Volume 1, 2025, Pages 01-10 *Elaris Computing Nexus*
| Regular Article | Open Access

Performance Evaluation of DDS Middleware Impact on Packet Loss and Latency Across QoS Profiles and Platforms

Nandhini Priya T
Department of Electrical and Electronics Engineering, Excel Engineering College, Komarapalayam, Namakkal, India.
nandhinipriyat.eec@excelcolleges.com

Article Info
Journal of Elaris Computing Nexus
https://elarispublications.com/journals/ecn/ecn_home.html

https://doi.org/10.65148/ECN/2025001

Received 10 December 2024
Revised from 30 January 2025
Accepted 02 February 2025
Available online 10 February 2025
Published by Elaris Publications.

Corresponding author(s):
Nandhini Priya T, Department of Electrical and Electronics Engineering, Excel Engineering College, Komarapalayam, Namakkal, India.
Email: nandhinipriyat.eec@excelcolleges.com

Abstract – This article presents a performance analysis of the DDS middleware, focusing on packet loss and delay across various Quality of Service (QoS) configurations and network modes. A testbed was established with three physical nodes (Mac13, Mac15, and Raspberry Pi4) utilizing Gigabit Ethernet and Wi-Fi interfaces to configure diverse operational conditions. These experiments were conducted across an Ethernet and a 5 GHz WiFi network to evaluate the performance of DDS under deterministic and probabilistic network settings. A series of experiments were conducted utilizing various payload sizes (ranging from 92 to 1024 bytes) and distinct Quality of Service configurations, including Best Effort, B2STKA, R10TKL, and B10TKL. The results provide a summary of the timing of packet loss, the impact of QoS regulations on latency, and the performance trade-offs among operating systems (Linux, macOS, and Raspberry Pi OS).

Keywords – Data Distribution Service, Packet Loss, Latency, Quality of Service, Network Performance, Operating Systems, Real-Time Systems, Ethernet, Wi-Fi, Middleware.

I. INTRODUCTION

Reliable Communications in distributed frameworks necessitate the participation of a middleware, communication channel, and an application. The program executes core functionalities and often assumes one of two critical functions: server or client. Irrespective of the function, the application may function as either a receiver and a transmitter (such as a client receiving feedback, or transmitting requests, correspondingly. The application counterparts interact over the communication channel, sometimes using the system middleware, which encompasses a collection of services situated above the network or under the application layer, often enabling dependable communication and coordination among dispersed application peers (e.g., RSocket and FSocket). Middleware generally offers application developers elevated programming abstractions (e.g., using remote items in lieu of sockets) and can also furnish an intermediary broker to dissociate the connection between receiver and sender (e.g., ZeroMQ), among other options.

The Data Distributed Service (DDS) is an official protocol established by the OMG (object management group), widely utilized in embedded frameworks, particularly within aerospace, military sectors, and industrial automation. DDS delineates an API intended for facilitating real-time data dissemination. It employs a publish-subscribe (Pub/Sub) communication mechanism and accommodates both data-object oriented data structures and messaging structures. Over time, DDS has progressed via several revisions and additions, including both commercial and open-source implementation. Numerous application sectors are embracing it, including smart grid, healthcare, military, aerospace, autonomous systems, industrial IoT, and robotics. **Table 1** enumerates some notable DDS middleware systems currently available.

DDS utilizes a data-oriented Pub/Sub system to provide efficient and reliable communications in instantaneous, mission-based, and diverse networked systems. It simplifies the intricacies of communication control, decreasing model interdependence while improving reliability and adaptability via wrapping. DDS enhances data compatibility by

accommodating many formats, such as OMG IDL (Interface Definition Language) and XML (Extensible Markup Language). Moreover, it is interoperable with UDP and TCP socked, facilitating efficient and flexible data transfer. It has comprehensive QoS standards that accommodate many communication topologies, including the specification of data transfer periods and the activation of temporal-event triggers. **Table 2** delineates significant milestones in the growth of DDS middleware, spanning from the original 2004 standardization to the most recent improvements in 2024.

Table 1. Major DDS Middleware Technologies

Authors	Middleware Name	Release Year	Version	Free & Open Source	Company/Organization
Liang, Yuan, and Lin, [1]	Cyclone DDS	May. 2024	0.10.5	Yes	Eclipse Foundation (via ZettaScale)
Dust et al. [2]	GurumDDS	-	3.2.0	No	GurumNetworks
Krinkin et al. [3]	Vortex OpenSplice	Mar. 2021	6.9.0	Yes	ADLINK Technology
Fujdiak et al. [4]	CoreDX DDS	2020	5.0.0	No	Twin Oaks Computing
Bode et al. [5]	Fast DDS	Mar. 2025	3.2.0	Yes	eProsima

Table 2. Sequential Chronology of DDS Evolution

Year	Milestones
2004	OMG's first DDS 1.0 release.
2006	DDS 1.2 protocol is created; initial industrial application starts.
2007	RTI introduces DDS middleware, which improves industrial application scalability.
2008–2010	DDS advances interest in aerospace and military due to its reduced latency communications. Initial use of IoT and intelligent grids starts.
2009	Initial ROS distribution launched: Mango Tango.
2010	ROS 1 launched.
2010	ROS employs DDS principles implicitly through integration layers.
2012	OMG released DDS v1.4, which has enhanced dynamic discovery capabilities and QoS protocols.
2014	Commencement of DDS Security Protocol Design. ROS 2 designates DDS as the standard middleware, enhancing acceptance in robotics and automations.
2015	DDSI-RTPS 2.20 released, enhancing real-time scalability.
2016	DDS used in driverless cars for instantaneous communications.
2017	Release of ROS 2, which formally designates DDS as the main middleware.
2018	Secure DDS version 1.1 has been finished, including encryption, access control, and authentication features. The ROS 2 “Ardent Apalone” version incorporates DDS, supplanting the centralized design of ROS 1. Prominent DDS providers (OpenDDS, Fast DDS, and RTI Connext) enhance their support for autonomous systems and IoT.
2019	Eclipse Cyclone DDS becomes the standard ROS 2 middleware and is extensively used in automations and robotics.
2020	X-Types v1.3, with flexible data structures, published to provide scalability in DDS communications. The application of DDS has broadened in self-driving cars, cloud-end-end fusion systems, intelligent grids, and drones for industrial robots.
2021	DDS security used in military robots and UAV swarms to mitigate rogue node assaults.
2023	FogROS2 Scheduler/Controller utilizes DDS to provide safe worldwide networking in decentralized robotics.
2023	Research emphasizes the latency-based trade-offs associated with implementing DDS privacy in real-time networks.
2024	Recent DDS updates (e.g., Fast DDS 3.2, RTI Connext 7.4) provide minimal delay and improved compatibility, accommodating ML/AI pipelines for real-time inferences and remote robotic learning.

Research on DDS and OPC UA for compatibility is now underway. Endeley et al. [6] introduced an intelligent gateway to facilitate compatibility between DDS and OPC UA within an IIoT context. The OPC UA function suit is integrated via the intelligent gateway, facilitating communication between DDS and OPC UA. Nonetheless, it is confined to OPC UA and lacks support for OPC UA Pub-Sub.

Cheikh, Mastouri, and Hasnaoui [7] advocated for the employment of data-centric middleware to facilitate collaborative vehicular models for vehicle-to-infrastructure or V2V interaction. Their suggested data-centric middleware utilizes the Pub-Sub concept and incorporate QoS capability. They offered a comprehensive distributed architecture using a DDS as an

embedded lab framework for avionics. The proposed system enables the evaluation of modular avionics alongside the whole model, which includes simulated entities and actual hardware, by integrating several communication bus protocols and enhancing adaptability via data gateways and DDS. They also presented a DDS gateway design facilitating interaction across DDS regions for extensive CPS (cyber-physical systems). Their design has four primary elements: network module, routing manager, topic manager, and interface module, along with a mechanism to mitigate blocks via sequential access.

The aforementioned studies indicate that DDS is used across several domains, such as IIoT, automotive, avionics, and CPS, highlighting the increasing significance of connectivity and interoperability across diverse systems. Furthermore, the majority of current research emphasizes gateways for DDS and OPC UA, with less consideration for scalability. Our work focuses on the systematic comparison of communication capabilities of DDS middleware in different network scenarios and profiles of Quality of Service (QoS). Through structured experimentation across several hardware platforms and operating systems, the experiment seeks to measure core performance variables like packet loss and latency.

The remaining sections of this study is organized in the following manner: Section II provides a background study of our work, highlighting the effect of QoS policies on real-time communication, and comparative network studies across different protocols and platforms. Section III describes our experimental architecture, data collection, as well as QoS profiles and system configuration. Section IV and V provide a detailed discussion of our findings. Lastly, Section VI concludes the study and highlights the significance of evaluating DDS middleware across various network setups and QoS profiles, which impact packet loss and latency.

II. BACKGROUND STUDY

Impact of QoS Policies On Real-Time Communication

Study on validating QoS policies for DDS is scarce. From a model-riven viewpoint, the specification of QoS policies is a component of DDS architecture modeling. This section summarizes current research in DDS modeling. We also examine several studies on the integration of DDStosmart grids. Bertaux et al. [8] examined the employment of DDS QoS rules to fulfill the quality needs of real-time systems comprehensively. They specifically concentrate on quality rules specification for data availability and latency using the MARTE (modeling and analysis of real-time embedded systems) framework. In this paper, the significance of QoS policy modeling in DDS is comprehensively elucidated. Their findings indicate that both the growth process and the quality of communication may be enhanced.

The work by Putra and Kim [9] advocated customizing the DDS participant discovery for a combat system depending on the system's attributes and needs. The customized model interacts with several levels in DDS, including those near to the discovery system and more remote ones, such as the interface layer. They suggested a model-based methodology for tailoring DDS to accommodate WSNs. A layer was established underneath DCPS in DDS to facilitate the adoption of DDS for WSNs. In their research [10], Alaerjan identified the deficiencies in DCPS to enhance DDS standard coverage using UML. The enhanced model serves as the foundation for developing flexible DDS to accommodate nodes with constrained computational resources.

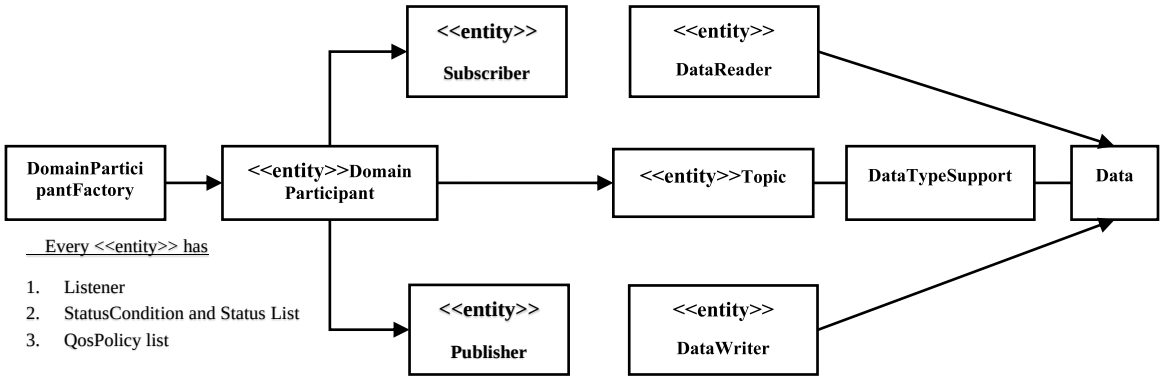

Fig 1. UML Flowchart Illustrating the DDS Data-Oriented Publish-Subscribe Interfaces.

Fig. 1 depicts the comprehensive data-centric Pub-Sub paradigm, including the following entities: Topic, Subscriber, Publisher, DataReader, DataWriter, and Domain Participant. All these entities inherit from Entity, signifying their capacity to be set by QoS rules, enabled, receive event notifications via listener objects, and accommodate situations that the application may await. Every entity base class specialization has a matching dedicated receiver and an entity of appropriate QoSPolicy settings. The publisher denotes the entities accountable for data dissemination. A Publisher can disseminate data of several forms. A DataWriter serves as a keyed interface to a Pub; participants employ DataWriter(s) to convey the value and modifications of data of a certain form. Upon receipt of novel data values, the Publisher is responsible for deciding the appropriate timing for issuing the associated message and executing the release, in accordance with its QoS, the QoS associated with the relevant DataWriter, or/and its internal condition.

Certain QoS regulations are now implemented in several Pub-Sub systems or models, including CORBA Notification Service, Distributed Asynchronous Collections, and Java Message Service. This pertains to message dependability, message priority, earliest delivery time, expiration time, message ordering, or duplicate message identification, for example. These QoS criteria may be supported or unsupported, depending upon the architecture. To our knowledge, QoS factors like as loss ratio, jitter, availability, bandwidth, and latency, extensively examined in the direct communication framework, are insufficiently addressed in Pub-Sub models.

An et al. [11] review various QoS strategies influenced by fluctuations in network and computing resources, as well as the automated configuration of Pub/Sub intermediary software in cloud settings. The suggested method modifies the network transport platform to synchronize with the middleware, addressing QoS configuration challenges in DRE (distributed real-time embedded) models inside cloud settings. The methodology employs ANN techniques to dynamically ascertain the appropriate transport channel for the Pub/Sub intermediary software after the execution of the DRE model. ANN instruments are learned with data settings to optimize QoS and forecast feedback duration required in DRE models. The method employed by the middleware employed the ANT (adaptive network transports) architecture to choose the optimal transport protocol, considering different QoS factors in relation to the availability of computing power.

Comparative Network Studies Across Platforms and Protocols

According to O'Ryan et al. [12], the selection of an operating system has a major influence on the performance of middleware, especially in real-time and distributed applications. They analyzed real-time communication protocols and found that Linux, due to its monolithic kernel and its full customization capabilities, is much faster in situations that promote low latency, high throughput communication. Because Linux supports explicit thread handling and core binding, it can be particularly useful to middleware that performs high-frequency messaging or real-time data flow. MacOS, meanwhile, is stable and easy to work with but lacks versatility in such optimizations and Raspberry Pi OS (a lightweight Linux operating system) is suitable in terms of edge execution due to affordably priced, but ultimately restricted by hardware constraints.

Middlewares also rely heavily on the type of network they use. The post-processing data will be based on Ethernet connections as these connections offer better stability, lower latency, and less packet loss than Wi-Fi connections, as mentioned by Carrascosa-Zamacois et al. [13]. The benefit of Wi-Fi, its convenience, comes at the cost of variable latency and increased interference susceptibility, and these factors can negatively impact middleware performance, particularly in applications where timely and reliable data delivery are required. This can also be inferred in the IoT middleware benchmarks, where Ethernet is more capable of greater throughput and consistent performance.

Transport modes, including unicast and multicast also affect middleware scalability and resource demands. Unicast is compatible and reliable, in that dedicated streams of data can be dedicated to each client, although the resource needs are linear with the number of clients. Multicast, in contrast supports efficient one-to-many communication, lowering server and network load with shared communication. Nevertheless, the use of multicast requires network architecture support and OS, and can be less dynamic or complicated in heterogeneous or wireless networks. Collectively, these works express that optimal middleware performance is obtained when there is a specific mapping of OS, network type and transport mode to application demands and deployment context.

III. DATA AND METHODS

To objectively analyze the communication performance of the DDS middleware, this study presented an experimental platform that included carefully controlled hardware platforms, network topologies and stacked software layers. The goal of this part was to record the packet behavior in real time, the change in latency and the packet loss patterns using appropriate data collection techniques and Industry standard analysis tools based on different QoS configurations.

Experimental Architecture

The hardware platform comprised three physical nodes, a publishing workstation (Mac13), a subscribing workstation (Mac15) and a Raspberry Pi 4 Model B. Every node was capable of Gigabit Ethernet interface and 5 GHz wireless LAN. The experiments were performed both over Ethernet and Wi-Fi to model different operating environments, e.g., deterministic links with Ethernet and probabilistic links in the case of Wi-Fi. The machines were run with macOS Ventura (v13.6.5), the Linux distribution of Ubuntu 22.04 LTS, and the Raspberry Pi distribution (Debian-based), providing a wide range of execution environment.

The communication structure between the DDS was implemented based on eProsima Fast DDS v2.10 that supported dynamic QoS policy setup, participant discovery, and real-time publish-subscribe. Wireshark v4.2 was used as the main packet capture application with the following filters frame.len == 194 being used to narrow down and capture DDS specific packets. In each test uniform sized data packets (payloads of 92 to 1024 bytes) were published at a uniform interval, and the transmission integrity was measured on 10,000 iterations.

Data Collection and Measurement

The key performance indicators were packet loss ratio and end-to-end latency, which were captured with timestamps placed in DDS message headers and trace packet information on the system level. The ratio was calculated as the packet loss ratio (PLR) by comparing total number of published packets P_{total} with successful packets received P_{recv} in Eq. (1).

$$\text{PLR} = \frac{P_{total} - P_{recv}}{P_{total}} \times 100 \quad (1)$$

Latency (L) was defined as the time arrival difference between the DDS message timestamp $T_{pub}^{(i)}$ of the publisher node and the timestamp of its reception $T_{sub}^{(i)}$ at the subscriber node. This is expressed as Eq. (2) where i varies to every message.

$$L_i = T_{sub}^{(i)} - T_{pub}^{(i)}, \quad \text{for } i = 1,2,\dots,N - W + 1 \quad (2)$$

The short-term jitter was solved by computing a rolling average latency measure with window of size W as in Eq. (3).

$$\bar{L}_j = \frac{1}{W}\sum_{k=j}^{j+W-1} L_k \quad \text{where } j = 1,2,\dots,N - W + 1 \quad (3)$$

The strategy ironizes out high signal fluctuations and allows displaying long-term structural latency tendencies.

QoS Profiles and System Configurations

The DDS middleware was both set with default and custom QoS settings. In particular, they contested the following configurations: Best Effort, B10TKL, R10TKL and B2STKA where reliability, durability, and history depth combinations vary in each case. These QoS profiles were then plotted to situations in which low latency or high reliability were prioritized as per application objective. The test procedure contained 10 sets in each of the configurations and each of the sets involved 10,000 messages transmitted. A restart of the system was made between tests to counteract cumulative memory or buffer effects. **Table 3** summarizes the details of testing conditions used in each of the test profiles.

Table 3. Test Configurations and QoS Settings

Test ID	Network Type	OS Platform	QoS Profile	Payload Size (Bytes)	Iterations	Interval (ms)
T1	Ethernet	Linux	Best Effort	194	10,000	10
T2	Wi-Fi (5 GHz)	macOS	B10TKL	194	10,000	10
T3	Ethernet	Raspberry Pi	R10TKL	194	10,000	10
T4	Wi-Fi (5 GHz)	Linux	B2STKA	194	10,000	10
T5	Ethernet	macOS	B10TKL	1024	10,000	5

Collisions, jitter and dropped frames were observed in network interfaces with a particular focus on multicast group results and interface queue latencies. Latency anomalies were matched with the operating system kernel logs where measurement validity was addressed.

IV. RESULTS

We evaluate the incidence of communication packet delay, considering the data volume and rate specifications of the users' application. We evaluated the efficacy of the DDS communication structure using DDS intermediary software via tests designed to assess network performance and detect occurrences of packet loss. We assessed the efficacy of the Ethernet protocol by transmitting successive batches of DDS data packets and quantifying the packet delay from the Pub to the Sub. We repeated the procedure for both multicast and unicast setups to ascertain the permissible degree of frame delay that might not hinder the application's efficiency. **Fig. 2** and **Fig. 3** depict the findings, demonstrating the duration of the trial in correlation with the incidence of packet delay, as documented by Wireshark. The trials were performed on shared-time servers, indicating that implementation on systems with real-time OS (operating systems) might enhance efficiency metrics further.

Fig. 2 presents a Wireshark screenshot of client data transferred over the DDS standard. **Fig. 3** illustrates the DDS standard frames, which were captured and processed protocol frames according to their packet size in Wireshark. The efficiency analysis shown in **Fig. 4** examines the correlation between test iterations and packet delay for DDS functioning under a best-effort QoS setting. Furthermore, **Fig. 5** presents an examination of the frequency of dropped data packets and the reliability of the DDS 5 GHz wireless link. It presents the exact technological specifications of the wireless interface employed by both the publisher and subscriber. The noted associations between the received and released data in DDS transport suggests a minimal packet loss rate, possibly attributable to the absence of networking traffic in the local test setting.

```
0000  01 2c ad eb db 77 0f a0  ab 40 52 54 50 53 53 02   .,...w...@RTPS..
0010  04 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ................
0020  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ................
0030  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ................
0040  32 32 20 4e 58 2d 44 43  56 2d 53 2d 31 20 56 4f   22 NX-DCV-S-1 VO
0050  2d 4c 31 00 00 00 00 00  00 00 00 00 00 00 00 00   -L1.............
0060  42 55 2d 4c 32 00 00 00  00 00 00 00 00 00 00 00   BU-L2...........
0070  31 2d 56 4f 2d 4c 31 00  00 00 00 00 00 00 00 00   1-VO-L1.........
0080  44 4f 2d 30 30 2d 31 00  00 00 00 00 00 00 00 00   DO-00-1.........
0090  02 05 04 20 2a cf b1 68  00 00 00 00               ... *..h....
```

Fig 2. Client data with 92 Bytes in the Retrieved DDS Protocol, As Analyzed by Wireshark.

Dnc	Source	Destination	Protocol	Length	Info
75956 118,6019501	192.168.1.137	192.168.1.44	RTPS	194	INFO_TS, DATA => Smartdevice
75959 118,6018903	192.168.1.137	192.168.1.44	RTPS	194	INFO_TS, DATA => Smartdevice
75960 118,6018901	192.168.1.144	192.168.1.44	RTPS	194	INFO_TS, DATA => Smartdevice
75961 118.6018913	192.168.1.137	192.168.1.44	RTPS	194	INFO_TS, DATA => Smartdevice
75982 118,6018963	192.168.1.144	192.168.1.44	RTPS	194	INFO_TS, DATA => Smartdevice
75963 118,6018917	192.168.1.137	192.168.1.44	RTPS	194	INFO_TS, DATA => Smartdevice
75964 118,6018998	192.168.1.144	192.168.1.44	RTPS	194	INFO_TS, DATA => Smartdevice

```
Frame 194: 194 bytes on wire (1552 bits), 194 bytes captured (1552 bits) on interface 0
Ethernet II, Src: Mellanox_9e:60:84 (90:e2:ba:9e:60:84), Dst: Ivanairburn79_62:4f:f0
Internet Protorol Version 4, Src: 192.168.1.137, Dst: 192.168.1.44, 192.168.1.44
Real-Time Publish-Subscribe Wire Protocol
  Protocol version
  Protocol : 2.1
  Vendor: RTPS
  Sennors OpenDDS
  GUID Prefis: 263eeb204e00
  Submessage ID: INFO_TS (6d)
  Flags: 0x0000
  Octets to inline QoS: 1
  Writer entity ID: 0000002e
  Octets to inline writer key
  Reader entity ID: 0x000000
  Subtype: USER_DEFINED_DATA (a operation ="defined writer-reader" with key ky 0)
  User Data: InfinityConflict ▸ User Data:228
```

Fig 3. Obtained DDS Protocol Packets; Wireshark filters (Frame.Len==194).

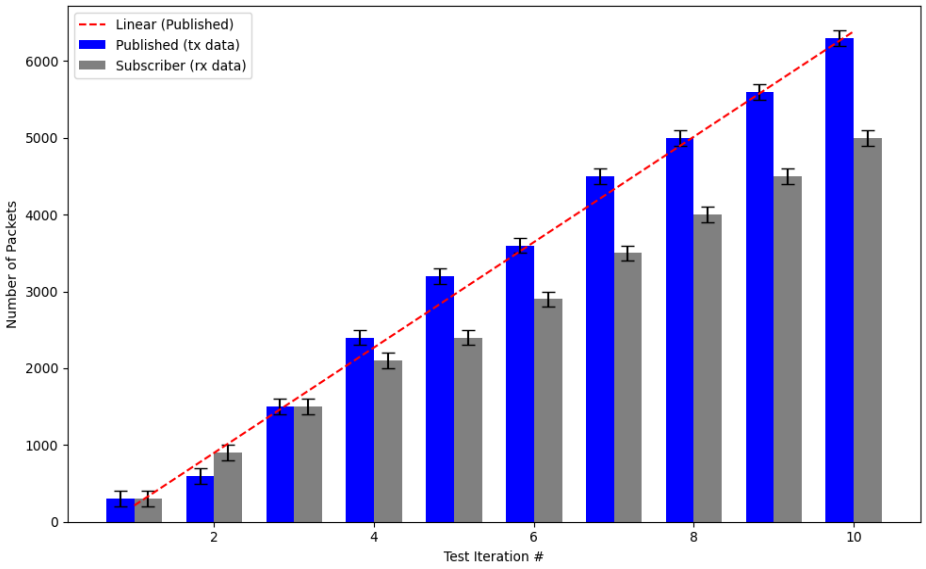

Fig 4. Best-Effort QoS and DDS.

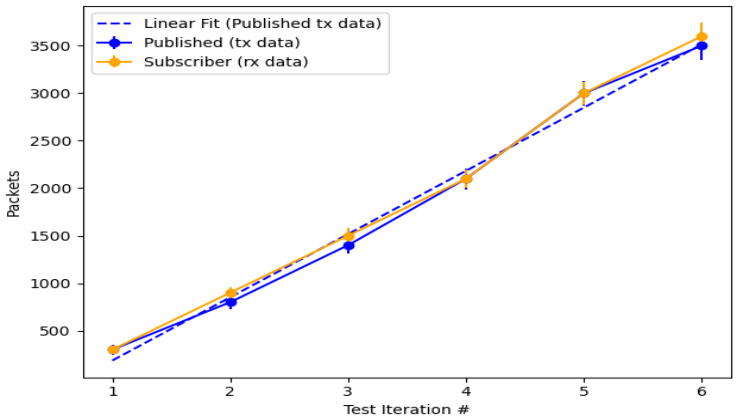

Fig 5. Packet Delay and Trustworthiness Efficiency for DDS On 5 GHz Wi Fi with A Best Effort QoS Configuration.

Fig. 6 and **Fig. 7** provide a detailed examination of the rolling average latency, taking into account various QoS rules, on 2 workstations designated as publisher (Mac13) and subscriber (Mac15). The statistics depict the fluctuations in latency patterns over different packet amounts, highlighting the impacts of various QoS rules, including R10TKL and B10TKL. Latency, quantified in ms, assesses the immediate efficiency of the communication channel.

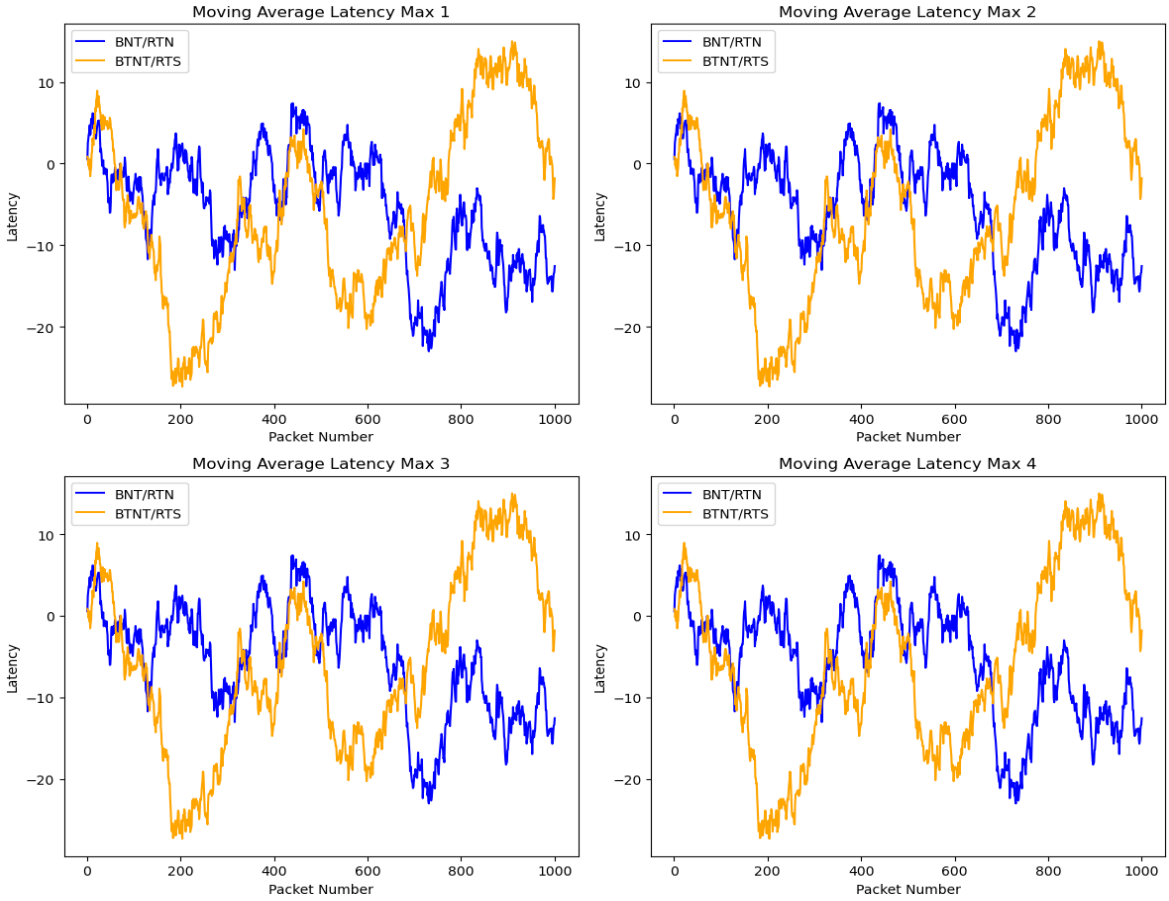

Fig 6. Analysis of Rolling Average Latency Patterns Across Various QoS Rules in a Pub/Sub Framework.

Fig. 7 presents a bar chart illustrating latency averages, offering a clear depiction of the differing efficiency across various groupings of OS across many QoS situations, including B2STKA, R10TKL, and B10TKL. The findings illustrate the delay anticipated when deploying these systems under specified QoS conditions. The results indicate that QoS settings substantially affect the delay encountered by the systems. B10TKL demonstrates differing latencies when utilized with MacOS and Linux in contrast to MacOS and Raspberry Pi, as seen in **Fig. 7**. These findings are crucial for enhancing real-time connectivity standards in microgrid environments, where rapid data transmission is critical.

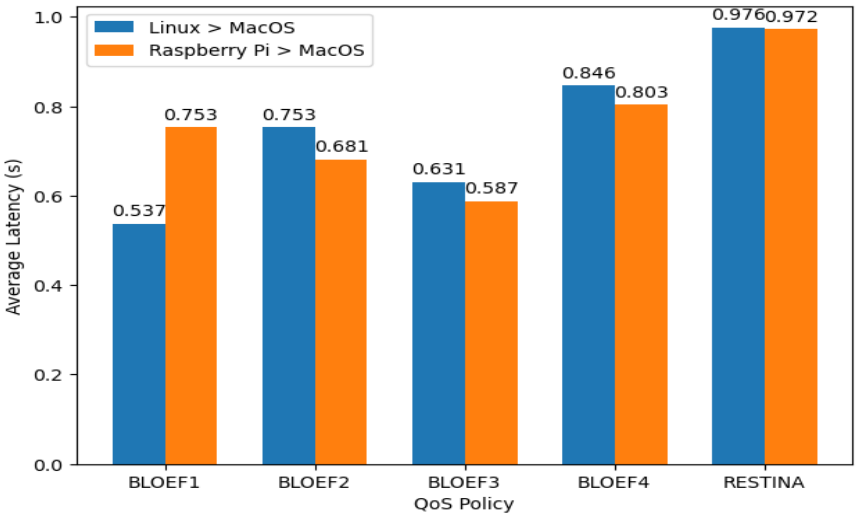

Fig 7. Comparison of Average Latency Trends Relative to QoS Policy Between MacOS and Linux, As Well as Between Raspberry Pi and MacOS Computers.

V. DISCUSSION

The dissemination of real-time data has lately become a significant study domain. A workshop focused on the subject "1st International Workshop on Data Distribution in Real-Time Systems (DDRTS'2003)" took place in May 2003. The OMG enhances research initiatives by normalizing data delivery inside an intermediary service. The formulation of dynamic scheduling techniques in data-centric Pub-Sub systems operating on real-time networks is a significant research challenge, and in recent years, several teams and businesses have actively engaged in this area. The issue of planning the transmission of real-time data is addressed in [14]. It offers a refined variant of the Longest-processing-time-first (LPT), which minimizes overhead.

Comparable research [15] delineates a Broadcast on Demand methodology that organizes the broadcast using the 1st deadline first, hybrid, or periodic scheduling techniques. The study presented in [16] outlines a conjectural data distribution service, which leverages temporal and geographic reference locality to ascertain the data to be distributed. These solutions cater for customers' deadline timing limitations but neglect both data time coherence and the employment of prevailing real-time systems.

Embedded sensor networks have been a significant focus of research efforts concerning data dissemination. Although the work presented below offers significant insights into addressing data distribution issues in sensor networks, it fails to account for the real-time nature of both the applications and data. In other words, neither time limits for data supply nor time coherence of the data are maintained. A study by Zervopoulos et al. [17] on real-time data dissemination was conducted at the University of Virginia (UVa) regarding wireless sensor networks. This work addressed the time limits of requests. Furthermore, time validity is acknowledged in that data levels are presented prior to their expiration, accompanied by appropriate confidence levels. Nonetheless, it does not guarantee that the information is time-valid upon arrival to the requester.

Critical findings during the evaluation of the DDS communication system indicated how variances in network situations and QoS settings manifest in DDS performance, especially in terms of packet loss and delay. The results indicate that when it comes to best effort QoS settings it only showed a few drops indicating that DDS can provide quality traffic assurances in less congested networks with controlled environments. Communication was over Ethernet in both unicast and multicast mode and this generally supported reliable data delivery. But multicast transmission showed a little more packet loss rate. Almadani et al. [18] concluded that multicast in DDS behaves poorly when the subscriber load becomes high and there are dynamic changes occurring in the network. Their study had shown that correct implementation of multicast is critical to the preservation of reliability especially in large or dense networks.

Latency analysis identified the importance of QoS policies in terms of influencing responsiveness of communications. Custom configuration options like B10TKL and R10TKL resulted in detectable differences in latency, indicating that these settings should be adjusted to meet application specific timing needs. The effect of QoS on latency has been quite well analyzed by Zhu et al. [19], where he has observed that DDS enhances the control of delivery guarantees and this fact greatly affects the timing behavior in distributed systems. The results hereby support the submission that QoS tuning is important in ensuring stability in performance, particularly in real-time environments.

Furthermore, the differences in the latency with different platforms also demonstrate the significance of the operating system on which it runs and the hardware. The findings showed that Linux performed steadily compared to macOS, and that Raspberry Pi systems were highly competitive in terms of latency in certain configurations. This observation reaffirms the finding of An et al. [20], which illustrated that the execution environment is also a key factor in the performance of a real-

time middleware. Their work highlighted that the system-level delays could not be disregarded in performance-sensitive systems even though middleware such as the DDS was used.

VI. CONCLUSION

The assessment of DDS middleware under different network topologies and various values of QoS parameters highlights the importance of those parameters in determining the latency and packet loss. Although Ethernet connections provide more stable results, Wi-Fi connections provide acceptable communication results when properly optimized QoS settings are established but these connections have a greater level of variability in delays. Moreover, the involvement of different operating systems revealed certain performance inconsistency, where Linux-based computers produced comparatively better results than macOS and Raspberry Pi OS. These observations lead to the conclusion about the need in system-level improvements and how real-time operating systems can help one substantially improve DDS performance in a real-time application.

CRediT Author Statement

The author reviewed the results and approved the final version of the manuscript.

Data Availability

The datasets generated during the current study are available from the corresponding author upon reasonable request.

Conflicts of Interests

The authors declare that they have no conflicts of interest regarding the publication of this paper.

Funding

No funding was received for conducting this research.

Competing Interests

The authors declare no competing interests.

References

[1]. W.-Y. Liang, Y. Yuan, and H.-J. Lin, “A performance study on the throughput and latency of Zenoh, MQTT, Kafka, and DDS,” arXiv (Cornell University), Jan. 2023, doi: 10.48550/arxiv.2303.09419.

[2]. L. J. Dust, E. Persson, M. Ekstrom, 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, Jul. 2022, doi: 10.1109/indin51773.2022.9976160.

[3]. K. Krinkin, A. Filatov, A. Filatov, O. Kurishev, and A. Lyanguzov, “Data Distribution Services Performance Evaluation Framework,” 2018 22nd Conference of Open Innovations Association (FRUCT), pp. 94–100, May 2018, doi: 10.23919/fruct.2018.8468297.

[4]. R. Fujdiak., “Security and Performance Trade-offs for Data Distribution Service in Flying Ad-Hoc Networks,” 2019 11th International Congress on Ultra-Modern Telecommunications and Control Systems and Workshops (ICUMT), pp. 1–5, Oct. 2019, doi: 10.1109/icumt48472.2019.8970670.

[5]. V. Bode, C. Trinitis, M. Schulz, D. Buettner, and T. Preclik, “DDS Implementations as Real-Time Middleware – A Systematic Evaluation,” 2023 IEEE 29th International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA), pp. 186–195, Aug. 2023, doi: 10.1109/rtcsa58653.2023.00030.

[6]. R. Endeley, T. Fleming, N. Jin, G. Fehringer, and S. Cammish, “A Smart Gateway Enabling OPC UA and DDS Interoperability,” 2019 IEEE SmartWorld, Ubiquitous Intelligence & Computing, Advanced & Trusted Computing, Scalable Computing & Communications, Cloud & Big Data Computing, Internet of People and Smart City Innovation (SmartWorld/SCALCOM/UIC/ATC/CBDCom/IOP/SCI), pp. 88–93, Aug. 2019, doi: 10.1109/smartworld-uic-atc-scalcom-iop-sci.2019.00058.

[7]. F. B. Cheikh, M. A. Mastouri, and S. Hasnaoui, “Implementing a Real-Time Middleware Based on DDS for the Cooperative Vehicle Infrastructure Systems,” 2010 6th International Conference on Wireless and Mobile Communications, pp. 492–497, Sep. 2010, doi: 10.1109/icwmc.2010.49.

[8]. L. Bertaux, A. Hakiri, S. Medjiah, P. Berthou, and S. Abdellatif, “A DDS/SDN Based Communication System for Efficient Support of Dynamic Distributed Real-Time Applications,” 2014 IEEE/ACM 18th International Symposium on Distributed Simulation and Real Time Applications, pp. 77–84, Oct. 2014, doi: 10.1109/ds-rt.2014.18.

[9]. H. A. Putra and D.-S. Kim, “Node discovery scheme of DDS for combat management system,” Computer Standards & Interfaces, vol. 37, pp. 20–28, Jun. 2014, doi: 10.1016/j.csi.2014.05.002.

[10]. A. Alaerjan, “Formalizing the semantics of DDS QOS policies for improved communications in distributed smart grid applications,” Electronics, vol. 12, no. 10, p. 2246, May 2023, doi: 10.3390/electronics12102246.

[11]. K. An, S. Pradhan, F. Caglar, and A. Gokhale, “A publish/subscribe middleware for dependable and real-time resource monitoring in the cloud,” SDMCMM ’12: Proceedings of the Workshop on Secure and Dependable Middleware for Cloud Monitoring and Management, pp. 1–6, Dec. 2012, doi: 10.1145/2405186.2405189.

[12]. C. O’Ryan, F. Kuhns, D. C. Schmidt, O. Othman, and J. Parsons, “The design and performance of a pluggable protocols framework for Real-Time Distributed Object Computing middleware,” in Lecture notes in computer science, 2000, pp. 372–395. doi: 10.1007/3-540-45559-0_19.

[13]. M. Carrascosa-Zamacois, G. Geraci, E. Knightly, and B. Bellalta, “Wi-Fi Multi-Link operation: An experimental study of latency and throughput,” IEEE/ACM Transactions on Networking, vol. 32, no. 1, pp. 308–322, Jun. 2023, doi: 10.1109/tnet.2023.3283154.

[14]. A. Saifullah, Y. Xu, C. Lu, and Y. Chen, “Real-Time Scheduling for WirelessHART Networks,” 2010 31st IEEE Real-Time Systems Symposium, Nov. 2010, doi: 10.1109/rtss.2010.41.

[15]. D. Aksoy and M. Franklin, “Scheduling for large-scale on-demand data broadcasting,” Proceedings. IEEE INFOCOM ’98, the Conference on Computer Communications. Seventeenth Annual Joint Conference of the IEEE Computer and Communications Societies. Gateway to the 21st Century (Cat. No.98, vol. 2, pp. 651–659, Nov. 2002, doi: 10.1109/infcom.1998.665086.

[16]. K. A. Hawick, P. D. Coddington, and H. A. James, "Distributed frameworks and parallel algorithms for processing large-scale geographic data," Parallel Computing, vol. 29, no. 10, pp. 1297–1333, Sep. 2003, doi: 10.1016/j.parco.2003.04.001.
[17]. A. Zervopoulos et al., "Wireless sensor network synchronization for precision agriculture applications," Agriculture, vol. 10, no. 3, p. 89, Mar. 2020, doi: 10.3390/agriculture10030089.
[18]. B. Almadani, M. N. Bajwa, S.-H. Yang, and A.-W. A. Saif, "Performance Evaluation of DDS-Based Middleware over Wireless Channel for Reconfigurable Manufacturing Systems," International Journal of Distributed Sensor Networks, vol. 11, no. 7, p. 863123, Jul. 2015 doi: 10.1155/2015/863123.
[19]. B. Li, Y. Zhu, X. Yao, C. Jiang, K. Lu, and Z. Sun, "Enabling deterministic transmission for DDS by leveraging IEEE 802.1Qbv time-sensitive networking," Computer Networks, vol. 261, p. 111128, Feb. 2025, doi: 10.1016/j.comnet.2025.111128.
[20]. K. An, S. Shekhar, F. Caglar, A. Gokhale, and S. Sastry, "A cloud middleware for assuring performance and high availability of soft real-time applications," Journal of Systems Architecture, vol. 60, no. 9, pp. 757–769, Feb. 2014, doi: 10.1016/j.sysarc.2014.01.009.

ISSN (Online): 3105-9082