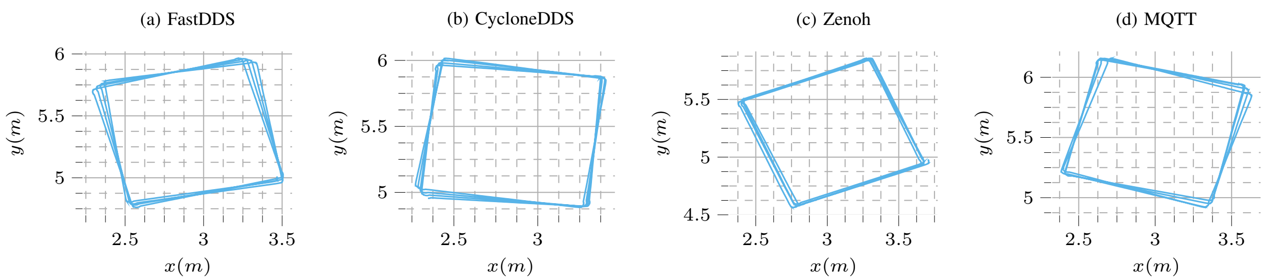


Fig. 7: Robot Real-world Moving Trajectory over 96 s of Different Middleware.

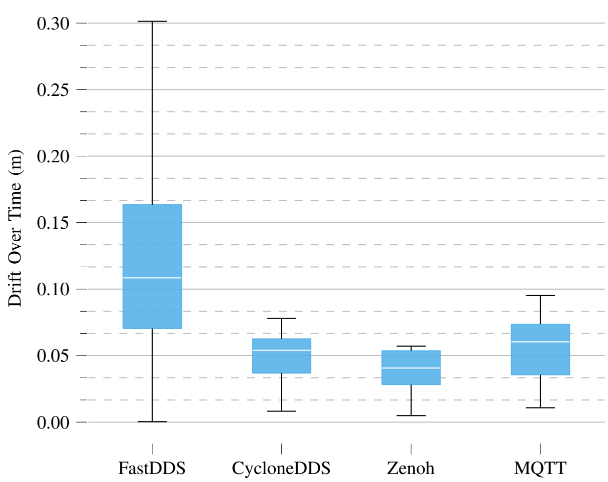


Fig. 8: Robot Real-world Moving Drift Over 96 s of Different Middlewares.

throughput within various network setups, including Ethernet, Wi-Fi, and 4G, yielded significant insights into the performance of different middlewares. Besides these insights, the ensuing subsections delve into two other dimensions worth noting—namely, “CPU Usage” and “Security”—to provide a multi-angle understanding of the middleware.

A. CPU usage

The CPU usage of Host 1 as shown in Table III, acquired through the “performance_test” tool, provides valuable information about the computational efficiency of each middleware. The results indicate that FastDDS exhibits the highest CPU usage with an average of 6%. In contrast, the other middlewares showcase significantly lower CPU usage, all remaining below 1%.

B. Security

Enabling ROS 2 security affects the performance metrics such as latency and throughput of the ROS 2 communication channel [6]. The use of Array1k messages to test latency, throughput, and CPU usage underlines the effects of enabling security features. As shown in Table IV, the results reveal that activating security measures causes a noticeable increase in mean latency, indicating the additional processing overhead introduced by security mechanisms. Furthermore, there is a decrease in throughput, which aligns with expectations due to the encryption and decryption processes associated with securing message communication. And CPU usage remains relatively unaffected by the security measures.

TABLE IV: Security difference

Security	Mean latency	Throughput	CPU usage
With security	77.35 ms	9701 bytes/sec	0.06 %
Without security	72.11 ms	9804 bytes/sec	0.06 %

TABLE III: CPU usage (%) with different network setup. The data is in the format of (Ethernet_CPU, Wi-Fi_CPU, 4G_CPU).

ROS Message	FastDDS	CycloneDDS	Zenoh-TCP	Zenoh-broker	MQTT-nobroker	MQTT-broker
Array1k	(6.04, 6.01, 5.89)	(0.04, 0.04, 0.04)	(0.05, 0.05 , 0.06)	(0.06, 0.05, 0.06)	(0.05, 0.05, 0.05)	(0.05, 0.05, 0.05)
Array4k	(6.04, 6.05, 5.87)	(0.04, 0.03, 0.04)	(0.05, 0.05, 0.06)	(0.06, 0.05, 0.06)	(0.05, 0.05, 0.05)	(0.05, 0.05, 0.05)
Array16k	(0.18, 5.58, 6.06)	(0.05, 0.04, 0.05)	(0.06, 0.06, 0.06)	(0.06, 0.06, 0.06)	(0.05, 0.05, 0.05)	(0.04, 0.05, 0.06)
Array64k	(0.67, 5.57, 6.05)	(0.07, 0.06, 0.06)	(0.06, 0.05, 0.05)	(0.07, 0.05, 0.06)	(0.05, 0.05, 0.06)	(0.05, 0.03, 0.06)
Array256k	(6.64, 4.42, 5.81)	(0.19, 0.09, 0.06)	(0.09, 0.05, 0.04)	(0.09, 0.05, 0.05)	(0.08, 0.05, 0.05)	(0.07, 0.04, 0.05)
Array1m	(6.92, 4.96, 6.05)	(0.51, 0.08, 0.11)	(0.16, 0.04, 0.03)	(0.18, 0.03, 0.04)	(0.18, 0.13, 0.13)	(0.17, 0.12, 0.14)
Array2m	(7.01, 5.58, 0.02)	(0.77, 0.08, 0.19)	(0.30, 0.04, 0.03)	(0.32, 0.04, 0.03)	(0.30, 0.17, 0.19)	(0.20, 0.18, 0.19)
PointCloud512k	(6.30, 4.98, 0.03)	(0.30, 0.07, 0.07)	(0.12, 0.06, 0.03)	(0.12, 0.04, 0.04)	(0.12, 0.08, 0.07)	(0.12, 0.09, 0.09)
PointCloud1m	(7.08, 0.04, 0.02)	(0.57, 0.09, 0.12)	(0.18, 0.04, 0.03)	(0.19, 0.04, 0.04)	(0.17, 0.12, 0.14)	(0.21, 0.15, 0.14)
PointCloud2m	(7.08, 8.40, 0.04)	(0.84, 0.11, 0.19)	(0.32, 0.03, 0.03)	(0.30, 0.03, 0.04)	(0.33, 0.18, 0.20)	(0.22, 0.22, 0.20)

VI. Conclusion

In this paper, we comprehensively evaluate three types of middleware used in ROS 2 system. We compared the latency and throughput of FastDDS, CycloneDDS, Zenoh, and MQTT with ROS Messages under different network setups, including Ethernet, Wi-Fi, and 4G. Moreover, we tested the performance of these middlewares on a real robot platform, TurtleBot 4. We set a host and the TurtleBot 4 under the local Wi-Fi network, and use the host to control the TurtleBot 4 to run a square-shape trajectory for 96 s through different middleware. The trajectories were recorded by the Optitrack MOCAP system. Our experimental results reveal that under Ethernet, CycloneDDS has minimal latency and throughput, due to its UDP multicast mechanism. Under wifi and 4g, zenoh has better performance. On the real robot platform, zenoh's trajectory drift error is the smallest.

Acknowledgment

This research work is supported by the Academy of Finland's RoboMesh project (Grant No. 336061) and AeroPolis project (Grant No. 348480), and by the R3Swarms project funded by the Secure Systems Research Center (SSRC), Technology Innovation Institute (TII).

References

[1] Jiaqiang Zhang, Farhad Keramat, Xianjia Yu, Daniel Montero Hernández, Jorge Pena Queralta, and Tomi Westerlund. Distributed robotic systems in the edge-cloud continuum with ros 2: a review on novel architectures and technology readiness. In *2022 Seventh International Conference on Fog and Mobile Edge Computing (FMEC)*, pages 1–8. IEEE, 2022.

[2] Gerardo Pardo-Castellote. Omg data-distribution service: Architectural overview. In *23rd International Conference on Distributed Computing Systems Workshops, 2003. Proceedings.*, pages 200–206. IEEE, 2003.

[3] Oasis message queuing telemetry transport (mqtt) tc. https://www.oasis-open.org/committees/mqtt/. [Online].

[4] Eclipse. Zenoh. https://zenoh.io/. [Online].

[5] Victor Mayoral-Vilches, Ruffin White, Gianluca Caiazza, and Mikael Arguedas. Sros2: Usable cyber security tools for ros 2. In *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 11253–11259. IEEE, 2022.

[6] Jongkil Kim, Jonathon M. Smereka, Calvin Cheung, Surya Nepal, and Marthie Grobler. Security and performance considerations in ros 2: A balancing act, 2018.

[7] Yuya Maruyama, Shinpei Kato, and Takuya Azumi. Exploring the performance of ros2. In *Proceedings of the 13th International Conference on Embedded Software*, pages 1–10, 2016.

[8] Daishi Yoshino, Yutaka Watanobe, and Keitaro Naruse. A highly reliable communication system for internet of robotic things and implementation in rt-middleware with amqp communication interfaces. *IEEE Access*, 9:167229–167241, 2021.

[9] Roger A Light. Mosquitto: server and client implementation of the mqtt protocol. *Journal of Open Source Software*, 2(13):265, 2017.

[10] Apex.ai: performance_test. https://gitlab.com/ApexAI/performance_test.

[11] Christophe Bédard, Ingo Lütkebohle, and Michel Dagenais. ros2_tracing: Multipurpose low-overhead framework for real-time tracing of ros 2. *IEEE Robotics and Automation Letters*, 7(3):6511–6518, 2022.

[12] Pranav Pandey and Ramviyas Parasuraman. Empirical analysis of bi-directional wi-fi network performance on mobile robots in indoor environments. In *2022 IEEE 95th Vehicular Technology Conference: (VTC2022-Spring)*, pages 1–7, 2022.

[13] Takahisa Kuboichi, Atsushi Hasegawa, Bo Peng, Keita Miura, Kenji Funaoka, Shinpei Kato, and Takuya Azumi. CARET: Chain-Aware ROS 2 Evaluation Tool. In *Proceedings of IEEE international conference on embedded and ubiquitous computing (EUC)*, 2022.

[14] Tobias Kronauer, Joshwa Pohlmann, Maximilian Matthe, Till Smejkal, and Gerhard Fettweis. Latency analysis of ros2 multi-node systems, 2021.

[15] Zihang Li, Atsushi Hasegawa, and Takuya Azumi. Autoware˙perf: A tracing and performance analysis framework for ros 2 applications. *Journal of Systems Architecture*, 123:102341, 2022.

[16] Y Justin Dhas and P Jeyanthi. A review on internet of things protocol and service oriented middleware. In *2019 International Conference on Communication and Signal Processing (ICCSP)*, pages 0104–0108. IEEE, 2019.

[17] Juan José López Escobar, Felipe Gil-Castiñeira, and Rebeca P Díaz Redondo. Decentralized serverless iot

dataflow architecture for the cloud-to-edge continuum. In *2023 26th Conference on Innovation in Clouds, Internet and Networks and Workshops (ICIN)*, pages 42–49. IEEE, 2023.

[18] Christophe Bédard, Pierre-Yves Lajoie, Giovanni Beltrame, and Michel Dagenais. Message flow analysis with complex causal links for distributed ros 2 systems. *Robotics and Autonomous Systems*, 161:104361, 2023.

[19] Lennart Puck, Philip Keller, Tristan Schnell, Carsten Plasberg, Atanas Tanev, Georg Heppner, Arne Roennau, and Rüdiger 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)*, pages 1670–1676. IEEE, 2021.

[20] Guo Fu, Yanfeng Zhang, and Ge Yu. A fair comparison of message queuing systems. *IEEE Access*, 9:421–432, 2020.

[21] Wen-Yew Liang, Yuyuan Yuan, and Hsiang-Jui Lin. A performance study on the throughput and latency of zenoh, mqtt, kafka, and dds, 2023.

[22] Kydos. Minimizing discovery overhead in ros2. https://zenoh.io/blog/2021-03-23-discovery/#leveraging-resource-generalisation. [Online].