The Three Dimensions of ROS 2 Middleware

Sanghoon Lee, Taehun Kim, Angelo Corsaro and Kyung-Joon Park

***Abstract*—ROS 2 (Robot Operating System 2) has emerged as the de facto standard for modern robot software development, with middleware implementations such as the Data Distribution Service (DDS) and Zenoh forming the core infrastructure for distributed robotic communication. Despite their architectural flexibility, these middleware systems exhibit structural limitations, particularly under dynamic and resource-constrained wireless environments. This paper presents a systematic survey of ROS 2 middleware and introduces a conceptual framework to examine its architectural limits through three structural dimensions required by distributed robotic systems, namely Space, Time, and State. We first provide a structured analysis of middleware architecture and operational dynamics, including discovery, data exchange, and state management mechanisms. Building on this foundation, we formalize Time as temporal predictability for control loops, Space as spatial abstraction from physical topology to enable modular deployment, and State as contextual continuity despite dynamic node participation and intermittent connectivity. Through a comprehensive review of existing implementations and prior studies, we organize middleware research according to the structural trade-offs that arise among these dimensions. Under constrained wireless conditions, spatial abstraction can obscure network variability and weaken temporal guarantees, while mechanisms that preserve state continuity introduce computational and network overhead that competes with time-critical communication. These interactions reveal structural trade-offs that characterize the practical limits of contemporary robot middleware. By synthesizing architectural patterns and identifying gaps in current modeling and analysis approaches, this survey outlines a principled research roadmap for robust and scalable robotic middleware architectures.**

***Index Terms*—Robot Operating System 2 (ROS 2), middleware, Data Distribution Service (DDS), Zenoh, Distributed robotic systems, Survey, Communication Architecture**

I. Introduction

Robot Operating System 2 (ROS 2) has emerged as the de facto standard for modern robot software development [1], [2]. The growing distribution and modularization of robotic systems has driven this transition, as perception, planning, and control components are deployed across multiple processes, devices, and network domains. As robotic platforms expand toward multi-robot coordination, edge-cloud integration, and wireless deployment, communication increasingly spans heterogeneous and resource-constrained environments [3]–[5]. Within ROS 2, the middleware layer abstracts and manages this complex communication through implementations such as Data Distribution Service (DDS) and Zenoh [6]. These middleware systems constitute the core infrastructure for discovery, data exchange, and state management, thereby playing a central architectural role in determining system scalability, responsiveness, and robustness [7], [8].

Sanghoon Lee, Taehun Kim, and Kyung-Joon Park are with the Daegu Gyeongbuk Institute of Science and Technology (DGIST), Daegu 42988, South Korea. (e-mail: leesh2913@dgist.ac.kr, taehun@dgist.ac.kr, kjp@dgist.ac.kr).

Angelo Corsaro is with Eclipse Zenoh. (e-mail:ac@zenoh.io).

Distributed robotic systems simultaneously require temporal predictability for control loops, spatial abstraction from physical topology to enable modular and scalable deployment, and sustained state continuity despite dynamic node participation and intermittent connectivity. When these requirements are exercised in real deployments, distributed systems frequently exhibit performance degradation [9], [10]. In wireless settings, bandwidth fluctuations and latency variability weaken temporal predictability [11]–[13]. In multi-node and multi-robot systems, dynamic topology changes delay or disrupt discovery and synchronization processes, undermining modular coordination across distributed components [14]. Furthermore, mechanisms intended to preserve state continuity, such as buffering, replication, and session maintenance, introduce additional computational and network overhead that, in long-running deployments, often results in congestion or session instability [15].

A substantial body of research on ROS 2 has concentrated on application utilization and framework development, as well as performance optimization within robotic systems, with particular emphasis on real-time scheduling [16]–[20], executor design [21]–[25], and latency control at the execution and transport layers [26], [27]. Quantitative surveys of ROS 2-specific publications indicate that only a very small fraction of the literature explicitly investigates middleware-layer design and analysis, accounting for approximately 3.5% of reported studies [13]. Although interest in ROS 2 middleware has grown in recent years, particularly in areas such as Quality of Service (QoS) policies, network behavior, and analytical modeling, it remains comparatively limited relative to the broader body of application-level and real-time research [2]. Furthermore, existing investigations typically proceed in isolation and lack a unifying architectural framework that systematically captures the cross-dimensional interactions among temporal predictability, spatial abstraction, and state continuity requirements [28], [29].

These observations reveal that current middleware architectures increasingly constrain the performance envelope of distributed robotic systems. As robotics advances toward vision-language-action integration, large-scale multi-robot coordination, and persistent autonomous operation, communication and state management can no longer be treated as secondary concerns [5], [11], [30]. The architectural properties of middleware directly influence system-level behavior under realistic deployment conditions, such as dynamic mesh networks and high-payload wireless scenarios. This survey therefore undertakes a systematic examination of middleware design to identify architectural gaps and establish a principled research roadmap for robust and scalable robotic middleware architectures.

The contributions of this paper are summarized as follows.

- We provide a structured examination of ROS 2 middleware by analyzing its architectural constructs and operational

dynamics, including discovery, data exchange, and state management mechanisms.
- Building on this analysis, we formalize three fundamental dimensions required by distributed robotic systems, namely *Space*, *Time*, and *State*, and articulate their architectural implications for middleware design.
- We systematically reinterpret existing middleware research through the lens of these dimensions, organizing it according to the structural trade-offs among temporal predictability, spatial abstraction, and state continuity.
- Finally, we synthesize the identified structural gaps and propose a research roadmap to guide the development of robust and scalable robotic middleware architectures.

The remainder of this paper is organized as follows. Section II provides background on ROS 2 and its middleware layer. Section III presents an architectural analysis of ROS 2 middleware constructs and their operational dynamics, including discovery, data exchange, and state management mechanisms. Section IV formalizes the three fundamental dimensions of robot middleware, namely *Space*, *Time*, and *State*, and discusses their realization in middleware implementations. Section V examines the structural trade-offs that arise among these dimensions and organizes related research within this framework. Section VI synthesizes the identified architectural gaps and outlines a research roadmap for robust and scalable robotic middleware design. Finally, Section VII concludes the paper.

II. Backgrounds

A. ROS 2

ROS 2 is an open-source software framework designed to support the development of distributed robotic systems. Unlike monolithic architectures, ROS 2 enables modular composition of perception, planning, and control components across multiple processes and networked devices [31], [32]. Its primary objective is to provide a standardized software infrastructure that ensures scalability across heterogeneous deployment environments. At its core, ROS 2 utilizes a common functional base through the ROS Client Library (RCL), which provides consistent semantics across language-specific implementations [33].

ROS 2 architects communication around the ROS Middleware (RMW) interface, which serves as an abstraction and translation layer between the RCL and the underlying middleware implementation. This design enables various external implementations, such as eProsima Fast DDS, Eclipse Cyclone DDS, or Zenoh, to integrate seamlessly beneath a uniform application programming interface [34], [35]. Unlike ROS 1, which utilized a bespoke and centrally managed protocol, ROS 2 decouples high-level framework logic from low-level mechanisms of discovery, matching, and transport [36]. ROS 2 translates application-level abstractions through the RMW interface into the primitives of the selected middleware implementation. This approach delegates connectivity and reliability management to the middleware while preserving a consistent programming model [37].

At the application level, users define communication policies such as QoS parameters and callback execution semantics within the ROS 2 programming model. The RMW layer propagates these policies, which the underlying middleware implementation then realizes [37]. The ROS 2 executor coordinates callback dispatch and intra-process scheduling, whereas the middleware handles the enforcement of communication reliability and transport semantics [38]. Consequently, temporal predictability and QoS behavior emerge from the interaction between application-level execution semantics and their realization within the middleware.

B. ROS 2 Middleware

Standard literature defines middleware as an abstraction layer positioned between the application layer and the underlying operating system (OS), which mediates resource access in an OS-independent manner and provides service coordination across distributed components [39]. Among various categories of middleware, communication middleware provides services such as addressing, message routing, reliability control, QoS enforcement, and data serialization, thereby shielding applications from low-level socket programming and transport-specific details [40]. In ROS 2, the middleware implementation corresponds specifically to a communication middleware engine. It abstracts QoS policies expressed through the RCL API and interacts with the ROS 2 executor through the RMW WaitSet mechanism, which enables coordinated event notification and data handling across distributed components [6].

To avoid ambiguity, we use the term RMW to collectively denote the RMW interface layer and its corresponding middleware-specific implementation. We do not interpret RMW as a passive data transport layer. Its internal architectural choices directly influence latency characteristics, discovery convergence, bandwidth utilization, and state persistence across distributed deployments. To understand how these architectural choices manifest in practice, we next examine the primary middleware implementations supported in ROS 2.

ROS 2 designates certain middleware implementations as Tier one, which indicates the highest level of support and compatibility in its ecosystem [41]. These include DDS-based implementations such as eProsima Fast DDS, Eclipse Cyclone DDS, and RTI Connext DDS, as well as Eclipse Zenoh. In this paper, we focus on DDS implementations and Zenoh, because they represent the primary communication backends in current ROS 2 deployments.

1) Data Distribution Service (DDS): DDS is a widely adopted communication middleware standardized by the Object Management Group (OMG) and originally developed for mission-critical applications in aerospace, defense, and autonomous systems [6]. DDS serves as the default middleware implementation in ROS 2 and operates based on a data-centric publish-subscribe paradigm in which information is exchanged through named topics. In DDS, each ROS 2 context typically creates a DomainParticipant to manage the relevant DDS entities, which establishes an entity-centric hierarchy. Explicit QoS policies shape communication behavior within this framework [42]. Through these mechanisms, DDS provides configurable parameters for reliability, durability, and deadline, which the RMW interface layer exposes to ROS 2 users. Of these, only the reliability policy enforces a delivery behavior, while others, such as deadline and transport_priority, signal or hint rather than enforce.

2) Zenoh: Zenoh is a communication middleware designed to enable efficient data exchange across distributed, heterogeneous, and resource-constrained environments [43]. It was developed to operate effectively under unreliable network conditions, including wireless and edge deployments [41], [44]. Beginning with the ROS 2 Kilted Kaiju release in 2025, the community officially designated Zenoh as a Tier one middleware implementation within the ROS 2 ecosystem. Architecturally, Zenoh employs a unified publish-subscribe and query protocol organized around a key/value data model, where hierarchical key expressions express interest and drive the resulting data flows. In Zenoh, the middleware operates through Zenoh sessions, where the specific mode, such as peer, router, and client, determines how they join the network. This design follows a session-centric approach that offers significant flexibility in connectivity and topology [45], [46]. In ROS 2 deployments, users commonly employ a Zenoh router, which facilitates discovery and data forwarding between sessions [47].

3) Specialized Middleware: In addition to DDS and Zenoh, the ROS 2 ecosystem includes other middleware-related technologies with specialized scopes. For example, iceoryx provides zero-copy shared memory transport to optimize intra-host data exchange [48], while micro-ROS extends ROS 2 to resource-constrained microcontroller environments with limited networking capabilities through an XRCE-DDS-based client-agent architecture [49]. As these technologies target specific deployment contexts rather than general-purpose distributed communication across networked hosts, this study focuses primarily on DDS and Zenoh.

III. ROS 2 Middleware Architecture and Dynamics

A. ROS 2 Middleware Architecture

At the application level, ROS 2 software is organized into packages as the fundamental deployment units [50]. Each package typically contains executable programs that instantiate one or more nodes at runtime to enable distributed computation. A node represents an independent execution entity within a running executable that encapsulates application logic and communicates through standardized communication abstractions [51]. Rather than interacting directly with network primitives, nodes create publishers, subscriptions, services, and actions through the RCL. The RMW interface layer then maps these abstractions into middleware-level communication constructs to enable typed data exchange [52], [53].

ROS 2 provides three communication abstractions at the RCL layer: Topics, Services, and Actions [54]. Topics implement an asynchronous publish-subscribe communication mechanism, where a publisher sends messages to one or more subscribers without waiting for a request-response exchange. [55]. Services implement synchronous request-response interactions, where a client sends a request and remains blocked until a service server returns a response [56]. Actions extend the request-response pattern, where a client sends a goal to an action server that manages execution, publishes feedback, and returns a result. This mechanism is internally implemented through a combination of topics and services [57]. All data types are defined in ROS 2 interface files, from which language-specific code is generated at build time and then used in the RCL layer [58] The RMW interface layer does not interpret these abstractions semantically, but maps each RCL entity to middleware-level entities, which operate within specific process and host boundaries [33].

To provide a unified view across middleware implementations, we abstract the ROS 2 communication architecture into five structural entities: Endpoint (E), Topic (T), Message (M), Process (P), and Host (H). Although DDS and Zenoh differ in their internal implementation, these entities can be consistently identified across deployment environments.

- **Endpoint (E):** An endpoint is the fundamental unit of data production or consumption in the middleware layer. At the RCL layer, endpoints correspond to publishers, subscriptions, service clients, and service servers, which the RMW interface layer maps to concrete middleware constructs such as DDS DataWriter/DataReader entities or Zenoh publishers, subscribers, queriers, and queryables. Through this mapping, endpoints function as event-generating entities within the middleware layer, triggering RCL callbacks upon entity readiness propagated via the WaitSet in the RMW layer and thereby enabling reactive execution in ROS 2 applications. Each endpoint is instantiated within a specific OS process and assigned a unique identifier within the middleware domain. The middleware forms logical associations between endpoints based on matching topic names and type definitions. Each endpoint operates under QoS policies that govern reliability, durability, and delivery semantics, and maintains internal state as required to realize these QoS guarantees.
- **Topic (T):** A topic is a logical communication channel identified by a name and an associated data type [59], explicitly formed at the RCL layer. The RMW layer maps each topic to middleware-specific constructs, such as topic names in DDS or hierarchical key expressions in Zenoh. Topics provide the naming and typing context required for endpoint matching across distributed contexts, thereby enabling communication between matching endpoints across process and host boundaries.
- **Message (M):** A message is a concrete data instance conforming to a type defined in ROS 2 interface. At the RCL layer, message structures are generated from interface definitions and instantiated by application logic. At the RMW interface, message instances are serialized into middleware-specific representations for transmission. In DDS, this representation corresponds to a serialized sample, whereas in Zenoh it corresponds to a value payload associated with a key expression. Messages are exchanged between matched endpoints within the scope defined by a topic and may be delivered across process and host boundaries via middleware-managed transport mechanisms. Endpoint-level QoS policies govern the lifetime and delivery behavior of message instances.
- **Process (P):** A process is an OS-level execution unit that serves as the direct counterpart to a participant: each process corresponds to one middleware participant, realized as a DDS DomainParticipant or a Zenoh session. Within a process, one or more ROS 2 nodes are instantiated and share a single ROS 2 context, which is mapped to a single participant. While

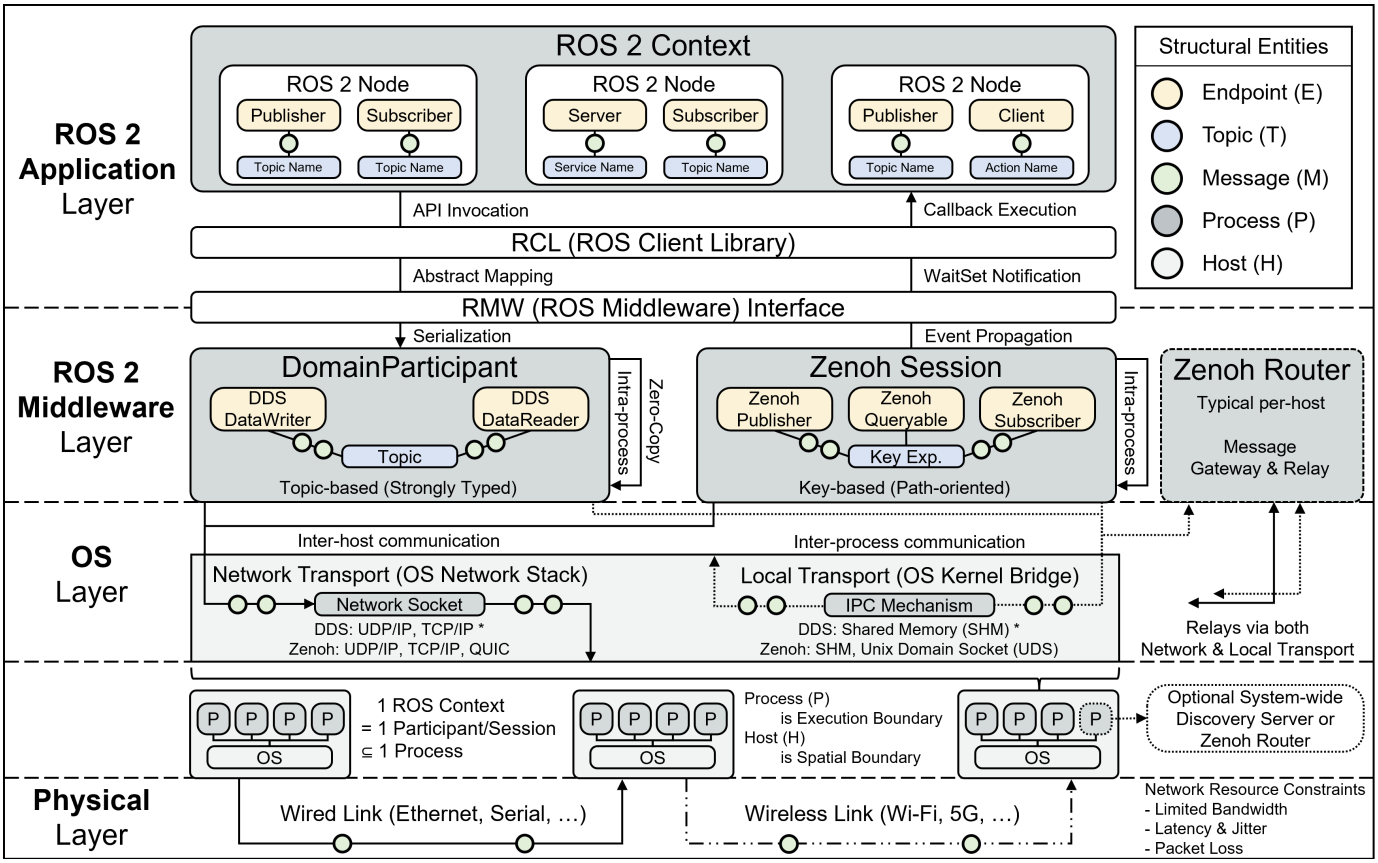

Fig. 1. Structural abstraction of the ROS 2 communication architecture. For DDS, TCP/IP and SHM (marked with *) are vendor-specific extensions.

most deployments run a single node per process, the node composition mechanism allows multiple nodes to be loaded into the same process, sharing a ROS 2 context and therefore a single participant. Each participant acts as the entry point through which nodes and their endpoints participate in middleware communication, managing the lifecycle of publishers, subscriptions, service clients, and service servers declared within it. Processes on the same host share hardware resources but are allocated distinct OS resources, including network ports and socket descriptors, thereby defining the execution boundary within which middleware participation is maintained and callbacks are scheduled. In advanced configurations, a process may maintain multiple contexts, each mapped to an independent participant, to support scenarios such as domain bridging or multi-domain operation.

- **Host (H):** A host is a distinct computing unit equipped with one or more network interfaces. It provides the physical execution environment within which one or more processes are deployed. Communication between processes on the same host may occur via shared memory or local transport mechanisms, bypassing network contention. In contrast, communication across hosts relies on network interfaces and shared communication media. Although a host is often associated with a single robot, complex robotic platforms commonly comprise multiple networked hosts interconnected through heterogeneous communication links, including Ethernet, wireless, and serial buses [60], [61]. The host, therefore, defines the spatial boundary across which middleware communication is routed and over which network-level resource constraints are manifested.

Middleware implementations can introduce additional components for routing and discovery. For example, Zenoh may employ one or more Zenoh routers, which can be deployed per host or as shared infrastructure within a local network to facilitate discovery and data forwarding. Similarly, certain DDS deployments may utilize discovery servers or routing services, instantiated as dedicated processes on specific hosts [62], [63]. These components introduce intermediary processes that alter the communication topology and influence network behavior, yet they do not redefine the fundamental structural entities introduced above. Instead, they mediate how processes and hosts are interconnected within a given deployment.

Fig. 1 illustrates the unified structural abstraction of the ROS 2 communication architecture, organized into a four-tier hierarchy of application, middleware, OS, and physical layers. Within the application layer, nodes instantiate endpoints that exchange messages over topics. These entities are hosted within a ROS 2 context mapped to a single participant in the middleware layer. Two fundamental boundaries govern the system. The process defines the execution boundary within which participation is maintained and callbacks are scheduled, while the host defines the spatial boundary across which processes communicate via local or network transport, including Ethernet, wireless, and serial buses. Together, these structural entities provide a consistent foundation for connecting heterogeneous distributed robotic systems across both high-level abstractions and physical hardware constraints.

B. Middleware Operational Dynamics

In practice, middleware communication is governed by three interrelated operational dynamics, namely discovery, data exchange, and state management, that operate over these structural entities during execution. Discovery mechanisms determine how participants and endpoints are discovered and logically interconnected across processes and hosts. Data exchange mechanisms regulate how messages are serialized, transmitted, and received under reliability and QoS constraints. State management mechanisms maintain and evolve the internal middleware state associated with these entities to preserve continuity under dynamic participation and long-running deployment. These dynamics are continuously active rather than sequential stages, and their interactions shape the performance, scalability, and robustness of distributed robotic systems. The following subsections examine how DDS and Zenoh realize these mechanisms and analyze the structural implications of their respective design choices.

1) Discovery: Discovery mechanisms establish the initial and ongoing spatial structure of a distributed ROS 2 system. Through discovery, middleware participants become aware of one another, exchange metadata, and construct the logical graph over which communication subsequently occurs. Unlike a one-time initialization step, discovery is continuously active, reacting to dynamic participation, restarts, and network reconfigurations [64]. The discovery process comprises (i) participant discovery and (ii) endpoint matching.

(i) Participant Discovery: Participant discovery refers to the mechanism by which DDS DomainParticipants or Zenoh sessions become mutually aware of one another. In DDS, participant discovery is, by default, performed using the Real-Time Publish-Subscribe (RTPS) protocol [65]. More specifically, DDS employs the Simple Participant Discovery Protocol (SPDP), a distributed peer-to-peer (P2P) mechanism in which DomainParticipants periodically announce their presence within a domain. These announcements allow other participants to detect newly joined participants and establish mutual awareness. By default, this discovery behavior relies on multicast communication within the local network scope, enabling direct P2P visibility without requiring centralized coordination [66].

In Zenoh, participant discovery can be implemented through multiple independently configurable mechanisms. One mechanism is multicast scouting, in which a session announces its presence on the local network so that nearby sessions can detect it directly. Another mechanism is gossip-based discovery, in which a session propagates discovery information about already known sessions and routers to other connected sessions [67]. These mechanisms are not inherently coupled and may be enabled or disabled independently. In the default configuration of a Zenoh-based RMW interface, multicast scouting is disabled while gossip-based discovery is enabled. In this setting, Zenoh routers propagate discovery information received from locally attached sessions and redistribute it to other routers and sessions through gossip messages [68].

(ii) Endpoint Matching: Following participant discovery, middleware systems advertise endpoint metadata and determine compatibility within the established communication context. In DDS, endpoint matching is performed through the Simple Endpoint Discovery Protocol (SEDP), which disseminates endpoint metadata via built-in discovery topics [69]. These topics convey information such as topic names, data types, and QoS policies. Matching occurs when DomainParticipants detect compatible name–type pairs and verify that their QoS policies satisfy the DDS compatibility rules. Only after this compatibility check do the corresponding entities establish a logical association.

In Zenoh, endpoint advertisement is performed by announcing resource key expressions and subscription interests to directly connected sessions and routers. Matching occurs when a published key expression satisfies a corresponding subscription interest [70]. Unlike DDS, Zenoh does not perform endpoint-level QoS compatibility checks during the matching phase; instead, QoS settings influence behavior at the data exchange stage. In router-mediated configurations, a router collects the key expressions and subscription interests of its connected sessions and shares them with other routers or with sessions not directly connected to one another. Based on this aggregated state, the router constructs and maintains visibility relationships among its connected sessions, thereby enabling endpoint matching among otherwise isolated sessions.

2) Data Exchange: Data exchange mechanisms determine how message instances are represented, routed, and delivered across processes or hosts under the underlying transports, routing topology, and required QoS. Data exchange governs the runtime behavior of communication after endpoint matching, shaping the latency, jitter, and loss characteristics observed by ROS 2 applications. In ROS 2, these behaviors emerge from the interaction among (i) serialization and encoding, (ii) transport mechanisms, and (iii) reliability and delivery control.

(i) Serialization and Encoding: Before transmission, ROS 2 message instances are serialized into middleware-specific wire representations. ROS 2 adopts Common Data Representation (CDR) [71] as its standard serialization format at the RCL layer, applied consistently regardless of the underlying middleware implementation.

In DDS, the serialized output constitutes a data sample, which serves as the atomic unit for transmission and caching [72]. DDS may attach inline metadata that carries RTPS-level information and selected QoS-related parameters required for interpreting or managing the sample at the receiver side. The resulting representation combines the application payload and protocol metadata into a form suitable for RTPS-level encapsulation and transport. In Zenoh, the serialized output is conveyed as a byte payload associated with a key expression [67]. The Zenoh wire protocol encapsulates the payload with protocol-level metadata, including the wire key expression, encoding information, and optional timestamp. Application-level metadata may additionally be attached on a per-message basis as key-value pairs via the attachment API, without requiring re-serialization of the payload.

(ii) Transport Mechanisms: Following serialization, the middleware selects transport mechanisms to deliver payloads across process and host boundaries. Both DDS and Zenoh

support shared-memory transport for intra-host communication, avoiding kernel-network overhead for co-located processes. DDS relies on the RTPS protocol, which defines UDP-based unicast and multicast as its primary transport. Modern DDS implementations support TCP to accommodate network environments where UDP multicast is restricted or unreliable.

Zenoh employs a pluggable transport layer that supports multiple protocols, including TCP, UDP, and QUIC [73]. Communication can proceed over direct P2P links or through router-mediated forwarding, depending on deployment configuration. Zenoh routers relay data based on link-state information and network visibility, enabling operation across heterogeneous network topologies [74].

(iii) Reliability and Delivery Control: To ensure delivery semantics under loss and contention, middleware can maintain protocol state and apply reliability, congestion, and flow control mechanisms. In DDS, reliable communication is typically realized through stateful RTPS-level mechanisms, such as AckNack exchanges, periodic Heartbeats, and retransmission windows maintained per endpoint [75]. These mechanisms operate in conjunction with reliability and history QoS policies, which define send-queue depth and the persistence of unacknowledged samples. Per-endpoint cache limits and timing parameters that bound Heartbeat and retransmission rates further govern congestion and flow behavior.

In Zenoh, each session provides a best-effort channel and a reliable channel by default, with the reliable channel maintaining sequence numbers and acknowledgment state for loss detection and retransmission [76]. When connection-oriented transports such as TCP or QUIC are used, their native ordering and loss recovery supplement these session-level guarantees [4]. Publishers control congestion behavior by selecting a per-message drop or block policy applied along the entire routing path, including Zenoh routers [76].

3) State Management: State management mechanisms determine how middleware maintains communication state and preserves continuity across process and host boundaries under dynamic conditions and intermittent connectivity. State management governs what information is tracked, cached, retained, and recovered. In ROS 2, these behaviors emerge from the interaction among three aspects: (i) liveliness maintenance, (ii) history restoration, and (iii) resource control.

(i) Liveliness Maintenance: Liveliness maintenance determines how middleware monitors the presence and health of contexts across processes and hosts. In DDS, liveliness is maintained through Heartbeat and lease mechanisms at both the participant and endpoint levels. These mechanisms are continuously active, with periodic liveliness assertions refreshing leases while missed assertions trigger liveliness-loss detection. This creates a decentralized monitoring structure in which each DomainParticipant observes the liveliness of others.

In Zenoh, liveliness is governed by a session lease and keep-alive mechanism [70]. Any data or protocol message exchanged on a session refreshes the lease, and keep-alive messages are sent only when the session would otherwise be idle. If the lease expires, the session is closed and the associated resources are removed. Zenoh also provides liveliness tokens for application-level monitoring [77]. These are explicitly declared resources whose presence is observed by peers, and whose loss is signaled when the declaring session is disconnected or the token is undeclared [78]. This design provides session-centric monitoring and avoids continuous P2P polling.

(ii) History Restoration: Middleware governs the synchronization of state for entities that rejoin after disruption or appear after messages have already been produced. In DDS, this behavior is localized within the HistoryCache of an endpoint [79]. Under durability QoS settings, a DataWriter replays messages from its runtime cache to newly matched DataReaders [80]. Consequently, in standard ROS 2 deployments, history restoration is limited to data retained in active memory and does not persist across entity restarts without specialized persistent storage services [81].

Zenoh decouples history from the producing endpoint by introducing independent storage managers [82]. These managers act as dedicated peers that store values for specific key expressions, enabling late-joining endpoints to pull historical state via query mechanisms even after the original endpoint is no longer available [67]. This design complements endpoint-local buffering by providing a process-level storage service.

(iii) Resource Control: Resource control mechanisms define the constraints and policies that prevent unbounded memory consumption and buffer exhaustion within the middleware. In DDS, these constraints are enforced through the resource limits QoS policy, which bounds HistoryCaches at the individual endpoint level [83]. By defining parameters such as the maximum number of stored samples per endpoint, DDS provides explicit control over the memory allocated to each entity. However, the resulting behavior depends on the preconfigured limits, and insufficient limits may cause message drops or blocking.

Zenoh manages resources through transmission queues configurable on sessions and routers [84]. Users can explicitly control queue sizes and the memory allocated to each queue. This makes resource management more path-based than endpoint-based, because available buffering depends on configured queues along the routing path [85]. Compared with DDS, Zenoh distributes resource control across sessions and routers rather than enforcing it at the individual endpoint level.

Fig. 2 illustrates a visual synthesis of the three primary operational dynamics in ROS 2 middleware: discovery, data exchange, and state management. These dynamics function as interdependent loops to collectively govern the distributed system. The spatial structure established by discovery directly shapes the routing paths available for data exchange, while the internal state managed by the middleware ensures the continuity of these interactions. This tight coupling implies that any modification to one dynamic inevitably propagates performance implications to the temporal and state-related aspects of the communication stack.

To systematically evaluate these inherent trade-offs, Section IV formalizes the three structural dimensions required by ROS 2 middleware: *Space*, *Time*, and *State*. We transition to a unified framework that defines the objectives pursued by ROS 2 middleware within its operational dynamics, grounding each dimension in the previously defined structural entities.

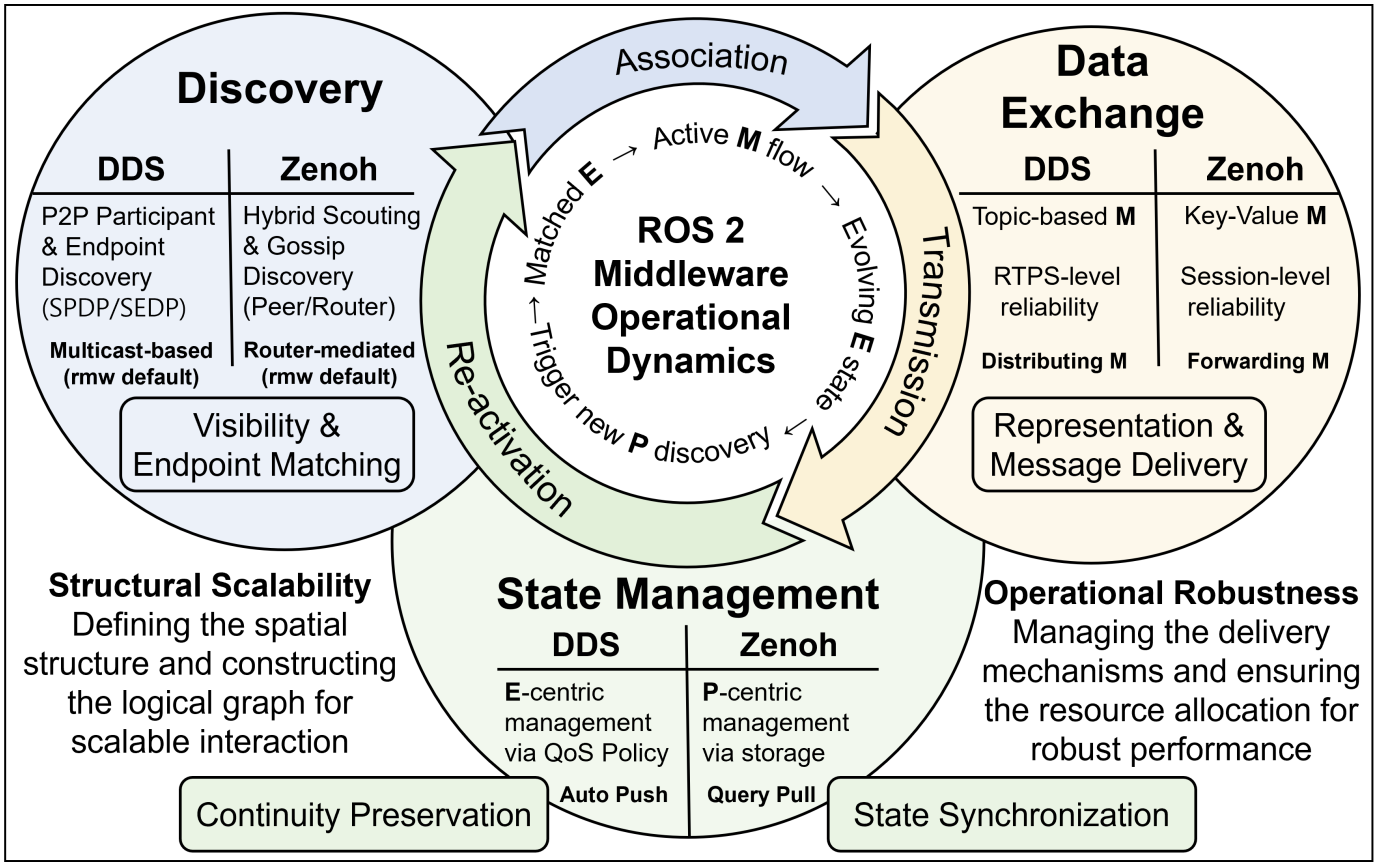

Fig. 2. ROS 2 Middleware Operational Dynamics.

IV. Three Dimensions of Robot Middleware

This section formalizes the three structural dimensions essential to distributed robotic systems: *Space*, *Time*, and *State*. These dimensions are not design choices unique to a particular middleware; they are universal structural requirements imposed by the nature of distributed robotic systems. ROS 2 middleware must provide (i) location transparency so that applications can interact without knowledge of physical placement, (ii) temporal predictability so that time-sensitive robotic tasks receive consistent and bounded communication behavior, and (iii) contextual continuity so that system state is preserved across dynamic node participation. DDS and Zenoh pursue identical objectives along each of these dimensions, even though their mechanisms for realizing them differ structurally. This section formalizes each dimension as a shared objective and establishes how it maps to the operational dynamics and structural entities defined in the preceding section.

Table I summarizes this mapping by associating each dimension with a key question that captures its operational concern and a primary objective that defines the goal middleware must pursue to satisfy it. The key question for each dimension is not merely descriptive but diagnostic: it identifies the class of failure that arises when the dimension is insufficiently realized. When *Space* is inadequately supported, the system cannot answer where an entity is located, and modular deployment breaks down. When *Time* is inadequately supported, the system cannot guarantee when a message is delivered, and control loops become unstable. When *State* is inadequately supported, the system cannot preserve how state is maintained across participation changes, and contextual continuity is lost. By mapping these dimensions to specific key questions and primary objectives, we establish a unified framework for evaluating how middleware supports the complex requirements of modern robotics.

TABLE I
Structural Dimensions of Robot Middleware

Dimension	Key Question	Primary Objective
Space	Where is the entity located?	Location Transparency
Time	When is the message delivered?	Temporal Predictability
State	How is state preserved?	Contextual Continuity

A. Space: Location Transparency

Space denotes the structural objective of maintaining a coherent logical communication structure that remains independent of the underlying network and deployment topology. The core requirement is location transparency, meaning that applications must be able to express interaction through data identifiers and semantic roles rather than physical network addresses or fixed placements. Without this property, system components cannot be composed, relocated, or scaled across processes and hosts, as doing so would require rewriting the application-level logic. This paper treats location transparency at the architectural level, requiring the middleware to sustain a coherent communication structure without operator-specified routing, intermediary configuration, or protocol bridges.

Both DDS and Zenoh pursue this objective through their discovery mechanisms, which transform entity participation events into logical visibility relationships. In each case, the middleware resolves the question of where an entity is located into an answer that is invisible to the application: a topic subscription matches a publisher not because the application is aware of its address, but because the middleware has constructed and maintained the appropriate logical association. The specific discovery mechanisms used, such as multicast communication, router-mediated scouting, represent distinct implementations of the same spatial abstraction objective. Consequently, the discovery scope, routing topology, and endpoint matching policies determine which remote entities are visible and reachable from the application's perspective.

B. Time: Temporal Predictability

Time denotes the structural objective of ensuring that middleware communication behavior satisfies the temporal requirements of distributed robotic tasks. The core requirement is temporal predictability, meaning that the middleware must regulate when messages are delivered to provide predictable and bounded latency under varying network conditions. Without this property, time-sensitive task loops cannot achieve stable operation across process and host boundaries.

Both DDS and Zenoh pursue this objective through their data exchange mechanisms, which govern how messages are serialized, transported, and delivered. In each case, the middleware must ensure that the application receives data within a useful time window while internally handling the complexity of transport heterogeneity and congestion. The specific transport layers and queue management strategies represent distinct implementations of the same temporal objective. These mechanisms also differ in how they treat message priority, with the DDS standard defining the transport_priority policy as a hint whose realization is left to each implementation [86], and Zenoh providing priority-driven delivery as part of its data exchange path. Any modification to these mechanisms, even if it does not affect the network topology, shapes the temporal envelope available to the robotic application.

C. State: Contextual Continuity

State denotes the structural objective of preserving state coherence across the system lifecycle despite dynamic participation and intermittent connectivity of entities. The core requirement is contextual continuity, meaning that the middleware must track, retain, and recover internal information so that late-joining or recovering components can synchronize their state without disrupting the application context. Without this property, distributed robotic systems cannot sustain consistent operation across node restarts, network disturbances, or prolonged deployments.

Both DDS and Zenoh pursue this objective through their state management mechanisms, which maintain the internal state associated with endpoints, participants, and their routing paths. In each case, the middleware must address the question of how state is preserved across boundaries where continuous connectivity can not be assumed. State management involves a fundamental trade-off between continuity and resource utilization. Mechanisms that extend the temporal reach of retained data inevitably consume memory and introduce synchronization overhead. The two designs differ in where this cost is located, with DDS placing it in HistoryCaches bound to the producing endpoint, and Zenoh assigning it to separate storage managers that need not be bound to the application. Bounding these costs through resource control policies necessarily limits the degree of continuity available to rejoining components.

D. Structural Trade-offs of Middleware Design

The three structural dimensions, *Space*, *Time*, and *State*, do not operate as independent design concerns. Each dimension is realized through operational dynamics that share the same execution and spatial boundaries, namely processes and hosts. As a result, the mechanisms that serve one dimension inevitably draw upon resources and infrastructure that are also required by the others. This shared dependency creates structural coupling among the dimensions, meaning that optimizing one dimension under constrained conditions imposes pressure on the remaining two.

The first coupling connects *Space* and *Time*. Location transparency relies on discovery mechanisms that continuously maintain logical visibility across dynamic participants. However, this abstraction conceals the physical network state that temporal performance depends upon, and the concealment itself becomes a source of degradation. In wireless or bandwidth-constrained environments, this degradation is amplified, because the overhead necessary to sustain a coherent spatial structure competes directly with the temporal budget available for latency-sensitive control and sensing communication, and the middleware possesses no architectural mechanism to detect or interrupt the escalation.

The second coupling connects *State* and *Time*. Contextual continuity requires liveliness maintenance, history restoration, and resource control that generate persistent background traffic and buffer overhead. In real-time deployments, these mechanisms do not merely consume residual capacity. They interact with the data exchange path through shared queues, lock-contended dispatch threads, and retransmission pathways. When these dynamics are active simultaneously, latency increases and jitter becomes less predictable, directly compromising the temporal predictability.

The third coupling connects *Space* and *State*. As the spatial scale of a deployment grows, the quantity of metadata that must be retained and propagated to maintain location transparency increases. Node anonymity withholds the physical identity information required for lifecycle state tracking, and path-blind forwarding withholds the network condition information required to prevent buffer saturation. Architectural responses to this pressure, such as introducing router intermediaries or centralized discovery servers, reduce the per-participant state cost but concentrate the system's visibility graph at a single point of failure for external connectivity.

Taken together, these three couplings reveal structural trade-offs inherent to ROS 2 middleware. Because the mechanisms required by each dimension compete for the same bounded

execution capacity, network bandwidth, and buffer memory, fully satisfying *Space*, *Time*, and *State* simultaneously under constrained deployment conditions remains structurally constrained under shared resource bounds. Any middleware design must therefore navigate the trade-off surface defined by these three dimensions, accepting partial compromises in at least one dimension to preserve the requirements of the others.

These structural trade-offs motivate the systematic review presented in the remainder of this paper. Section V maps 94 middleware-focused studies on ROS 2 middleware onto the trade-off space defined by the three dimensions. Rather than categorizing these works by implementation details or individual performance metrics, we organize them according to the primary dimensional conflict each study addresses and the dimension it prioritizes for optimization. This classification reveals the structural boundaries that contemporary middleware research has encountered and exposes the architectural gaps that remain unresolved in the design space of distributed robotic middleware.

V. Dimensional Conflict Analysis

The three dimensions in Section IV, namely *Space*, *Time*, and *State*, are not independent axes but represent structurally coupled mechanisms that inherently compete for shared computational and communication resources. This coupling produces three structural conflict pairs. First, spatial abstraction imposes unavoidable costs on temporal predictability. Second, state continuity mechanisms contend for the same resources required by time-critical transmissions. Third, the anonymity and full-mesh visibility that location transparency requires withhold the identity information that state continuity demands. Crucially, these conflicts do not arise within a single layer but follow escalation patterns across the system.

In this section, we map 94 middleware-focused studies onto the three conflict pairs identified above. We organize them according to the primary dimensional conflict each study addresses and the structural mechanism through which that conflict manifests, rather than cataloging these works by implementation detail or individual performance metric. For each conflict pair, we examine the underlying architectural assumptions that give rise to the tension, trace how proposed approaches attempt to relieve it, and identify the structural commonality that recurs across otherwise diverse approaches.

A. Space–Time Conflict

Spatial abstraction is the cornerstone of ROS 2, providing location transparency that decouples applications from their physical deployment. However, maintaining this abstraction imposes an unavoidable structural cost on temporal performance. This conflict is not confined to a single architectural layer; rather, it forms an escalation structure that originates within the middleware's internal logic and propagates toward the physical network as the scope of spatial independence widens. This subsection examines this escalation through four manifestations: serialization overhead at the middleware boundary, latency amplification induced by concealed physical Maximum Transmission Unit (MTU) limits, resource redundancy caused by location-agnostic delivery, and jitter amplification by physical multicast.

1) Serialization Overhead: Communication across heterogeneous hosts requires the middleware to transform application messages into a standardized wire format that is independent of programming language, endianness, and word size, so that data can traverse process and host boundaries [87]. Serialization is therefore intrinsic to communication that crosses host boundaries, required whether or not location transparency is sought. The Space–Time conflict arises because location transparency renders serialization non-elidable even when physical co-location would permit an intra-host non-serialized path.

Under large-payload conditions, this structural cost amplifies and propagates into the broader communication pipeline. Serialization of high-volume messages directly translates into CPU and memory pressure, as each transmission requires allocation, encoding, and subsequent decoding of the full payload. Measurements indicate that, under payloads exceeding a few hundred kilobytes, the dominant fraction of end-to-end communication latency originates in the serialization and memory-handling stages of the DDS, resulting in measurable CPU saturation [88]. Concurrently, the auxiliary data accompanying packet encapsulation consumes a disproportionately large share of total execution time and available bandwidth, independent of whether the application payload itself is small or large [87]. The internal queues maintained to relay messages between the logical and physical layers become saturated when transmission volume exceeds their service rate, blocking the publisher write process and introducing queuing latency into the delivery path [89].

The problem escalates structurally as endpoint counts increase. Analytical modeling of DDS internal queue-processing policies demonstrates that buffer saturation produces compounding delays, where thread batch-processing stalls and queue contention interact, causing latency to grow nonlinearly with load [87]. When multiple publishers simultaneously inject large payloads into shared internal queues, the per-message serialization cost accumulates into sustained throughput degradation, propagating backpressure upstream toward the application callback layer. Under these conditions, the communication pipeline degenerates from a bounded-latency channel into a congestion-driven bottleneck, undermining the temporal predictability required by closed-loop robotic tasks.

Several approaches have been proposed to reduce this overhead. Alternative message formats such as ROS-SF preserve the C++ struct memory layout to eliminate encoding and decoding without requiring source modification [90]. Zenoh-based architectures such as Meta-ROS reduce serialization overhead by optimizing the encoding pipeline under high data-rate conditions [91]. Partial serialization schemes such as TZC decompose messages into a control segment transmitted via socket and a bulk-data segment placed in shared memory, avoiding full-payload copying [92]. Zero-copy mechanisms such as ROS 2 Agnocast eliminate serialization entirely by mapping the publisher heap into shared memory and employing dedicated smart pointers, supporting even unsized message types [93]. At the framework level, Flexbot bypasses the DDS middleware for intra-host communication entirely, substituting native FIFO (First In, First Out) structures and pointer passing to remove serialization, copying, and fragmentation costs [15]. More fundamentally, the

ROS 2 RMW abstraction inherits the strongly typed DataWriter and DataReader model from DDS, which structurally forecloses serialization elision. Zenoh performs no serialization when transmitting user data placed in Zenoh-managed shared memory, yet this path is unreachable through the ROS 2 API because the API always serializes user data [47].

Each of these mitigations achieves cost reduction by reintroducing the locality awareness that spatial abstraction withholds. Alternative encodings that deviate from canonical CDR restrict interoperability across heterogeneous nodes, narrowing the scope of cross-host communication [94]. Zero-copy and pointer-passing approaches explicitly exploit the physical co-location of endpoints within the same process or host, making location transparency conditional on deployment topology. Intra-process optimizations eliminate boundary crossings by design but become inapplicable the moment communicating endpoints are distributed across hosts. Every proposed mitigation thus converges on the same structural outcome. Avoiding serialization cost requires partially relinquishing spatial abstraction.

2) Payload Fragmentation: The topic-based publish-subscribe model of ROS 2 is designed to completely conceal the transmission constraints of the physical network from the application layer. From the application's perspective, each message constitutes a single logical unit: the middleware accepts an arbitrarily sized payload and assumes responsibility for delivering it to all matched subscribers, irrespective of the physical transport characteristics that lie between publisher and subscriber [6]. This design is a direct expression of location transparency; the application is liberated from any knowledge of the MTU constraint that governs the largest packet a given network link can carry [88]. The abstraction is structurally sound in its intent, but it introduces a hidden vulnerability. Large payloads such as camera images and LiDAR point clouds, which are common in robotic perception pipelines, are silently decomposed at the physical layer into hundreds of fragments of approximately 1400 bytes each [95], without any visibility of this decomposition being surfaced to the middleware.

The consequence of this concealment is that the reliability of the entire logical message becomes contingent on the successful delivery of every individual fragment. Under nominal network conditions this dependency is largely invisible, but in wireless or resource-constrained deployments, physical-layer packet loss exposes the fragmentation structure that spatial abstraction had concealed. Experimental evaluation under simulated wireless conditions confirms that the retransmission process under a reliable QoS profile produces severe latency spikes, and that repackaging lost data into large retransmission units further degrades packet loss rates [96]. Quantitative results demonstrate that hierarchical fragmentation amplifies individual packet errors into message-level latency inflation, as a single lost fragment prevents receiver-side reassembly and forces retransmission of the affected fragment window [97].

Experiments demonstrate that even a moderate packet error rate on a wireless channel is sufficient to invalidate entire multi-fragmented messages, because the loss of a single MAC-layer frame renders the full reassembly attempt unrecoverable [98]. If the reassembly timer maintained by the OS kernel expires before all fragments arrive, the partially assembled datagram is discarded and all successfully delivered fragments are wasted. Because the middleware operates above the IP fragmentation layer, it remains unaware of a message's existence until all fragments have been reassembled into a complete datagram. As a result, the middleware repeatedly requests retransmission of the entire message rather than targeting only the missing fragments, generating redundant traffic [99].

Several approaches have been proposed to mitigate fragmentation-induced latency. Node composition and zero-copy APIs eliminate serialization and UDP fragmentation overhead entirely for intra-host communication paths [100]. Shared-memory pointer-passing architectures such as CyberRT resolve large-payload fragmentation by bypassing UDP transport altogether [101]. Configuring the maximum message size to match the network MTU boundary suppresses uncontrolled IP-layer fragmentation and constrains the retransmission burden triggered by individual packet loss [102]. Application-layer fragmentation protocols employing bitmap-based acknowledgment decompose large samples into independently retransmissible fragments, confining the cost of individual fragment loss to only the affected unit rather than the full message [98]. Subscriber-selective transfer mechanisms further reduce exposure to fragment loss by allowing subscribers to request only the message portions relevant to their processing context [103]. Transport-level alternatives such as TCP and QUIC, evaluated in automotive middleware comparisons, absorb fragment loss through connection-oriented recovery but introduce connection management overhead and head-of-line blocking under congestion [104].

Each of these mitigation strategies achieves its temporal benefit by partially dismantling the physical-layer independence that spatial abstraction provides. Node composition and shared-memory approaches eliminate fragmentation by exploiting physical co-location, making them inapplicable to the distributed, multi-host deployments [6]. Fragment-aware middleware and MTU-aligned configurations must explicitly recognize the boundaries of the underlying network, which means that transport characteristics originally abstracted away must be reintroduced into the middleware's operational logic. Application-layer fragmentation schemes require the publisher to reason about physical link properties and message partitioning structure, directly violating the principle that communicating endpoints remain agnostic to deployment topology [103]. TCP-based fallback constrains transport behavior to a unicast client–server connectivity model that is structurally incompatible with the decoupled multicast publish-subscribe paradigm [60]. The structural conclusion is therefore consistent across all approaches: reducing the temporal cost of fragmentation requires the middleware to reintroduce physical-layer awareness into its forwarding path.

3) Intra-Host Redundancy: The logical endpoint abstraction that underlies location transparency in ROS 2 middleware manages each subscriber as an independent logical entity, irrespective of where that subscriber is physically deployed. This design guarantees uniform communication semantics across all endpoints: a subscriber running on a remote host and

a subscriber running on the same host as the publisher receive identical treatment at the middleware layer [105]. Any differentiation based on physical co-location would require the system to expose deployment topology to the middleware layer, violating the independence that spatial abstraction is designed to provide. Absent explicit co-location awareness, the logical endpoint abstraction has no basis for recognizing that multiple subscribers on the same host could share a single physical copy of a message. It instead treats each subscriber as an isolated destination and issues a separate, independent transmission for each, with the number of message copies produced on a single host scaling linearly with the number of locally deployed subscribers [106].

Under inter-host communication, each transmission requires kernel-level copies through the socket buffers, repeated once per subscriber regardless of physical proximity between subscribers. Experiments confirm that unicast-based point-to-point distribution consumes CPU and memory resources that scale linearly with endpoint count, a relationship formally established through complexity analysis [107]. In unicast configurations, publisher-side throughput decreases proportionally as subscriber count grows, because each additional subscriber imposes an independent transmission workload on the same fixed send queue [108]. Even within a single process, the absence of strict smart-pointer optimization causes a distinct copy of the message to be instantiated per subscriber, confirming that the redundancy originates at the logical separation model rather than at the network layer [109]. Across multiple DDS implementations, results show systematically that per-subscriber latency increases by measurable increments with each additional subscriber under one-to-N configurations, a pattern that persists across implementations and payload sizes [110], [111].

As subscriber count increases, the cumulative overhead of redundant intra-host delivery escalates from a localized resource pressure into a system-wide timing constraint. Benchmarks of standard unicast delivery confirm that publisher-side Ethernet bandwidth utilization doubles under two-subscriber configurations and that latency grows proportionally thereafter, with throughput falling inversely with subscriber count [112]. In many-to-one architectures, once node count exceeds a threshold, receiver-side buffer overload drives message update failure rates toward saturation and causes average data age to increase sharply [113]. Analytical modeling of the internal HistoryCache buffers and publisher dispatch thread demonstrates that under high-frequency publication, message accumulation in these queues produces worst-case communication delays that grow with publication rate and subscriber count [87]. In scaled multi-robot deployments, the OS scheduling overhead introduced by independently executing processes further compounds this pressure, consuming over 1400% of available CPU resources under socket-based baseline configurations [114] and producing message loss and latency attributable to the redundant dispatch burden [115].

Several approaches have been proposed to reduce the overhead of intra-host redundancy. Intra-process node composition eliminates serialization and copy overhead for co-located endpoints [100], a principle extended to multi-process environments by zero-copy mechanisms such as Agnocast [93]. Shared memory transport replaces N independent socket transmissions with a single write, with iceoryx-backed deployments demonstrating near-constant latency regardless of message size [116]. DDS-based deployments address the same problem through shared-memory transport and intra-process communication optimizations natively supported within Fast DDS and Cyclone DDS [116], [117]. The Zenoh RMW similarly delivers a single copy to local subscribers through shared-memory transport [47].

Each of these solutions achieves its efficiency gain by introducing explicit physical co-location awareness into the middleware's forwarding logic, and in doing so, partially relinquishes the location transparency that motivates the logical endpoint abstraction. Shared memory transport requires the middleware to detect whether communicating endpoints reside on the same host and to apply a differentiated delivery path based on that detection, violating uniform communication semantics internally even where the application interface appears unchanged [117]. Intra-process communication concentrates endpoints within a shared process boundary, so that a failure in one node propagates to all co-located nodes, a consequence that is architecturally unacceptable in safety-critical deployments such as autonomous driving [93]. The performance gains are therefore achievable only within deployment configurations explicitly constrained by physical placement, confirming that logical endpoint independence and the elimination of redundant physical delivery have not been simultaneously achieved within the current middleware design space [89].

4) Inter-Host Redundancy: Inter-host communication faces a structural redundancy problem when a single publisher delivers a message to multiple subscribers across host boundaries. Under naive unicast delivery, the publisher issues separate transmissions per subscriber, so network-level transmission cost scales linearly with the number of remote subscribers. ROS 2 middleware addresses this through two mechanisms at different layers, namely multicast at the network layer and router-mediated fan-out at the middleware layer [108]. The OMG DDS specification relies on multicast as the default transport for discovery traffic within a local network domain [118]. Multicast is expected to provide a stable and predictable delivery path regardless of subscriber count, as the network infrastructure absorbs the replication burden without imposing additional cost on the middleware layer.

The structural vulnerability of this assumption is that packet replication under multicast is not performed at the logical layer by the middleware but at the physical layer by the network switch. When a multicast packet arrives at a switch, the switch must inspect its group membership state and produce one copy of the packet per outgoing port associated with that multicast group, a process that consumes switch-internal forwarding resources proportional to both the number of destination ports and the rate at which multicast packets arrive. IP multicast forces routers to maintain per-group state that varies over time, imposing scalability and reliability penalties that prevent the mechanism from scaling to large numbers of groups [119]. The middleware has no visibility into this threshold, as the physical replication constraint is precisely the kind of infrastructure

detail that spatial abstraction is designed to conceal. In wireless environments, this concealment carries direct operational consequences, as UDP multicast induces uncontrolled flooding that degrades delivery performance beyond what the middleware layer can detect or compensate [118].

Once the subscriber count crosses the switch's processing threshold, queuing delays accumulate from the mismatch between the logical delivery model and the physical infrastructure's replication capacity, escalating into network-wide jitter amplification. Because all subscribers in the multicast group receive packets from the same switch output queue, a single switch bottleneck degrades the timing behavior of the entire logical communication graph. Benchmarking studies report that delivery latency remains stable below payload-size and subscriber-count thresholds but rises exponentially once exceeded, with maximum latency spikes approaching 5000 microseconds observed at specific configurations [120]–[122]. The escalation is therefore not a gradual degradation but a threshold effect, where system-wide timing predictability collapses abruptly rather than deteriorating incrementally, and the middleware possesses no mechanism for anticipating or signaling this transition to the application layer [123].

Several approaches have been proposed to address this inter-host redundancy. Unicast fallback replaces multicast delivery with individual per-subscriber transmissions once the subscriber count exceeds a configured threshold, eliminating switch-level replication entirely. Experiments demonstrate that beyond endpoint-count thresholds of approximately 50 to 75 publishers and subscribers, unicast communication outperforms multicast in both latency and throughput, providing empirical justification for dynamic mode switching [124]. Container network interface plugins such as Kubernetes WeaveNet intercept multicast packets and transparently convert them to unicast transmissions at the physical network layer, preventing flooding without requiring changes to the middleware configuration [108]. Bridge-based domain segmentation approaches such as zenoh-bridge-ros2dds partition large DDS multicast domains into isolated logical segments interconnected by unicast gossip protocols, resolving infrastructure constraints and topology limitations that standard multicast cannot traverse [60]. In swarm deployments of 20 or more robots, standard UDP multicast over Wi-Fi has been found to be entirely unusable in practice, and centralized discovery servers or unicast-based protocol bridges have been proposed as complete replacements for the multicast-dependent default configuration [125]. Beyond network-layer multicast, middleware-layer router-mediated fan-out provides an alternative absorption point. A Zenoh router on the destination host receives a single message over the network from the publisher and delivers it to local subscribers via the host's intra-host mechanisms, decoupling network-level transmission cost from the number of remote subscribers [47], [76].

Each of these approaches resolves the timing instability by reintroducing physical infrastructure awareness into a system that spatial abstraction was designed to render infrastructure-agnostic. Unicast fallback preserves temporal predictability but abandons the spatial efficiency that motivated multicast adoption in the first place. Unless a routing intermediary absorbs the fan-out, publisher-side transmission overhead scales linearly with the subscriber count, reproducing at the logical layer the cost that multicast was designed to absorb at the physical layer [126]. Router-mediated fan-out shifts the concentration point from the publisher to the routing intermediary, relocating the redundancy cost rather than eliminating it [76]. Bridge-based segmentation requires the deployment architect to partition the logical communication graph according to the physical topology of the underlying network, a dependency that directly contradicts the physical-layer independence that location transparency provides [60]. Correct multicast operation therefore requires operators to possess detailed knowledge of the physical infrastructure rather than treating it as an opaque conduit. Recovering temporal predictability under high subscriber counts requires either sacrificing multicast's spatial efficiency or accepting infrastructure management costs.

Table II summarizes the 42 studies surveyed under the Space–Time conflict. The majority target DDS and are evaluated on wired or local-host configurations, a distribution that reflects the dominance of controlled testbed environments in the literature and partially explains why the physical-layer consequences of spatial abstraction have been undercharacterized relative to their operational severity. Wireless evaluations are concentrated in the fragmentation and multicast subgroups, where physical-layer variability directly exposes the hidden costs that location transparency imposes on delivery timing. Modeling studies remain a minority across all four manifestations, indicating that analytical characterization of Space–Time interactions lags behind empirical evaluation. The key approach column reveals that proposed mitigations consistently fall into one of two structural categories: those that exploit physical co-location, such as shared-memory transport, zero-copy mechanisms, and node composition, and those that reintroduce network-layer awareness, such as MTU-alignment, bitmap-based fragmentation, and domain bridging. This distribution reflects the underlying structure of the conflict itself. Regardless of which manifestation a study targets, every viable mitigation either narrows the deployment scope to physically constrained configurations or partially dismantles the network independence that spatial abstraction provides, confirming that the Space–Time conflict has been navigated at the cost of one or both of its constituent guarantees.

B. Time–State Conflict

Mechanisms for contextual continuity, including reliability protocols and liveliness monitoring, ensure logical consistency in the face of dynamic participation and intermittent connectivity. Yet, the execution of these mechanisms consumes the same computational and communication resources demanded by temporal performance. This conflict initiates at the middleware execution layer and escalates into network-wide resource competition, occasionally triggering uncontrollable positive feedback loops. The trade-off is structural: the more strictly a system enforces state consistency, the more resources it consumes for state management. This subsection examines three manifestations of this escalation. We analyze jitter from internal synchronization, bandwidth erosion by control messages, and congestion arising from reliability-driven recovery.

TABLE II
TAXONOMY OF LITERATURE ADDRESSING THE SPACE–TIME CONFLICT

[Ref]	Protocol	Application Domain	Evaluation Platform	Network			Research Type			Key Approach
				Wired	Wire-less	Local Host	Evalu-ation	System Design	Model-ing	
[105]	DDS	Robotics, Auto.	Workstation	✓		✓	✓			Unicast-baseline
[117]		Robotics	Embedded, Workstation			✓	✓	✓		SHM-orchestration
[88]						✓	✓			Serialization-profiling
[100]			Embedded			✓	✓			Node-composition
[115]			Simulator			✓	✓	✓		Process-overhead
[113]			Workstation	✓		✓	✓			Subscriber-scaling
[15]				✓		✓	✓	✓		Middleware-bypass
[102]					✓		✓	✓		Burst-analysis
[99]					✓		✓	✓		Burst-analysis
[97]				✓			✓		✓	Heartbeat-modeling
[87]						✓	✓		✓	Queue-modeling
[6]						✓	✓			P2P-motivation
[92]						✓	✓	✓		Split-transport
[90]				✓		✓	✓	✓		Layout-preserving
[109]						✓	✓			Intra-process-copy
[108]		IIoT	Embedded, Workstation	✓		✓	✓			Container-overhead
[112]			Embedded	✓			✓			Bandwidth-profiling
[111]			Workstation		✓		✓		✓	Subscriber-latency
[110]				✓	✓		✓	✓		QoS-profiling
[119]				✓			✓	✓		Multicast-scalability
[89]				✓			✓			AutoThrottle
[95]				✓		✓	✓	✓		App-fragmentation
[106]		Auto.	Embedded, Workstation			✓	✓			Copy-overhead
[116]			Embedded			✓	✓	✓		Zero-copy-SHM
[103]			Simulator, Workstation	✓			✓	✓		Subscriber-selective
[96]			Simulator		✓		✓	✓		Wireless-fragment
[98]					✓		✓	✓		Bitmap-fragmentation
[101]			Workstation			✓	✓			SHM-bypass
[122]		(None)	Simulator, Workstation	✓			✓	✓		Threshold-modeling
[126]			Simulator	✓			✓	✓		Unicast-fallback
[121]				✓			✓	✓		Unicast-threshold
[107]			Workstation	✓			✓	✓		Complexity-modeling
[120]				✓			✓	✓		Multicast-spike
[124]						✓	✓			Topology-aware
[94]				✓			✓			CDR-interoperability
[125]	Both	Robotics, IIoT	Embedded, Simulator		✓		✓	✓		Swarm-bridging
[118]			Embedded, Workstation	✓	✓		✓			Protocol-comparison
[91]				✓	✓	✓	✓	✓		Encoding-tuning
[60]		Robotics	Workstation	✓	✓		✓	✓		Domain-bridging
[114]		Auto.	Embedded, Workstation			✓	✓	✓		Middleware-bypass
[93]			Workstation			✓	✓	✓		Heap-zerocopy
[104]				✓		✓	✓			Transport-comparison

1) Lock Contention: Guaranteeing the logical ordering of messages and the consistency of shared state in a distributed environment is a structural requirement for ROS 2 middleware. Without enforced ordering and synchronized access to internal data structures, the middleware cannot provide the contextual continuity that applications depend upon across dynamic participation and concurrent message flows. To satisfy this requirement, the middleware imposes synchronization at its internal queue and callback layers, where multiple execution threads concurrently access shared endpoint state, message histories, and dispatch queues. This synchronization is the mechanism by which state consistency is enforced, and its presence is proportional to the strength of the consistency guarantee the middleware is required to provide [127]. As the continuity requirements intensify, through stricter ordering guarantees, deeper history retention, or higher callback concurrency, the frequency and duration of synchronization operations increase correspondingly.

The primary manifestation of this synchronization cost at the execution layer is lock contention. When multiple threads simultaneously attempt to access a shared queue or endpoint state structure, all but one must enter a waiting state until the lock is released by the current holder. The duration of each waiting interval is nondeterministically distributed, as it depends on the scheduling behavior of the OS, the execution time of the lock holder, and the number of competing threads. A single-threaded ROS 2 executor serializes all subscription callbacks through a fixed polling-point mechanism, causing buffer overflow and message loss when multiple publishers simultaneously saturate the dispatch path [128]. In multi-threaded configurations, publishers, flow controllers, and listeners interact through shared pending message queues, where contention introduces context-switching overhead and nondeterministic delays [129]. As concurrent callback invocations increase, inter-thread interference at these shared structures compounds, degrading timing predictability across the execution layer [130].

As system complexity increases, lock contention escalates from isolated execution stalls into a system-wide source of timing nondeterminism that propagates across middleware, OS, and kernel layers. At the kernel level, FIFO ring buffer processing and thread scheduling contention have been identified as the root cause of end-to-end response time degradation from five ms to 122 ms under increasing message load, with jitter scaling directly with publication rate and message size [131]. Context switching overhead further amplifies this effect: each lock acquisition failure causes the blocked thread to yield its CPU slot, triggering a context switch whose latency adds to the total waiting time and whose frequency grows with the contention rate. Under high publication rates, the combined effect of lock waiting and context switching causes internal dispatch latency to exceed the inter-message interval. At this point, the callback queue accumulates a backlog. Consequently, the delivery timing of subsequent messages becomes correlated with the processing delays of their predecessors rather than their arrival times [128].

Several research directions have addressed contention and timing nondeterminism through structural improvements at multiple layers. Distributed state synchronization using conflict-free replicated data types and atomic operations eliminates halting synchronization from concurrent execution paths without requiring modification of the middleware internals, a principle demonstrated using Zenoh [132]. Assigning mutually exclusive callback groups to dedicated executors separates synchronization concerns from data publication without requiring changes to application code [133]. The synchronization burden can further be removed from the execution path by delegating history retention and state consistency to an independent storage layer, structurally decoupling state management from the latency-critical data exchange pipeline [45]. To resolve this at the middleware layer, priority-aware executor redesigns introduce native preemptive scheduling models such as producer-consumer and thread-dispatch variants. These redesigns allow high-priority callbacks to predictably preempt lower-priority ones without external OS interventions [134]. Middleware-integrated QoS resource managers extend this predictability further by interfacing directly with the RTOS to enforce budget scheduling and temporal isolation, maintaining correct execution semantics during real-time system reconfiguration [135].

Each approach reduces contention overhead by accepting constraints on state consistency or infrastructure. Storage-based decoupling removes the synchronization burden from the execution path but introduces an independent storage infrastructure, adding resource and configuration overhead. Priority-aware executors improve latency predictability for prioritized callbacks but do not eliminate contention for lower-priority flows [134]. To enable end-to-end timing analysis, communication flows must be decoupled, which means accepting delayed consistency over immediate processing, and strict destination order guarantees must be relaxed to make the system schedulable [136]. Under these relaxed consistency conditions, when asynchronous publication rates exceed subscription processing rates, message loss cannot be prevented even under bounded buffer configurations, meaning that temporal predictability can only be preserved by accepting data loss rather than full state continuity [128]. The structural outcome is consistent: achieving temporal predictability in the presence of concurrent state access requires either weakening the consistency guarantees or redistributing consistency responsibility onto a separate architectural component.

2) Control Overhead: Reliable communication in ROS 2 middleware requires each entity to continuously exchange control messages with its peers, including Heartbeat announcements, AckNack acknowledgments, and session management messages, to verify liveliness and delivery state. These control traffic constitute a permanent structural tax on network capacity. The OMG DDS specification mandates that each entity continuously signal its alive status within the configured lease duration, leaving the middleware no architectural basis for suspending this exchange during periods of low data activity [86]. Control traffic is a steady-state background load primarily driven by the number of active endpoint relationships, though it remains partially influenced by application data rates. In multi-robot deployments, control messages for data-readiness notifications and keepalive mechanisms accumulate under scaled configurations. This accumulation has been directly measured to cause bandwidth overflow and network saturation [137].

In DDS, control messages and application messages compete for the same network bandwidth and CPU resources without priority separation, so Heartbeat and acknowledgment signals directly consume bandwidth otherwise available for latency-sensitive delivery [138]. This contention is further compounded in containerized environments, where network virtualization plugins introduce additional CPU and bandwidth competition that becomes particularly pronounced at high transmission frequencies and small payload sizes [108]. In wireless lossy networks, this interference manifests as local latency spikes under reliable QoS, where lower-layer loss recovery and middleware-level retransmission triggered by missed acknowledgments compete directly for the same path [139]. As participant count grows, discovery and Heartbeat or AckNack retransmission traffic scales linearly and saturates the software transmission queue at the data link layer, creating a bottleneck shared with application data [140].

As system scale increases, the bandwidth erosion caused by control traffic escalates from a marginal overhead into a dominant structural constraint. This erosion is continuous: each active endpoint relationship contributes a persistent stream of Heartbeat and acknowledgment exchanges that cannot be suspended regardless of application data rate. Prior work formally proves that the total transfer count in a unicast domain scales as $O(E \times P)$, where E and P denote the number of endpoints and participants in the domain, respectively, exhausting both network resources and memory as the endpoint population grows [107]. Empirical scaling experiments confirm that as the number of ROS 2 nodes increases from two to five, latency spikes sharply and message drop rate rises exponentially, demonstrating that this steady-state bandwidth erosion constitutes a systemic bottleneck rather than a localized performance penalty [139].

Several approaches have been proposed to reduce the bandwidth footprint of control traffic without abandoning the reliability guarantees it supports. Heartbeat period and lease duration tuning reduces the frequency of liveliness announcements by adjusting the interval between consecutive control message transmissions, and analytical modeling confirms that the Heartbeat period directly governs the frequency of AckNack generation and retransmission attempts [97], [141]. DDS parameter optimization, including synchronous publish modes and QoS tuning such as best-effort QoS and keep last history policies for small high-frequency packets, has been shown to reduce communication latency and improve real-time performance in distributed robotic control scenarios [142]. At the protocol level, batching and coalescing mechanisms aggregate multiple messages into single message; both DDS implementations and Zenoh's session protocol employ batching to consolidate control messages and minimize protocol overhead. More fundamentally, transport-level reliability substitution via connection-oriented protocols such as TCP or QUIC removes the per-endpoint AckNack exchange from the middleware-visible communication path entirely. This has been demonstrated using Zenoh [70], and comparable latency benefits are achievable in DDS deployments by switching to TCP-based transport configurations [143]. At the architectural level, Zenoh assigns a configurable priority to control traffic, allowing an architect to shift this competition between control and application flows rather than eliminate it.

Each mitigation strategy either reduces control traffic at a cost to contextual continuity, displaces that responsibility onto the transport layer, or reorders the contention through priority assignment without removing it. Heartbeat period tuning and batching reduce message frequency but increase the latency with which liveliness failures are detected, directly weakening the timeliness of contextual continuity maintenance [138]. In large-scale multi-robot deployments, sustained state consistency maintenance has been observed to generate excessive data flows that exhaust communication resources and destabilize the network [144]. Experiments further demonstrate that, in DDS-based deployments, activating complete security and reliability guarantees reduces throughput from approximately 70,000 to 14,000 packets per second and increases latency by a factor of five [142], and that achieving timing determinism in practice requires adopting best-effort QoS rather than reliable QoS [145]. The dimensional outcome is consistent: reducing the steady-state cost of control traffic requires either weakening the state consistency that the traffic exists to maintain or accepting degraded temporal predictability [108].

3) Retransmission Congestion: Reliable communication in ROS 2 middleware is predicated on the assumption that delivery failures must be detected and corrected as quickly as possible to preserve contextual continuity across distributed nodes. Under reliable QoS, the DDS specification requires the middleware to retransmit missing samples and replay retained history for reconnecting endpoints, treating every unacknowledged sample as a deficit to be resolved without delay [86], [138]. Consequently, the DDS architecture prioritizes state completeness over transmission efficiency, as the middleware must reconstruct a correct historical snapshot before exposing new message to the application [127].

The structural vulnerability of this approach surfaces specifically when the underlying network is already operating under congestion. Because the reliability mechanism monitors endpoint-level sequence state rather than network-level resource availability, it cannot distinguish between isolated packet loss and symptomatic of link saturation. In both cases, the response is identical: retransmit immediately. Analytical modeling shows that periodic Heartbeat messages broadcast by the publisher induce AckNack responses from receivers and trigger immediate retransmission for every detected sequence gap, establishing a structurally unconditional retransmission pathway that operates independently of channel load [141]. Under congested conditions, this retransmission injects additional traffic into a channel that is already unable to clear its existing load, directly increasing queue occupancy and elevating the probability of further loss. The contrast between reliable and best-effort QoS makes this dynamic explicit. Reliable QoS continuously retries every missing sample in the HistoryCache, consuming retransmission resources without bound, whereas best-effort does not attempt recovery and thereby avoids contributing to congestion [146].

The transition from isolated retransmission events to system-wide degradation follows a positive feedback structure. Each retransmission attempt adds traffic to an already-saturated channel, which induces further packet loss, which in turn triggers additional retransmission, forming a self-reinforcing loop in which the state recovery mechanism itself amplifies the loss it was designed to correct. Experiments demonstrate that scaling ROS 2 nodes over DDS in a wireless lossy network under reliable QoS causes average latency and message drop rate to increase exponentially, reaching communication collapse as retransmission attempts overwhelm the available channel capacity [147]. History recovery at reconnection further compounds this escalation. The middleware floods the channel with retained samples at the moment of rejoining, competing directly with control messages on the same transport path and prolonging the period during which the feedback loop remains active. As system scale grows, the communication cost and computation overhead associated with state synchronization increase concurrently [148], making the system increasingly susceptible to retransmission feedback loops.

Several approaches have been proposed to interrupt the positive feedback loop. Time-based filtering mechanisms discard intermediate updates under high-volume delivery conditions, preventing channel flooding without disabling the reliability mechanism entirely [149]. Custom QoS policies incorporating inter-robot velocity differentials can dynamically adjust transmission window sizes to minimize retransmission events [150]. Congestion-aware retransmission control feeds network congestion signals from the 5G layer back into the middleware, enabling proactive throttling before saturation is reached [151]. Disabling MAC-layer retransmission and replacing the AckNack loop with a purpose-built protocol prevents the collision between lower-layer and middleware-layer retransmission cycles [98]. Zenoh supports storage-based separation of history recovery from active data exchange. Zenoh

provides dedicated storage managers that retain samples for specific key expressions and serve them to late joiners via pull-based queries, ensuring that history delivery no longer competes with ongoing transmissions on the primary data channel [82]. DDS can achieve analogous separation through persistent durability services. Fundamentally, leveraging the frame replication within Time-Sensitive Networking (TSN) bypasses middleware-level retransmissions entirely by providing lossless redundancy at the network layer [152], but doing so couples the middleware to a network configuration, introducing a Space–State conflict.

These approaches achieve their benefits by accepting a constraint on either the timeliness of state recovery or the completeness of delivery guarantees. Congestion-aware throttling introduces delay between loss detection and retransmission delivery. Multi-robot results demonstrate that network bottlenecks can only be prevented by differentially applying reliable and transient local QoS to high-priority information while relegating other data to best-effort and volatile QoS, confirming that complete reliability guarantees and real-time network performance have not been simultaneously achieved in current deployments [153]. Storage-based decoupling removes the reconnection burst but introduces a consistency boundary between the storage manager and the producing endpoint whose divergence must be managed as a separate operational concern, trading one form of state overhead for another. No mechanism simultaneously preserves the immediacy of state recovery and protects temporal performance under congestion; every viable approach either delays state restoration or redistributes its cost onto a separate architectural component.

Table III summarizes the 34 studies surveyed under the Time–State conflict. Wireless evaluations account for a notably higher proportion than in the Space–Time group, reflecting that retransmission congestion and control overhead manifest most severely under lossy network conditions where the gap between reliability demands and available capacity is widest. Modeling studies are more evenly distributed across the three subgroups, indicating that the Time–State interaction has received comparatively more formal analytical attention, particularly in characterizing Heartbeat periodicity, retransmission dynamics, and executor contention. The key approach column reveals that mitigation strategies cluster around two structural directions. Those that reduce state enforcement cost by relaxing consistency guarantees, including best-effort-tuning, time-filtering, order-relaxation, and policy-selection, and those that relocate state management responsibility away from the latency-critical path, including state-decoupling, lock-free, and TSN-replication. This distribution mirrors the structural conclusion common to all three subgroups: no proposed approach simultaneously preserves the immediacy of state recovery and the predictability of data delivery.

C. Space–State Conflict

While spatial abstraction aims to provide location transparency by shielding applications from physical placement, maintaining this independence either increases the structural cost of state management or makes state tracking structurally constrained under shared resource bounds. Conversely, attempts to reduce state management overhead consistently converge toward restricting the scope of spatial abstraction. This conflict forms a paradoxical escalation structure: it originates at the endpoint level, extends to transmission paths, and intensifies at the network level. This subsection examines four manifestations of this tension: node anonymity that obscures lifecycle state, path-blind forwarding that conceals buffer conditions, the discovery storms that emerge from P2P state growth, and the structural vulnerabilities introduced by intermediary-dependent state management.

1) Node Anonymity: Location transparency in ROS 2 middleware is realized through data-centric anonymity: communication is organized around named topics and key expressions rather than physical node identities, so that receivers obtain data by declaring interest in a logical channel without any knowledge of who is producing it or from where. This design is a deliberate structural principle that allows applications to remain indifferent to deployment topology and to interoperate across heterogeneous nodes without prior coordination [107]. The publish-subscribe model explicitly decouples data sources and data sinks with respect to node location and temporal requirements, ensuring that applications require no knowledge of the existence or location of other participating entities [154]. The same anonymity, however, imposes a structural constraint on state management. Because the middleware does not track the physical identity of communicating entities, it has no architectural basis for associating the persistent state accumulated during middleware operations with the specific physical entity that produced it, making contextual continuity dependent on an identity resolution capability that spatial abstraction structurally withholds.

Under the anonymous endpoint model, the middleware has no architectural basis for distinguishing a rejoining node from a newly appeared one across deployment events. Runtime endpoint identifiers such as the DDS GUID and the Zenoh session ID can distinguish a reconnecting endpoint from a first-time connection during the identifier's lifetime, but they do not survive redeployment or identifier change. The DDS liveliness QoS policy governs whether an entity is still active by treating any entity that fails to assert its alive status within the configured lease duration as no longer active [86]. Each participant therefore stores the information of all remote endpoints it has discovered in an internal database, and per-participant memory consumption grows linearly with the total number of endpoints in the domain [107]. Because DDS operates without a message broker, the burden of tracking which stored state belongs to which physical entity falls entirely on the individual participant [155].

The system-level consequences of this identity ambiguity escalate along two distinct paths. In the first path, ghost nodes that persist beyond their operational lifetime occupy endpoint slots in the discovery state and consume liveliness monitoring resources as the middleware continues issuing Heartbeat checks to unreachable addresses. In some configurations, newly joining nodes with matching topic profiles receive stale cached state intended for the original terminated entity. Prior work demonstrates that when a publisher is deleted without calling

TABLE III
TAXONOMY OF LITERATURE ADDRESSING THE TIME–STATE CONFLICT

[Ref]	Protocol	Application Domain	Evaluation Platform	Network			Research Type			Key Approach
				Wired	Wireless	Local Host	Evaluation	System Design	Modeling	
[151]	DDS	Robotics, IIoT	Workstation, Simulator		✓		✓	✓		Congestion-throttle
[128]		Robotics, Auto.	Workstation, Simulator			✓	✓	✓	✓	Executor-modeling
[147]			Simulator		✓		✓			Wireless-scaling
[133]			Workstation			✓	✓	✓		Callback-isolation
[134]						✓	✓	✓	✓	Preemptive-executor
[130]						✓	✓		✓	Contention-modeling
[148]		Robotics	Embedded, Workstation, Simulator		✓		✓	✓		Sync-scaling
[150]			Embedded, Simulator		✓		✓	✓		Adaptive-QoS
[153]					✓		✓	✓		Policy-selection
[144]			Embedded		✓		✓	✓		Swarm-overhead
[137]					✓		✓			Wireless-scaling
[131]				✓			✓	✓	✓	Kernel-contention
[129]						✓	✓	✓		Queue-isolation
[139]			Simulator		✓		✓			Control-scaling
[142]			Workstation	✓			✓			BestEffort-tuning
[145]					✓	✓	✓	✓		Reliability-cost
[97]				✓			✓		✓	Heartbeat-modeling
[140]					✓		✓	✓	✓	Discovery-saturation
[138]					✓			✓	✓	Control-dependency
[141]				✓			✓		✓	Retransmit-modeling
[108]		IIoT	Embedded, Workstation	✓		✓	✓			Container-overhead
[149]			Workstation		✓		✓	✓		Time-filtering
[135]				✓			✓	✓		Budget-scheduling
[152]		Auto.	Embedded, Workstation	✓			✓	✓	✓	TSN-replication
[136]			Embedded	✓			✓		✓	Order-relaxation
[98]			Simulator		✓		✓	✓		Bitmap-fragmentation
[146]		(None)	Workstation		✓		✓			QoS-contrast
[107]				✓			✓	✓		Complexity-modeling
[127]			-					✓		Consistency-model
[86]								✓		Spec-analysis
[70]	Zenoh	IIoT	Embedded	✓	✓		✓			Transport-replace
[45]	Both	Robotics, IIoT, Auto.	Workstation	✓		✓	✓	✓		State-decoupling
[132]						✓	✓	✓		Lock-free
[143]		Auto.	Embedded, Workstation	✓	✓		✓			Protocol-comparison

a disconnect service, the tracker continues to register the publisher as active, accumulating stale state that cannot be cleared without explicit lifecycle signaling [115]. In the second path, because the middleware associates endpoints with logical topic identifiers rather than persistent physical identities, a reconnecting node is indistinguishable from a newly arrived entity sharing the same configuration. The middleware has no criterion for determining whether a reconnecting endpoint should receive previously accumulated lifecycle context or be treated as a new lifecycle context. When both paths are active simultaneously, the system must manage an expanding ghost state while attempting to restore contextual continuity for reconnecting nodes. It produces a state management burden whose cost grows with both system scale and the frequency of node lifecycle events.

Several structural approaches address the node anonymity problem from different layers. Assigning unique physical identifiers to each robot and creating dedicated per-robot communication channels through topic name suffixing, combined with explicit connection and disconnection service calls, provides a functional approach to dynamic participation management [115]. Purpose-built DDS-based middleware circumvent the anonymity-induced tracking gap through application-level identity enforcement, such as explicit join codes and periodic Heartbeats [156]. A Named Data Networking (NDN) approach combined with a distributed query interface identifies data for specific key expressions regardless of which physical node originally produced it, allowing a reconnecting node to retrieve its prior state via a pull query without requiring identity resolution at the middleware layer. This addresses state recovery continuity rather than the underlying anonymity constraint, and Zenoh demonstrates the principle in practice [132]. Combining ROS 2 managed nodes with external orchestrator health-check control loops such as Kubernetes binds logical component identities to physical process lifecycles and enables automatic restart management at the deployment level [157].

Each approach resolves the identity-state coupling by partially dismantling the anonymity on which location transparency depends. Storage-based decoupling requires the storage manager to explicitly track which endpoint produces which data and to keep its retained state aligned with that endpoint, a coupling that the fully anonymous publish-subscribe model was designed to render irrelevant [45]. Process-level lifecycle management requires external orchestrators to explicitly track deployment-level information that location transparency was designed to render irrelevant [157]. Prior work further establishes that in DDS architectures, state tracking cannot be fully decoupled from physical identity to maintain mutual consistency [127]. The structural outcome is consistent across all approaches: achieving lifecycle state continuity under dynamic node participation requires the system to acquire and use physical identity information that spatial abstraction withholds by design.

2) Path-Blind Forwarding: Spatial abstraction in ROS 2 middleware structurally constrains the state that the middleware is permitted to maintain. Because location transparency requires the routing path to be treated as an opaque conduit, the middleware holds no representation of downstream buffer occupancy, queue depth, or link capacity as part of its state. This is not an implementation omission but a structural consequence of spatial abstraction. The middleware accepts a message, serializes it, and injects it into the socket send buffer without consulting any lower-layer information about the physical path connecting sender to receiver [102]. The middleware's state is therefore spatially bounded, complete with respect to logical endpoint relationships, but blind to the physical infrastructure over which those relationships are realized.

When network load is low, this state incompleteness proceeds without visible consequence because downstream buffers drain faster than the middleware fills them. Under saturation, however, the kernel socket buffer and switch queues accumulate backlogged packets at a rate that exceeds their service rate, causing subsequently injected packets to experience queuing delays independent of the physical propagation latency between sender and receiver. Results show that when a wireless link outage occurs, ROS 2 applications and the DDS middleware continue accumulating unacknowledged samples in the HistoryCache until its capacity limit is reached [99]. Upon link recovery, the middleware releases the accumulated backlog in an instantaneous burst without regulating the transmission rate, while the application continues publishing new messages into the cache at its original rate, compounding the saturation [102]. Because the middleware maintains no state representing the physical path condition, it possesses no mechanism to detect this accumulation or moderate its injection behavior in response.

The escalation from localized buffer saturation to system-wide delivery degradation follows the same positive feedback structure identified in Section V-B3. CSMA/CA-based wireless channel contention at the data link layer causes backlog to accumulate in the software transmission queue, resulting in packet drops and a bottleneck shared across all flows on the affected interface [140]. This backlog propagates upward. Because ROS 2 employs FIFO buffering, end-to-end latency increases linearly with backlog depth when the publisher injection rate exceeds the subscriber processing rate [128], so that a single saturation event corrupts the timing of an extended sequence of subsequent messages [115]. When multiple high-frequency topics share the same physical interface, the saturation of one topic's buffer path induces secondary saturation on co-located flows, spreading the impact of a single overload event across the entire communication graph [158].

Several approaches have been proposed to mitigate the performance consequences of path-blind forwarding. Socket buffer resizing offers a structurally simpler alternative by raising the saturation threshold, reducing the frequency of bloat events under transient load spikes without modifying the injection model [158]. Local cache mechanisms discard redundant data before it enters the middleware buffer, and mathematical modeling of the relationship between deadline and depth QoS parameters enables balancing optimization that eliminates transmission rate and buffer size mismatches [159], [160]. Building on this, adaptive task execution rate regulation addresses the injection-consumption mismatch by analyzing message consumption capacity at runtime and dynamically adjusting the sensor message sampling and injection rate to match the actual processing bottleneck [161]. Within the middleware, runtime queue occupancy monitoring with dynamic QoS reconfiguration, including queue depth and time-based filters, provides a self-adaptive framework for preventing buffer overflow [162]. Configuring DDS history and resource limits QoS policies in proportion to the available link bandwidth prevents both buffer overflow and link saturation [102], [138]. Congestion-aware transmission control introduces a feedback path from the network layer to the middleware's injection logic, where DDS tracks send window occupancy and Nack volume to actively modulate the publication rate under congestion [89].

Each mitigation strategy achieves its benefit through a different degree of path state awareness, but all share the structural consequence of partially undermining the path independence that spatial abstraction provides. Socket buffer resizing defers rather than eliminates saturation, addressing the consequence of blind forwarding without resolving its structural cause [27]. Congestion-aware control and queue monitoring require the middleware to actively track physical forwarding state [89]. Adaptive injection rate control binds transmission behavior to deployment-specific execution characteristics that location transparency was designed to render irrelevant [161]. QoS-based resource bounding requires the operator to configure limits explicitly calibrated to the physical link's capacity [102]. The dimensional outcome is consistent: approaches that preserve path independence defer saturation rather than eliminate it, while those that substantively reduce forwarding overhead do so by reintroducing physical path awareness, thereby narrowing the scope of spatial abstraction.

3) Discovery Storm: Maintaining location transparency requires the middleware to continuously construct and update a logical visibility graph in which every participating entity possesses awareness of every other entity's existence and metadata. This full-mesh visibility is a structural prerequisite of location transparency, since without it the middleware cannot guarantee that newly joining entities will be incorporated into the communication graph without manual configuration. Prior work establishes that DDS creates and maintains a fully connected graph by design, causing all participants, topics, and services to discover one another mutually and producing $O(N^2)$ network traffic as a structural consequence [44]. The coupling between spatial scale and state management cost is therefore not a contingent implementation choice but a necessary consequence of the full mutual visibility that location transparency demands.

The total number of DDS discovery messages scales as $O(P^2)$ with participant count, with per-node transmissions on the order of $P \times E$, where E is the endpoint count. Memory consumption at each node is closely correlated with the number of endpoints it must store, forcing nodes in large networks to retain endpoint information entirely unrelated to their own communication context [163]. In UAV swarm environments, the default discovery process generates substantial communication

overhead that prevents reliable peer detection between vehicles, with network traffic growing nonlinearly with node count to the point where the discovery process itself can stall [164].

The transition from elevated discovery traffic to system-wide determinism collapse follows a congestion-driven escalation path. The initial burst of DDS discovery traffic during simultaneous node startup interacts with the software transmission queue's waiting-time threshold to cause severe network saturation and packet drops. This interaction has been formally modeled as the root cause of discovery-driven packet loss in multi-robot ROS 2 systems [140]. Experiments demonstrate that latency increases from approximately 5.5 ms at five nodes to 1309 ms at twenty nodes, and that at twenty-five nodes the packet loss rate reaches 94.98%, rendering the network functionally unusable [137]. This escalation is particularly damaging for latency-sensitive robotic applications, because the periods of highest discovery load, such as system initialization and dynamic reconfiguration, coincide precisely with the moments when timely message delivery is most critical for system safety and coordination.

Several approaches have been proposed to reduce the discovery state management overhead and prevent discovery storm events. Memory-efficient discovery optimization using extended threshold Bloom filter vectors transmits and stores only compact filter representations of participant endpoint descriptors rather than full metadata records, substantially reducing both memory consumption and network transmission volume [165]. Domain segmentation partitions the participant space into chip-local and vehicle-area network segments, confining discovery traffic within each segment and using message bridges to forward data across boundaries, preventing cross-segment $O(N^2)$ growth while preserving inter-segment communication [166]. Hierarchical discovery structures replace the P2P model with a topology in which sessions register with a designated router; measurements show reductions in discovery overhead of 97% to 99.9% relative to P2P DDS implementations, with corresponding reductions in CPU utilization and latency [167], and DDS deployments address the same scalability problem through optional discovery servers that similarly replace flat multicast discovery with a star topology [62]. Zenoh-DDS bridges in swarm testbeds achieve transparent ROS 2 connectivity while substantially lowering discovery traffic levels and eliminating the multicast dependency of flat DDS discovery [125].

Each mitigation strategy reduces discovery state cost by constraining the scope of mutual visibility, and in doing so partially relinquishes the full-mesh awareness on which location transparency depends. Domain segmentation reduces per-domain discovery cost but introduces explicit topological boundaries that applications must account for, requiring deployment-level knowledge that location transparency was designed to render unnecessary. Centralized discovery architectures achieve the most substantial scalability improvement but concentrate the entire system's visibility state in a single intermediary; prior work explicitly notes that if the intermediary fails, the entire system may fail to function [163], [168]. This structural consequence introduces a different conflict examined in the following subsection. The structural outcome is consistent: no approach simultaneously preserves complete mutual visibility and eliminates $O(N^2)$ discovery cost, and every viable reduction in state management overhead is obtained by accepting a corresponding reduction in the spatial scope of location transparency.

4) Intermediary Dependency: The centralized discovery architectures introduced in the preceding subsection replace the flat P2P visibility graph with a hierarchical structure in which contexts no longer maintain direct mutual awareness. Instead, each context registers its endpoint state with a central intermediary, such as a DDS Discovery Server or a Zenoh router, which assumes responsibility for matching and forwarding decisions [44]. This restructuring shifts visibility responsibility from each context to one or more intermediaries, reducing per-participant overhead while concentrating the discovery process at the intermediary tier. When the intermediary tier lacks redundancy, the loss of a router suspends new endpoint discovery and cross-host communication, even though already-established peer sessions on the same host continue to exchange data directly [169]. Although this resolves the $O(N^2)$ scalability problem, it introduces a structural dependency whose failure semantics are categorically different from those of the P2P model it replaces. Notably, ROS 2 adopted P2P DDS discovery precisely to eliminate the single point of failure inherent in the ROS 1 centralized name server [6], meaning that the centralization adopted for scalability reintroduces a single point of failure of the kind that P2P discovery was designed to avoid.

The failure consequences of these two topologies differ not merely in degree but in kind. In a flat P2P topology, the failure of any single node degrades the visibility graph only locally, where the affected node's endpoints become unreachable but all remaining participants retain mutual awareness and continue to communicate. In a centralized topology, such as a single router with all applications in client mode, the broker mediates all communication between applications. Its unavailability therefore halts both endpoint discovery and ongoing data exchange, since client-mode applications do not establish direct sessions with one another [170]. Prior work argues that centralized approaches are fundamentally unsuitable for real-time and fault-tolerant applications, as concentrating the true values of all data objects on a single computer is architecturally unsound [42].

The severity of this vulnerability is compounded by the operational position of the intermediary within the system. The discovery intermediary handles the aggregated discovery traffic of all contexts and must process endpoint registration and matching requests at a rate that scales with system size, making it the node most exposed to sustained load. As systems grow, maintaining availability and throughput requires deploying additional intermediaries, with cost, complexity, and latency all worsening proportionally [171]. Achieving high availability further demands non-default redundancy mechanisms that lie outside the standard specification scope [155]. In cloud robotics deployments, this structural constraint has been formally characterized through the LSC theorem, which proves that latency reliability, singleton deployment, and commodity infrastructure cannot be simultaneously achieved [172].

Several mechanisms have been proposed to mitigate this vulnerability. Multi-level service discovery eliminates central dependency by partitioning communication into local and wide-area segments using message bridges, which decentralizes discovery and prevents any localized failures from compromising the entire system [166]. Repository federation allows multiple centralized discovery servers to collaborate, effectively eliminating the single point of failure by ensuring system robustness even if the primary repository becomes unavailable [163]. Redundant router deployment over independent network paths distributes failure risk across replicated intermediaries and allows participants to fall back to an available replica; this approach is supported by both Zenoh's routed topology [169] and DDS deployment configurations that combine multiple discovery servers [62]. Probabilistic latency-reliability models that replicate requests across independent network interfaces and cloud servers reduce communication failure probability exponentially, providing a quantitative framework for trading infrastructure cost against availability guarantees [172].

Each of these responses reveals the same structural cycle that connects this subsection to the preceding one. Intermediary replication reduces failure risk but increases the complexity and resource cost of maintaining consistent state across replicas, partially negating the operational simplicity that centralization provided [155]. Hybrid topologies that maintain limited cross-group P2P links bound the failure impact but reintroduce partial peer-to-peer state management within each group, reproducing a bounded version of the $O(N^2)$ cost that centralization was designed to eliminate [169]. Automatic P2P fallback preserves communication continuity after intermediary failure but does so by reverting to precisely the scalability problem that motivated centralization [44]. This cycle constitutes a structural boundary of the present design space. Eliminating the single point of failure reintroduces distributed state cost, while eliminating distributed state cost reintroduces the single point of failure. The structural outcome is consistent: no architecture within the current middleware design space resolves both constraints simultaneously [171].

Table IV summarizes the 39 studies surveyed under the Space–State conflict, the largest of the three groups. Wireless evaluations dominate, consistent with the observation that node anonymity and path-blind forwarding manifest most severely under intermittent connectivity, where the gap between logical endpoint state and physical network reality is widest. A notable portion of entries lack a specified evaluation platform, concentrated among architectural analyses and specification-level studies that provide the structural foundation for the discovery storm and intermediary dependency subgroups. The key approach column reveals a mitigation landscape that is markedly more diverse than in the other two conflict pairs, spanning identity enforcement, path-aware congestion control, Bloom-filter-based discovery, domain segmentation, NDN-based querying, and router federation. This diversity is itself a structural observation: unlike the Space–Time conflict, where mitigations cluster around two recognizable categories, the Space–State conflict has not converged on a dominant architectural response. Every proposed approach addresses

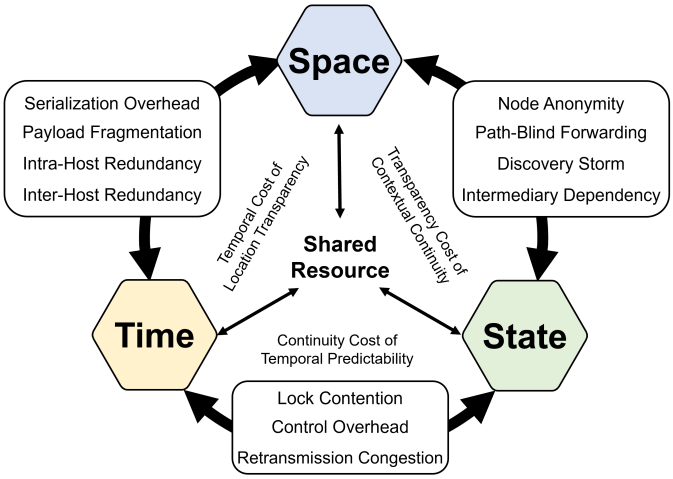

Fig. 3. Structural Trade-off Cycle of ROS 2 Middleware

one aspect of the anonymity-state or scale-fragility cycle, confirming that the Space–State conflict has not been resolved within the current middleware design space.

Fig. 3 synthesizes the structural trade-off cycle identified across the eleven manifestations analyzed in this section. Each directed arc represents an unavoidable cost: location transparency imposes a temporal cost by concealing physical network state from the middleware, temporal predictability imposes a continuity cost by forcing reliability guarantees to be relaxed under real-time constraints, and contextual continuity imposes a transparency cost by requiring physical identity information that spatial abstraction structurally withholds. Because all three mechanisms draw upon the same bounded shared resources, namely execution capacity, network bandwidth, and buffer memory, fully satisfying one dimension imposes pressure on at least one of the others. No architecture within the current middleware design space resolves all three simultaneously; any proposed mitigation navigates the trade-off surface rather than escaping it. This unresolved tension is not an artifact of implementation choices but a structural property of distributed robotic middleware under constrained deployment conditions. Section VI addresses this structural boundary at two levels: immediate gaps in current middleware practice that can be closed without architectural redesign, and long-term research directions that require rethinking the foundations of how *Space*, *Time*, and *State* are jointly realized in distributed robotic systems.

The central finding that emerges from this review is not that individual solutions are inadequate, but that every proposed mitigation trades one dimensional guarantee for another. Studies addressing the Space–Time conflict consistently demonstrate that reducing serialization cost, fragmentation loss, or multicast instability requires the middleware to reintroduce physical-layer awareness that location transparency was designed to conceal. Studies addressing the Time–State conflict reveal that the control traffic and retransmission mechanisms required to maintain contextual continuity consume the same execution and network resources that temporal predictability depends upon, and that relaxing reliability guarantees is the only operationally viable path to recovering bounded latency under congestion. Studies addressing the Space–State conflict show

TABLE IV
TAXONOMY OF LITERATURE ADDRESSING THE SPACE–STATE CONFLICT

[Ref]	Protocol	Application Domain	Evaluation Platform	Network			Research Type			Key Approach
				Wired	Wireless	Local Host	Evaluation	System Design	Modeling	
[172]	DDS	Robotics, Auto.	Embedded, Workstation		✓		✓	✓		LSC-theorem
[128]			Workstation, Simulator			✓	✓	✓	✓	Executor-modeling
[161]			Workstation			✓	✓	✓		Rate-adaptation
[168]		Robotics, IIoT	-					✓		Intermediary-risk
[171]								✓		Centrality-cost
[148]		Robotics	Embedded, Workstation, Simulator		✓		✓	✓		Sync-scaling
[159]			Embedded, Workstation		✓		✓	✓		Buffer-modeling
[160]					✓		✓	✓		Deadline-balancing
[27]			Embedded			✓	✓	✓		Path-blind
[137]					✓		✓			Wireless-scaling
[115]			Simulator			✓	✓	✓		Process-overhead
[156]			Workstation		✓		✓	✓		Identity-enforce
[140]					✓		✓	✓	✓	Discovery-saturation
[138]					✓			✓	✓	Control-dependency
[102]					✓		✓	✓		Burst-analysis
[99]					✓		✓	✓		Burst-analysis
[6]						✓	✓			P2P-motivation
[162]						✓	✓	✓		Self-adaptive
[154]			-					✓		Anonymity-model
[149]		IIoT	Workstation		✓		✓	✓		Time-filtering
[89]				✓			✓			AutoThrottle
[165]				✓			✓	✓	✓	Bloom-filter
[163]				✓			✓	✓		Bloom-filter
[42]			-					✓		Centrality-risk
[155]							✓			Broker-analysis
[166]		Auto.	Embedded			✓	✓	✓		Domain-segment
[158]			Workstation, Simulator			✓	✓		✓	Buffer-sizing
[107]		(None)	Workstation	✓			✓	✓		Complexity-modeling
[86]			-					✓		Spec-analysis
[127]								✓		Consistency-model
[169]	Zenoh	Robotics, IIoT	-					✓		Router-federation
[132]	Both	Robotics, IIoT, Auto.	Workstation			✓	✓	✓		Lock-free
[125]		Robotics, IIoT	Embedded, Simulator		✓		✓	✓		Swarm-bridging
[164]		Robotics	Embedded, Simulator	✓	✓	✓	✓	✓		UAV-discovery
[167]			Embedded		✓		✓			Router-overhead
[170]			-				✓			Failure-semantics
[44]								✓		P2P-cost
[157]		IIoT	-					✓		Lifecycle-mgmt
[143]		Auto.	Embedded, Workstation	✓	✓		✓			Protocol-comparison

that the anonymity and full-mesh visibility on which location transparency relies simultaneously prevent lifecycle state from being tracked and drive discovery overhead that grows with system scale, while the centralized architectures introduced to contain that growth reintroduce a single point of failure of the kind that peer-to-peer discovery was designed to avoid. Across all three pairs, the structural conclusion is the same: proposed solutions navigate the trade-off surface rather than escape it, and the boundaries of that surface are determined by the shared resource constraints under which all three dimensions must be simultaneously realized.

VI. STRUCTURAL GAPS AND RESEARCH ROADMAP

The dimensional conflict analysis in Section V characterizes the trade-off surface that current middleware architectures must navigate. Navigating this surface effectively, however, depends on resolving practical deficiencies that exist independently of the dimensional conflicts themselves. This section addresses both concerns at two distinct levels. The first level identifies structural gaps in current middleware design and operational practice whose resolution is a prerequisite for reliable deployment under realistic conditions. The second level proposes long-term architectural directions that target the root causes of the dimensional conflicts identified in Section V.

A. Structural Gaps

This subsection identifies three structural gaps in current middleware practice, namely QoS semantic loss, evaluation inadequacy, and security overhead. These gaps are not derived from the dimensional conflict analysis but represent independent deficiencies in how middleware is configured, evaluated, and secured in practice. Each admits targeted improvements within the current design space, yet none has been comprehensively resolved in the existing literature.

1) QoS Semantic Loss: QoS semantic loss refers to the degradation of QoS enforcement at the boundary between the middleware layer and the underlying OS and physical layer. Temporal predictability has repeatedly been undermined because logical QoS configurations are not propagated into OS kernel scheduling or network-level priority mechanisms. This enforcement failure produces priority inversions and unpredictable delivery latency under congested wireless conditions [131], [173]. In wide-area deployments, the absence of a standard mechanism

for end-to-end QoS propagation further causes bandwidth and latency violations that the middleware cannot detect or compensate [119]. This breakdown has been systematically identified as a major deployment obstacle in DDS-based systems [168]. Its recurrence across wireless deployments, wide-area networks, and real-time control scenarios confirms that it is not an isolated tuning problem but a structural gap in how QoS intent is realized across the communication stack.

The structural origin of this gap lies in how the middleware maps application-level QoS parameters to physical transport mechanisms. QoS configurations are treated as isolated per-endpoint attributes rather than cross-layer properties with dependency chains that span the execution, transport, and network layers [138]. The strict architectural separation between the RMW interface and the OS prevents the middleware from directly influencing hardware acceleration pipelines or network scheduling queues, creating a systematic mismatch between logical intent and physical packet processing [72]. This mismatch does not require a fundamental redesign of the core communication architecture to address, as the enforcement logic and the translation infrastructure surrounding it can be improved independently within the current design space [136], [151].

Prior work has proposed modeling and validation approaches that reduce configuration errors and improve semantic consistency within the middleware layer. Automated dependency validation and QoS capability profiling at design time can prevent semantic inconsistencies between component ports before execution begins [138], [174]. Domain-specific modeling languages and generative frameworks can synthesize semantically compatible policy configurations and eliminate trial-and-error tuning by producing correct-by-construction parameter sets prior to deployment [175], [176]. Mathematical optimization over conflicting QoS parameters such as publication rate and buffer depth provides a further tractable path to reducing packet loss and memory pressure without modifying the internal data exchange logic of the middleware [177]. These contributions improve configuration correctness and reduce semantic inconsistency within the middleware boundary, but they do not resolve the underlying enforcement gap: QoS intent formulated at the application layer still fails to propagate into OS kernel scheduling or physical network priority mechanisms.

Narrowing this gap requires a targeted bridging component that connects middleware-level QoS semantics to lower-layer execution and scheduling mechanisms. Rather than redesigning the middleware architecture, this component would either translate existing ROS 2 QoS parameters into a form that OS schedulers and network interface queues can natively interpret, or intercept middleware-level policy expressions and forward them to the appropriate lower-layer primitives such as Linux traffic control, CPU scheduling classes, or hardware offload queues. Such a connector would allow temporal constraints formulated at the application layer to be enforced continuously down the communication stack without requiring changes to the RMW interface or the application programming model. Developing and standardizing this bridging layer within the ROS 2 ecosystem represents a concrete and tractable near-term improvement that does not presuppose a fundamental redesign of the core middleware architecture.

2) Evaluation Inadequacy: Evaluation inadequacy refers to the failure of current evaluation methodologies to capture the emergent degradation that arises under realistic distributed deployment conditions. Isolated component benchmarks consistently fail to surface the latency fluctuations and network saturation that characterize operational robotic systems, because they rely on static payloads or uncongested traffic conditions that conceal the architectural costs of spatial abstraction [96], [105], [112]. Real-time scheduling and intra-process optimization studies conducted without realistic network contention produce characterizations that do not transfer to multi-host wireless deployments [178], [179]. This pattern is not attributable to individual experimental choices but reflects a structural deficiency in how middleware evaluation is currently scoped [180], [181].

The structural origin of this gap lies in the disjointed operational domains of physics-based robot simulators and discrete-event network simulators, which prevents the Perception–Action–Communication loop from being evaluated as an integrated whole. Because the two simulation domains advance time independently and apply different abstraction models to shared events, the emergent interactions among discovery traffic, reliability retransmission, and application data exchange cannot be faithfully reproduced within either domain alone. This mismatch does not require a redesign of the ROS 2 middleware architecture to address, as the evaluation infrastructure surrounding the middleware can be improved independently within the current design space [150], [182]–[184].

Prior work has proposed co-simulation and domain adaptation approaches that improve evaluation fidelity without modifying the middleware itself. Synchronized co-simulation frameworks enforce consistent temporal boundaries across heterogeneous simulation components, and modular wrappers around existing robotic and network simulators substantially improve the fidelity of emulated traffic behavior. Deep learning-based domain adaptation provides a tractable path to sim-to-real transfer by aligning estimated network parameters with observed physical behavior, reducing the divergence between simulated evaluation results and operational outcomes [185], [186]. These contributions improve evaluation fidelity within individual simulation domains, but they do not resolve the underlying integration gap: the emergent interactions among discovery traffic, reliability retransmission, and application data exchange still cannot be characterized as a unified whole under realistic scale and contention.

Narrowing this gap requires a targeted integration of robotic and network simulation environments into unified testbeds that reflect the scale and contention characteristics of physical deployments. Open multi-robot testbeds capable of exercising publish-subscribe middleware under varying traffic densities and node counts would allow the scalability boundaries imposed by discovery storms and control traffic overhead to be quantified before physical deployment [89], [125], [187]. Memory-efficient discovery protocols and filtered state propagation mechanisms should be systematically included in these environments to ensure that state management costs are characterized alongside data exchange performance [163], [165]. Developing and standardizing such integrated evaluation infrastructure within the ROS 2 ecosystem represents a concrete and tractable near-term improvement that does not presuppose a fundamental redesign of the core middleware architecture.

3) Security Overhead: Security overhead refers to the structural incompatibility between security enforcement and temporal predictability under realistic deployment conditions. Enabling mandatory security plugins, including encryption and authentication, introduces latency and throughput degradation that directly competes with time-critical data exchange [188], [189]. Heterogeneous implementations further exhibit interoperability failures and disproportionate computational overhead when strict security policies are enforced at the transport boundary [190], [191]. Cryptographic operations do not impose a fixed additive cost but instead interact with the same execution and network resources that temporal predictability depends upon, compounding timing nondeterminism under load [192].

The structural origin of this gap lies in the decoupling of security verification from the middleware's core discovery and state management dynamics. Unprotected discovery services and software-level policy evaluation expose the system to replay attacks and identity spoofing, and the resulting need for redundant cryptographic checks amplifies resource consumption at precisely the moments when discovery and liveliness traffic are already competing for shared capacity [193], [194]. The disjointed management of security artifacts further allows adversarial entities to exploit the gap between logical permissions and physical execution, bypassing access controls through the same anonymity mechanisms that location transparency relies upon [195], [196]. This mismatch does not require a redesign of the spatial abstraction layer to address, as the overhead originates in how cryptographic operations are integrated into the execution path rather than in the communication model itself, and the surrounding security infrastructure can be improved independently within the current design space [197].

Prior work has proposed policy provisioning and cryptographic execution approaches that reduce overhead without modifying the core middleware architecture. Procedurally generated access control policies can eliminate manual misconfigurations and reduce the computational footprint of authorization checks during endpoint matching, without altering the QoS or discovery mechanisms that govern communication behavior [198]. Delegating cryptographic operations to trusted execution environments isolates verification workloads from the primary communication pipeline, preventing security enforcement from consuming the execution resources required by latency-sensitive callbacks [199]. Trusted Platform Module (TPM)-based remote attestation integrated into the existing handshake protocol further provides continuous hardware-backed validation of participant identity, structurally preventing spoofing while preserving the location transparency on which spatial abstraction depends [200]. At the architectural level, Zenoh adopts a boundary-security model where security is enabled selectively at chosen boundaries rather than uniformly across all communication, so that the throughput and latency cost of cryptographic operations applies only to traffic that requires it [45]. These contributions reduce the per-operation cost of security enforcement, but they do not resolve the underlying coupling gap: cryptographic verification remains structurally entangled with the same execution and network resources that discovery, liveliness maintenance, and data exchange depend upon.

Narrowing this gap requires a targeted mechanism that structurally separates security verification from the latency-critical communication path. Rather than redesigning the middleware architecture, this mechanism would intercept authentication and attestation operations and delegate them to a dedicated hardware-backed execution context, so that cryptographic overhead no longer competes with time-sensitive callbacks on shared execution resources. Such a separation would allow security guarantees to be maintained continuously without consuming the bounded execution capacity that real-time data exchange depends upon. Developing and standardizing this separation mechanism within the ROS 2 security architecture represents a concrete and tractable near-term improvement that does not presuppose a fundamental redesign of the core middleware architecture.

Although each gap arises in a distinct operational domain, they are not independent. QoS semantic loss prevents middleware-level policies from being faithfully enforced at the transport and OS layers, and evaluation inadequacy conceals the emergent consequences of this enforcement failure under realistic deployment conditions. Evaluation inadequacy further prevents security overhead from being accurately characterized, because isolated benchmarks conducted without realistic traffic contention cannot reproduce the timing nondeterminism that cryptographic operations introduce under load. Security overhead compounds both by introducing additional resource contention that interacts with the same execution and network resources that temporal predictability depends upon. Addressing any one of these gaps in isolation therefore provides incomplete relief; progress toward robust middleware deployment requires that all three be pursued in parallel.

B. Research Roadmap

The research directions presented in this section target the root cause of the dimensional conflicts, namely the shared execution boundary and structural coupling among the three dimensions that render their simultaneous satisfaction structurally constrained under shared resource bounds. Each direction directly addresses a specific structural conflict identified in Section V, and together they outline a long-term trajectory toward middleware architectures that renegotiate, rather than merely navigate, the trade-off surface defined by *Space*, *Time*, and *State*.

1) Cross-Layer Co-Design: The ROS 2 communication stack is hierarchically structured across user space, RCL, RMW, middleware, and the kernel OS, with information flowing strictly in one direction [33]. Upper layers issue configurations and commands to lower layers, but the runtime state of these underlying components is never fed back to the application level. The middleware therefore remains unaware of OS-level scheduling dynamics and physical network conditions, while applications cannot observe internal middleware states [201], [202]. This structural unidirectionality is the fundamental source of the Space–Time conflict, as QoS parameters lose their semantic enforcement across these decoupled boundaries [203].

Addressing this limitation requires a cross-layer co-design approach that enables bidirectional interaction across the

communication stack [204]. The core of this approach is the bidirectional integration between the RCL and the middleware via the RMW, so that middleware states are dynamically propagated upward while the execution context of the RCL actively shapes middleware behavior. The interaction between the kernel OS and the middleware must rely on indirect state estimation rather than direct coupling [205]. This strategy allows the middleware to recognize physical infrastructure constraints without compromising the location transparency that the Space dimension requires [206]. This direction focuses specifically on the upward surfacing of middleware states and the indirect estimation of lower-layer conditions.

Several open problems must be addressed before this bidirectional paradigm can be realized in operational robotic middleware. Expanding the RMW interface for bidirectional state exchange must preserve backward compatibility with established application programming models [207]. Estimation mechanisms that infer kernel OS conditions without directly exposing low-level system states require principled design and standardization before they can be deployed reliably across heterogeneous platforms. The continuous monitoring and state-sharing processes inherent to bidirectional interaction impose additional execution and network load, and robust methodologies are needed to ensure that this overhead does not itself degrade the temporal predictability of time-critical control loops [208]. Resolving these problems is a prerequisite for middleware infrastructure that structurally closes the unidirectional information gap across the ROS 2 communication stack.

The fundamental difficulty of cross-layer co-design lies in the fact that the unidirectionality it seeks to overcome is not an accidental design choice but a deliberate architectural boundary that preserves modularity, portability, and vendor independence across the ROS 2 ecosystem. Introducing upward state propagation through the RMW interface risks exposing middleware-internal representations to the RCL, creating implicit coupling between layers that the RMW abstraction was specifically designed to prevent. Similarly, indirect OS state estimation requires the middleware to maintain persistent runtime models of lower-layer behavior, which must remain accurate across heterogeneous hardware platforms, OS configurations, and network interface drivers without direct access to the state being estimated. The tension between the architectural openness required for cross-layer feedback and the strict boundary enforcement required for middleware portability means that any realization of this direction must resolve a structural conflict that is internal to the ROS 2 design philosophy itself.

2) Adaptive Middleware: The structural conflict between the Space and Time dimensions isolates logical endpoint abstractions from physical network states, preventing the middleware execution layer from perceiving underlying link fluctuations and congestion. Complete decoupling of the application from the deployment topology removes the opportunity for the middleware to adapt its transmission behavior to deteriorating network conditions [209]. Attempts to resolve these inefficiencies exclusively at the application layer are structurally insufficient, as the application lacks visibility into lower-level network conditions [210]. This rigid separation leaves distributed components unable to reliably cope with the intermittent connectivity and fluctuating resource availability characteristic of dynamic edge environments [211].

Addressing this limitation without dismantling the benefits of spatial abstraction requires an adaptive strategy that captures physical constraints with minimal latency and computational overhead. Rather than directly exposing raw physical network states to the execution layer, adaptive middleware must continuously estimate underlying link capacity using observable, lightweight indicators [212]. Monitoring accessible endpoint metrics such as acknowledgment timing, message delivery age, and internal queue delays allows the system to indirectly infer physical constraints and congestion boundaries without violating location transparency. Based on these context-aware estimations, the middleware can adapt its transmission behavior and failover mechanisms to sustain temporal predictability without introducing rigid cross-layer dependencies [213].

Several open problems must be resolved before adaptive middleware can be operationalized in distributed robotic deployments. Achieving a deterministic integration of fault-tolerance mechanisms with real-time scheduling remains challenging, as dynamic recovery procedures frequently disrupt strict temporal deadlines [214]. Formalizing consistency models that tolerate bounded state divergence without destabilizing closed-loop control remains a further necessity, particularly for deployments where intermittent connectivity prevents liveliness and history restoration mechanisms from operating under their design assumptions [215]. Resolving these problems requires novel cross-dimensional algorithms that quantitatively model the trade-offs among latency, replication overhead, and network topology, establishing a principled foundation for middleware that adapts across the *Space*, *Time*, and *State* dimensions simultaneously rather than optimizing each in isolation.

The core difficulty of adaptive middleware lies in the constraint that physical inference must be performed using only information that is already visible at the middleware boundary, without penetrating the spatial abstraction that the middleware is obligated to preserve. Observable endpoint metrics such as acknowledgment timing and queue delays are indirect proxies that reflect physical conditions only under specific traffic patterns, and their predictive accuracy degrades precisely under the congestion and link variability conditions where adaptation is most urgently needed. Furthermore, the adaptation logic itself consumes execution resources on the same threads and queues that carry time-critical callbacks, meaning that the overhead of estimating and responding to physical constraints competes directly with the temporal predictability it is intended to protect. Any adaptive strategy must therefore operate within a narrow execution budget that shrinks as network conditions deteriorate, imposing a self-limiting constraint on the sophistication of the inference and adaptation mechanisms that can be practically deployed.

3) Decoupled State Architecture: The Space–State conflict arises because discovery and state management mechanisms are structurally coupled to the same resources required for temporal predictability, preventing spatial scalability and state continuity from being simultaneously sustained within the current architec-

ture [216], [217]. Fully meshed peer-to-peer topologies carry an $O(N^2)$ metadata exchange cost, while centralized architectures concentrate the state management burden at a single point of failure [218], [219]. The continuous metadata exchange in distributed topologies directly competes with latency-sensitive data transmission, whereas centralized intermediaries introduce potential network bottlenecks [220]. This architectural dichotomy establishes a structural tension that remains unresolved within the current middleware design space [221].

Addressing this tension without permanently committing to either extreme requires an adaptive hybrid state architecture that maintains a centralized structure for baseline efficiency while falling back to decentralized mechanisms under varying failure and scaling conditions [222], [223]. During stable operation, this approach exploits the low computational and network overhead of centralized brokers, mitigating the discovery traffic typically associated with dynamic node participation [224], [225]. When the system encounters intermediary broker failures, severe network partitions, or abrupt surges in node density, the architecture transitions to a peer-to-peer mechanism to preserve contextual continuity [197]. This structural flexibility allows the middleware to combine the spatial efficiency of hierarchical control with the fault tolerance of distributed state retention [226], [227].

Several open problems must be addressed before a decoupled state architecture can be operationalized in distributed robotic deployments. Minimizing transition costs and establishing precise criteria for triggering the fallback from centralized to distributed modes remain open design questions [228], [229]. Reintegrating the distributed state accumulated during the fallback phase into a unified centralized structure upon recovery requires mechanisms that prevent data conflicts and logical inconsistencies [230], [231]. Rapid structural shifts between the two operational modes must not disrupt or delay time-critical communication sessions already in progress across partitioned domains [232]. Resolving these problems requires deeper integration of dynamic reconfiguration logic with underlying transport interfaces, so that location transparency is preserved across volatile network boundaries [233].

Recent developments illustrate partial progress in this direction. Zenoh introduces a Regions architecture that replaces the fixed router, peer, and client tiers with arbitrary-depth nested topologies and operator-defined gateways, structurally addressing the static-topology dimension of the problem [234]. The dynamic case that this direction targets remains open, since runtime transition between centralized and peer-to-peer operation under broker failure or severe partition still triggers a discovery burst from a state of mutual ignorance, and the divergent state accumulated during the fallback interval must still be reconciled on recovery. These are open challenges across distributed-systems research, not unique to any single middleware.

The core difficulty of the hybrid fallback architecture lies in the asymmetry between the two operational modes it must support. Centralized operation accumulates a globally consistent state view at the broker, but this view is not replicated across participants, meaning that a fallback to peer-to-peer mode begins from a state of mutual ignorance among nodes that have never maintained direct visibility of one another. Reconstructing sufficient mutual awareness to sustain communication during the fallback phase requires a burst of discovery traffic at precisely the moment when network conditions are most stressed, reproducing a bounded version of the discovery storm dynamics identified in Section V. The transition back to centralized operation introduces a symmetric problem: the broker must reconcile the divergent state accumulated across distributed participants during the fallback interval, and the cost of this reconciliation scales with both the duration of the fallback and the rate of topology change during that period.

Taken together, these three directions represent a qualitative departure from the mitigation strategies surveyed in Section V. Where existing approaches improved one dimension by partially relinquishing another, each roadmap direction attempts to restructure the underlying mechanism so that both dimensions of the targeted conflict can be addressed without forcing an explicit sacrifice. Cross-layer co-design pursues bidirectional state visibility without dismantling the modularity boundaries that spatial abstraction depends upon. Adaptive middleware pursues physical constraint awareness without exposing deployment topology to the application layer. Decoupled state architecture pursues the efficiency of centralization without permanently surrendering the fault tolerance of distributed operation. None of these directions eliminates the structural tension identified in Section V; the shared resource bounds that produce dimensional conflict remain a physical reality. What they collectively offer is a principled basis for operating closer to the boundary of what those resource bounds permit, rather than accepting the conservative compromises that characterize current deployments. Realizing this potential requires sustained research investment across middleware design, OS integration, and distributed systems theory, guided by the evaluation infrastructure and cross-layer modeling tools.

VII. Conclusion

This paper has presented a systematic survey of ROS 2 middleware through the lens of three structural dimensions, namely *Space*, *Time*, and *State*. We first analyzed the architectural constructs and operational dynamics of ROS 2 middleware, encompassing discovery, data exchange, and state management mechanisms across DDS and Zenoh. Building on this foundation, we formalized *Space* as the requirement for location transparency, *Time* as the requirement for temporal predictability, and *State* as the requirement for contextual continuity. Through a comprehensive review of 94 middleware-focused studies, we mapped the structural trade-offs among these dimensions into three conflict pairs, Space–Time, Time–State, and Space–State, and identified eleven distinct manifestations through which these conflicts propagate across middleware, OS, and physical layers.

A central finding of this survey is that there is no silver bullet for robotic middleware design. Every mitigation strategy examined across the eleven manifestations improves one dimension by partially relinquishing the guarantees of another. Location transparency imposes a cost on temporal

performance by concealing physical network state from the middleware. Temporal predictability can only be preserved by relaxing the reliability guarantees that contextual continuity depends upon. Contextual continuity, in turn, demands physical identity information that spatial abstraction is designed to withhold. Because all three dimensions compete for the same bounded shared resources, namely execution capacity, network bandwidth, and buffer memory, satisfying any one dimension fully exerts pressure on at least one of the others, and any proposed mitigation navigates the trade-off surface rather than escaping it. Middleware designers and practitioners must therefore accept that deploying a robotic system always entails an explicit or implicit choice of which dimensional guarantee to prioritize and which to partially sacrifice, and this choice should be made deliberately rather than by default.

Looking ahead, the primary research challenge is not to escape the trade-off surface but to navigate it more efficiently, by designing middleware architectures that maximize the achievable guarantees across all three dimensions simultaneously within the constraints imposed by shared resources. The roadmap directions proposed in this paper, namely cross-layer co-design, adaptive middleware, and state architecture, each represent a principled step toward shrinking the region of the trade-off space in which current systems operate, rather than merely shifting cost from one dimension to another. This challenge is acquiring renewed urgency in the era of Physical AI, where vision-language-action models, embodied agents, and large-scale multi-robot coordination are driving an unprecedented increase in the volume, heterogeneity, and latency-sensitivity of data exchanged over robotic networks. As the communication demands of intelligent robotic systems continue to expand, the architectural choices embedded in middleware design will increasingly determine whether distributed robotic platforms can operate reliably and responsively in the open-world environments they are expected to inhabit.

REFERENCES

[1] Open Robotics, "ROS metrics," [Online]. Available: http://metrics.ros.org/, 2026.

[2] K. Scott and T. Foote, "2025 ROS metrics report," Open Source Robotics Foundation (OSRF), Tech. Rep., 2025. [Online]. Available: https://discourse.openrobotics.org/t/2025-ros-metrics-report/52575

[3] J. Zhang, F. Keramat, X. Yu, D. M. Hernández, J. P. Queralta, and T. Westerlund, "Distributed robotic systems in the edge-cloud continuum with ROS 2: A review on novel architectures and technology readiness," in *Proc. 2022 Seventh Int. Conf. on Fog and Mobile Edge Computing (FMEC)*. IEEE, 2022, pp. 1–8.

[4] L. Fu, G. Kapoor, L. Militano, G. T. Carughi, and T. M. Bohnert, "IoRT ROS 2 applications: Evaluating Zenoh and VPN for robotic networking in the edge-cloud continuum," in *Proc. 2025 IEEE Symp. on Computers and Communications (ISCC)*. IEEE, 2025, pp. 1–6.

[5] X. An, C. Wu, Y. Lin, M. Lin, T. Yoshinaga, and Y. Ji, "Multi-robot systems and cooperative object transport: Communications, platforms, and challenges," *IEEE Open J. of the Computer Society*, vol. 4, pp. 23–36, 2023.

[6] 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.

[7] J. M. Cruz, A. Romero-Garcés, J. P. B. Rubio, R. M. Robles, and A. B. Rubio, "A DDS-based middleware for quality-of-service and high-performance networked robotics," *Concurrency and Computation: Practice and Experience*, vol. 24, no. 16, pp. 1940–1952, 2012.

[8] B. Al-Madani, S. Hasan, A. Abualhassan, and F. Aliyu, "Integrating Data Distribution Service (DDS) in smart traffic systems: A comprehensive review," *Computer*, vol. 58, no. 2, pp. 25–34, 2025.

[9] J.-B. Castillo-Sánchez, E. González-Parada, and J.-M. Cano-García, "A novel testbed for evaluating ROS 2 robot swarm wireless communications," in *Proc. 2024 IEEE 22nd Mediterranean Electrotechnical Conf. (MELECON)*. IEEE, 2024, pp. 68–73.

[10] S. Agarwal, A. Ribeiro, and V. Kumar, "A scalable multi-robot framework for decentralized and asynchronous perception-action-communication loops," *arXiv preprint arXiv:2309.10164*, 2025.

[11] S. Kong, H. Zhang, C. Ye, Y. Xu, and A. Li, "Design of a multimodal control system based on ROS 2: A hierarchical architecture for real-time human-robot collaboration," in *Proc. 2025 7th Int. Conf. on Data-driven Optimization of Complex Systems (DOCS)*. IEEE, 2025, pp. 441–446.

[12] M. Groshev, G. Baldoni, L. Cominardi, A. De La Oliva, and R. Gazda, "Edge robotics: Are we ready? an experimental evaluation of current vision and future directions," *Digital Communications and Networks*, vol. 9, no. 1, pp. 166–174, 2023.

[13] A. Al-Batati, A. Koubaa, K. Gabr, M. Abdelkader, and H. Aloquaily, "ROS 2 in a nutshell: A survey," *ACM Computing Surveys*, 2025.

[14] L. Naury, A. Gouguet, G. Lozenguez, and L. Fabresse, "Communication isolation for multi-robot systems using ROS 2," in *Proc. 40th ACM/SIGAPP Symp. on Applied Computing*, 2025, pp. 850–858.

[15] L. Hietasalo, "Overview and performance analysis of robotic software framework Flexbot," Master's thesis, Tampere University, 2024.

[16] D. Casini, T. Blaß, I. Lütkebohle, and B. Brandenburg, "Response-time analysis of ROS 2 processing chains under reservation-based scheduling," in *Proc. 31st Euromicro Conf. on Real-Time Systems*. Schloss Dagstuhl, 2019, pp. 1–23.

[17] H. Teper, T. Betz, M. Günzel, D. Ebner, G. Von Der Brüggen, J. Betz, and J.-J. Chen, "End-to-end timing analysis and optimization of multi-executor ROS 2 systems," in *Proc. 2024 IEEE 30th Real-Time and Embedded Technology and Applications Symp. (RTAS)*. IEEE, 2024, pp. 212–224.

[18] Y. Tang, N. Guan, X. Jiang, X. Luo, and W. Yi, "Real-time performance analysis of processing systems on ROS 2 executors," in *Proc. 2023 IEEE 29th Real-Time and Embedded Technology and Applications Symp. (RTAS)*. IEEE, 2023, pp. 80–92.

[19] Y. Tang, Z. Feng, N. Guan, X. Jiang, M. Lv, Q. Deng, and W. Yi, "Response time analysis and priority assignment of processing chains on ROS 2 executors," in *Proc. 2020 IEEE Real-Time Systems Symp. (RTSS)*. IEEE, 2020, pp. 231–243.

[20] L. Dust and S. Mubeen, "Dynamic priority scheduling for periodic systems using ROS 2," in *Proc. Int. Conf. on Engineering of Computer-Based Systems*. Springer, 2023, pp. 239–243.

[21] S. Liu, X. Jiang, N. Guan, Z. Wang, M. Yu, and W. Yi, "RTeX: An efficient and timing-predictable multithreaded executor for ROS 2," *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, vol. 43, no. 9, pp. 2578–2591, 2024.

[22] J. Staschulat, I. Lütkebohle, and R. Lange, "The rclc executor: Domain-specific deterministic scheduling mechanisms for ROS applications on microcontrollers: Work-in-progress," in *Proc. 2020 Int. Conf. on Embedded Software (EMSOFT)*. IEEE, 2020, pp. 18–19.

[23] Y. Yang and T. Azumi, "Exploring real-time executor on ROS 2," in *Proc. 2020 IEEE Int. Conf. on Embedded Software and Systems (ICESS)*. IEEE, 2020, pp. 1–8.

[24] D. Enright, Y. Xiang, H. Choi, and H. Kim, "PAAM: A framework for coordinated and priority-driven accelerator management in ROS 2," in *Proc. 2024 IEEE 30th Real-Time and Embedded Technology and Applications Symp. (RTAS)*. IEEE, 2024.

[25] H. Choi, Y. Xiang, and H. Kim, "PiCAS: New design of priority-driven chain-aware scheduling for ROS 2," in *Proc. 2021 IEEE 27th Real-Time and Embedded Technology and Applications Symp. (RTAS)*. IEEE, 2021, pp. 251–263.

[26] T. Xu, X. Chen, C. Wu, J. Wang, R. Zheng, D. Liu, Y. Tan, A. Ren, and J. Li, "3DS: An efficient DPDK-based Data Distribution Service for distributed real-time applications," in *Proc. 2022 IEEE 24th Int. Conf. on High Performance Computing & Communications (HPCC/DSS/SmartCity/DependSys)*. IEEE, 2022, pp. 1283–1290.

[27] T. Blass, A. Hamann, R. Lange, D. Ziegenbein, and B. B. Brandenburg, "Automatic latency management for ROS 2: Benefits, challenges, and open problems," in *Proc. 2021 IEEE 27th Real-Time and Embedded Technology and Applications Symp. (RTAS)*. IEEE, 2021, pp. 264–277.

[28] L. Qi, X. Zhang, H. Tan, H. Chen, and Y. Wang, "A novel open and efficient robot development framework based on Data Distribution Service orchestration for agile manufacturing," *Robotics and Computer-Integrated Manufacturing*, vol. 96, p. 103067, 2025.

[29] G. Baldoni, J. Loudet, L. Cominardi, A. Corsaro, and Y. He, "Facilitating distributed data-flow programming with Eclipse Zenoh: The ERDOS

case," in *Proc. 1st Workshop on Serverless Mobile Networking for 6G Communications*, 2021, pp. 13–18.
[30] S. Lee, J. Chae, and K.-J. Park, "Harness engineering for Physical AI: Robot middleware is the harness layer," *arXiv preprint arXiv:2606.09416*, 2026.
[31] Z. Li, A. Hasegawa, and T. Azumi, "Autoware_Perf: A tracing and performance analysis framework for ROS 2 applications," *J. of Systems Architecture*, vol. 123, p. 102341, 2022.
[32] A. Bonci, F. Gaudeni, M. C. Giannini, and S. Longhi, "Robot Operating System 2 (ROS 2)-based frameworks for increasing robot autonomy: A survey," *Applied Sciences*, vol. 13, no. 23, p. 12796, 2023.
[33] Open Robotics, "Internal ROS 2 interfaces," [Online]. Available: https://docs.ros.org/en/kilted/Concepts/Advanced/About-Internal-Interfaces.html, accessed: Mar. 28, 2026.
[34] ——, "Different ROS 2 middleware vendors," [Online]. Available: https://docs.ros.org/en/kilted/Concepts/Intermediate/About-Different-Middleware-Vendors.html, accessed: Mar. 28, 2026.
[35] ——, "ROS 2 middleware implementations," [Online]. Available: https://docs.ros.org/en/kilted/Concepts/Advanced/About-Middleware-Implementations.html, accessed: Mar. 28, 2026.
[36] ——, "Changes between ROS 1 and ROS 2," [Online]. Available: https://design.ros2.org/articles/changes.html, accessed: Mar. 28, 2026.
[37] ——, "Creating an RMW implementation," [Online]. Available: https://docs.ros.org/en/kilted/Tutorials/Advanced/Creating-An-RMW-Implementation.html, accessed: Mar. 28, 2026.
[38] ——, "Executors," [Online]. Available: https://docs.ros.org/en/kilted/Concepts/Intermediate/About-Executors.html, accessed: Mar. 28, 2026.
[39] P. A. Bernstein, "Middleware: A model for distributed system services," *Communications of the ACM*, vol. 39, no. 2, pp. 86–98, 1996.
[40] W. Emmerich, "Software engineering and middleware: A roadmap," in *Proc. Conf. on the Future of Software Engineering*, 2000, pp. 117–129.
[41] Open Robotics, "ROS 2 Kilted Kaiju released," [Online]. Available: https://www.openrobotics.org/blog/2025/5/23/ros-2-kilted-kaiju-released, accessed: Mar. 28, 2026.
[42] G. Pardo-Castellote, "OMG Data Distribution Service: Architectural overview," in *Proc. 23rd Int. Conf. on Distributed Computing Systems Workshops*. IEEE, 2003, pp. 200–206.
[43] ZettaScale Technology, "What is Zenoh?" [Online]. Available: https://zenoh.io/docs/overview/what-is-zenoh/, accessed: Mar. 28, 2026.
[44] Open Robotics, "ROS 2 alternative middleware report," Open Robotics, Tech. Rep., Sep. 2023, accessed: 2026-03-11. [Online]. Available: https://discourse.ros.org/t/ros-2-alternative-middleware-report/33771
[45] A. Corsaro, L. Cominardi, O. Hecart, G. Baldoni, J. E. P. Avital, J. Loudet, C. Guimares, M. Ilyin, and D. Bannov, "Zenoh: Unifying communication, storage and computation from the cloud to the microcontroller," in *Proc. 2023 26th Euromicro Conf. on Digital System Design (DSD)*. IEEE, 2023, pp. 422–428.
[46] ZettaScale Technology, "Zenoh abstractions," [Online]. Available: https://zenoh.io/docs/manual/abstractions/, accessed: Mar. 28, 2026.
[47] Open Robotics, "rmw_zenoh," [Online]. Available: https://github.com/ros2/rmw_zenoh/tree/kilted, gitHub repository, Accessed: Mar. 28, 2026.
[48] M. Pöhnl, C. Eltzschig, and T. Blass, "Shared-memory-based lock-free queues: The key to fast and robust communication on safety-critical edge devices," in *Proc. Cyber-Physical Systems and Internet of Things Week 2023*, 2023, pp. 179–184.
[49] B. Stadnik and A. Wymysłowski, "Overview analysis of Micro-ROS system as an embedded solution for microcontrollers in automatics and robotics applications," *Przegląd Elektrotechniczny*, vol. 100, 2024.
[50] N. Parmar, V. Ranga, and B. Simhachalam Naidu, "Syntactic interoperability in real-time systems, ROS 2, and adaptive AUTOSAR using Data Distribution Services: An approach," in *Inventive Communication and Computational Technologies: Proc. ICICCT 2019*. Springer, 2020, pp. 257–274.
[51] D. Casini, J.-J. Chen, J. Li, F. Reghenzani, and H. Teper, "A survey of real-time support, analysis, and advancements in ROS 2," *Leibniz Trans. on Embedded Systems (LITES)*, vol. 11, no. 1, pp. 1:1–1:37, 2026.
[52] B. Peng, A. Hasegawa, and T. Azumi, "Scheduling performance evaluation framework for ROS 2 applications," in *Proc. 2022 IEEE 24th Int. Conf. on High Performance Computing & Communications (HPCC/DSS/SmartCity/DependSys)*. IEEE, 2022, pp. 2031–2038.
[53] J. Kouril, B. Schäufele, I. Radusch, and B. Schnor, "Performance evaluation of a ROS 2 based automated driving system," in *Proc. 10th Int. Conf. on Vehicle Technology and Intelligent Transport Systems (VEHITS)*. SciTePress, 2024, pp. 52–63.
[54] Open Robotics, "Topics vs services vs actions," [Online]. Available: https://docs.ros.org/en/kilted/How-To-Guides/Topics-Services-Actions.html, accessed: Mar. 28, 2026.
[55] ——, "Understanding topics," [Online]. Available: https://docs.ros.org/en/kilted/Tutorials/Beginner-CLI-Tools/Understanding-ROS2-Topics/Understanding-ROS2-Topics.html, accessed: Mar. 28, 2026.
[56] ——, "Understanding services," [Online]. Available: https://docs.ros.org/en/kilted/Tutorials/Beginner-CLI-Tools/Understanding-ROS2-Services/Understanding-ROS2-Services.html, accessed: Mar. 28, 2026.
[57] ——, "Understanding actions," [Online]. Available: https://docs.ros.org/en/kilted/Tutorials/Beginner-CLI-Tools/Understanding-ROS2-Actions/Understanding-ROS2-Actions.html, accessed: Mar. 28, 2026.
[58] ——, "Interfaces," [Online]. Available: https://docs.ros.org/en/kilted/Concepts/Basic/About-Interfaces.html, accessed: Mar. 28, 2026.
[59] K. Krinkin, A. Filatov, A. Filatov, O. Kurishev, and A. Lyanguzov, "Data Distribution Services performance evaluation framework," in *Proc. 2018 22nd Conf. of Open Innovations Association (FRUCT)*. IEEE, 2018, pp. 94–100.
[60] F. J. Blanco Romero, "Enhancing communication security in ROS 2," Master's thesis, Universidad Miguel Hernández de Elche, 2024.
[61] Z. Jiang, Y. Gong, J. Zhai, Y.-P. Wang, W. Liu, H. Wu, and J. Jin, "Message passing optimization in Robot Operating System," *Int. J. of Parallel Programming*, vol. 48, no. 1, pp. 119–136, 2020.
[62] eProsima, "ROS 2 discovery server — Fast DDS documentation," [Online]. Available: https://fast-dds.docs.eprosima.com/en/latest/fastdds/ros2/discovery_server/ros2_discovery_server.html, 2024, accessed: 2026-04-03.
[63] Real-Time Innovations (RTI), "RTI routing service — RTI Connext DDS Professional 7.3.0 documentation," [Online]. Available: https://community.rti.com/static/documentation/connext-dds/7.3.0/doc/manuals/connext_dds_professional/services/routing_service/index.html, 2024, accessed: 2026-04-03.
[64] H. A. Putra and D.-S. Kim, "Node discovery scheme of DDS for combat management system," *Computer Standards & Interfaces*, vol. 37, pp. 20–28, 2015.
[65] Object Management Group, *The real-time publish-subscribe protocol DDS interoperability wire protocol (DDSI-RTPS) specification, version 2.5*, Object Management Group, Apr. 2022, OMG Document Number: formal/2022-04-01. [Online]. Available: https://www.omg.org/spec/DDSI-RTPS/2.5
[66] F. Maggi, R. Vosseler, M. Cheng, P. Kuo, C. Toyama, T. Yen, and E. B. V. Vilches, "A security analysis of the Data Distribution Service (DDS) protocol," *Trend Micro Research*, pp. 15–20, 2022.
[67] A. Zanni, "Efficient support for deep sleeping modes in embedded systems: The case of Zenoh-pico," Master's thesis, University of Bologna, 2024.
[68] G. Caruso, "Autonomous navigation for quadruped robots: Development and optimization on the Unitree Go1 platform," Ph.D. dissertation, Politecnico di Torino, 2024.
[69] M. J. Michaud, T. Dean, and S. P. Leblanc, "Attacking OMG Data Distribution Service (DDS) based real-time mission critical distributed systems," in *Proc. 2018 13th Int. Conf. on Malicious and Unwanted Software (MALWARE)*. IEEE, 2018, pp. 68–77.
[70] M. Baron, L. Diez, M. Zverev, J. R. Juarez, and R. Agüero, "On the performance of Zenoh in industrial IoT scenarios," *Ad Hoc Networks*, vol. 170, p. 103784, 2025.
[71] H. P. Tijero and J. J. Gutiérrez, "On the schedulability of a data-centric real-time distribution middleware," *Computer Standards & Interfaces*, vol. 34, no. 1, pp. 203–211, 2012.
[72] C. Scordino, A. G. Mariño, and F. Fons, "Hardware acceleration of Data Distribution Service (DDS) for automotive communication and computing," *IEEE Access*, vol. 10, pp. 109 626–109 651, 2022.
[73] W.-Y. Liang, Y. Yuan, and H.-J. Lin, "A performance study on the throughput and latency of Zenoh, MQTT, Kafka, and DDS," *arXiv preprint arXiv:2303.09419*, 2023.
[74] F. Desbiens, "Zenoh: A next-generation protocol for IoT and edge computing," Presentation at Open Source Summit + Embedded Linux Conf. (OSS + ELC) 2021, Eclipse Foundation, Sep. 2021.
[75] A. Hakiri, P. Berthou, A. Gokhale, D. C. Schmidt, and T. Gayraud, "Supporting end-to-end quality of service properties in OMG Data Distribution Service publish/subscribe middleware over wide area networks," *J. of Systems and Software*, vol. 86, no. 10, pp. 2574–2593, 2013.

[76] A. Corsaro. (2021, Jun.) Zenoh reliability: How it works? [Online]. Available: https://zenoh.io/blog/2021-06-14-zenoh-reliability/. ZettaScale Technology.

[77] ZettaScale Technology. (2023, Jun.) Zenoh for ROS 2: Charmander and beyond. [Online]. Available: https://zenoh.io/blog/2023-06-05-charmander2/.

[78] ——. (2024) Module zenoh::liveliness — Rust documentation. [Online]. Available: https://docs.rs/zenoh/latest/zenoh/liveliness/index.html. Official Rust API Reference.

[79] J. Fernandez, B. Allen, P. Thulasiraman, and B. Bingham, "Performance study of the Robot Operating System 2 with QoS and cyber security settings," in *Proc. 2020 IEEE Int. Systems Conf. (SysCon)*. IEEE, 2020, pp. 1–6.

[80] M. Barciś, A. Barciś, and H. Hellwagner, "Information distribution in multi-robot systems: Utility-based evaluation model," *Sensors*, vol. 20, no. 3, p. 710, 2020.

[81] G. Pardo-Castellote, B. Farabaugh, and R. Warren, "An introduction to DDS and data-centric communications," *Real-Time Innovations*, vol. 61, 2005.

[82] ZettaScale Technology. (2024) Zenoh plugin storage manager. [Online]. Available: https://zenoh.io/docs/manual/plugin-storage-manager/. Accessed: 2026-03-30.

[83] A. Corsaro, "The Data Distribution Service tutorial," *Technical Report 4.0*, 2014.

[84] ZettaScale Technology. (2024) Module zenoh::session — Rust documentation. [Online]. Available: https://docs.rs/zenoh/latest/zenoh/session/index.html. Official Rust API Reference.

[85] ——. (2024) Zenoh deployment guide. [Online]. Available: https://zenoh.io/docs/getting-started/deployment/. Accessed: 2026-03-30.

[86] Object Management Group, *Data Distribution Service (DDS), version 1.4*, Object Management Group, Apr. 2015, formal/2015-04-10. [Online]. Available: https://www.omg.org/spec/DDS/1.4

[87] X. Luo, X. Jiang, N. Guan, H. Liang, S. Liu, and W. Yi, "Modeling and analysis of inter-process communication delay in ROS 2," in *Proc. 2023 IEEE Real-Time Systems Symp. (RTSS)*. IEEE, 2023, pp. 198–209.

[88] T. Kronauer, J. Pohlmann, M. Matthé, T. Smejkal, and G. Fettweis, "Latency analysis of ROS 2 multi-node systems," in *Proc. 2021 IEEE Int. Conf. on Multisensor Fusion and Integration for Intelligent Systems (MFI)*. IEEE, 2021, pp. 1–7.

[89] Z. Kang, R. Canady, A. Dubey, A. Gokhale, S. Shekhar, and M. Sedlacek, "A study of publish/subscribe middleware under different IoT traffic conditions," in *Proc. Int. Workshop on Middleware and Applications for the Internet of Things*, 2020, pp. 7–12.

[90] Y.-P. Wang, Y. Dong, and G. Tan, "ROS-SF: A transparent and efficient ROS middleware using serialization-free message," in *Proc. 23rd ACM/IFIP Int. Middleware Conf.*, 2022, pp. 82–93.

[91] A. Ranjan, A. Damodar, N. Chougule, D. S. Nayak *et al.*, "Meta-ROS: A next-generation middleware architecture for adaptive and scalable robotic systems," *arXiv preprint arXiv:2601.21011*, 2026.

[92] Y.-P. Wang, W. Tan, X.-Q. Hu, D. Manocha, and S.-M. Hu, "TZC: Efficient inter-process communication for robotics middleware with partial serialization," in *Proc. 2019 IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*. IEEE, 2019, pp. 7805–7812.

[93] T. Ishikawa-Aso and S. Kato, "ROS 2 Agnocast: Supporting unsized message types for true zero-copy publish/subscribe IPC," in *Proc. 2025 28th Int. Symp. on Real-Time Distributed Computing (ISORC)*. IEEE, 2025, pp. 1–10.

[94] M. Xiong, J. Parsons, J. Edmondson, H. Nguyen, and D. C. Schmidt, "Evaluating the performance of publish/subscribe platforms for information management in distributed real-time and embedded systems," *omgwiki.org/dds*, 2010.

[95] C.-T. Lin and H.-J. Lu, "An intelligent product-driven manufacturing system using Data Distribution Service," *IEEE Access*, vol. 12, pp. 16 447–16 461, 2024.

[96] Z. Chen, "Performance analysis of ROS 2 networks using variable Quality of Service and security constraints for autonomous systems," Ph.D. dissertation, Naval Postgraduate School, 2019.

[97] S. Lee, H.-S. Park, J. Chae, and K.-J. Park, "Probabilistic latency analysis of the Data Distribution Service in ROS 2," *arXiv preprint arXiv:2508.10413*, 2025.

[98] J. Peeck, M. Möstl, T. Ishigooka, and R. Ernst, "A middleware protocol for time-critical wireless communication of large data samples," in *Proc. 2021 IEEE Real-Time Systems Symp. (RTSS)*. IEEE, 2021, pp. 1–13.

[99] S. Lee, T. Kim, J. Chae, and K.-J. Park, "Poster: How to send large data in ROS 2," in *Proc. 2025 IEEE 33rd Int. Conf. on Network Protocols (ICNP)*. IEEE, 2025, pp. 1–3.

[100] 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, no. 7, pp. 3996–4003, 2023.

[101] H.-Y. Jung, D.-H. Paek, and S.-H. Kong, "Open-source autonomous driving software platforms: Comparison of Autoware and Apollo," *arXiv preprint arXiv:2501.18942*, 2025.

[102] S. Lee, T. Kim, J. Chae, and K.-J. Park, "Optimizing ROS 2 communication for wireless robotic systems," *arXiv preprint arXiv:2508.11366*, 2025.

[103] N. Sperling and R. Ernst, "Low latency communication of large data objects by subscriber-centric selective data transfer," *ACM Trans. on Cyber-Physical Systems*, 2026.

[104] D. P. Klüner, L. Hegerath, A. D. Hatib, S. Kowalewski, B. Alrifaee, and A. Kampmann, "Automotive middleware performance: Comparison of FastDDS, Zenoh and vSomeIP," in *Proc. 2025 IEEE Int. Conf. on Vehicular Electronics and Safety (ICVES)*. IEEE, 2025.

[105] Y. Maruyama, S. Kato, and T. Azumi, "Exploring the performance of ROS 2," in *Proc. 13th Int. Conf. on Embedded Software*, 2016, pp. 1–10.

[106] T. Wu, B. Wu, S. Wang, L. Liu, S. Liu, Y. Bao, and W. Shi, "Oops! it's too late. your autonomous driving system needs a faster middleware," *IEEE Robotics and Automation Letters*, vol. 6, no. 4, pp. 7301–7308, 2021.

[107] K. An, A. Gokhale, D. Schmidt, S. Tambe, P. Pazandak, and G. Pardo-Castellote, "Content-based filtering discovery protocol (CFDP): Scalable and efficient OMG DDS discovery protocol," in *Proc. 8th ACM Int. Conf. on Distributed Event-Based Systems*, 2014, pp. 130–141.

[108] Z. Kang, K. An, A. Gokhale, and P. Pazandak, "A comprehensive performance evaluation of different Kubernetes CNI plugins for edge-based and containerized publish/subscribe applications," in *Proc. 2021 IEEE Int. Conf. on Cloud Engineering (IC2E)*. IEEE, 2021, pp. 31–42.

[109] Y. Ye, Z. Nie, X. Liu, F. Xie, Z. Li, and P. Li, "ROS 2 real-time performance optimization and evaluation," *Chinese J. of Mechanical Engineering*, vol. 36, no. 1, p. 144, 2023.

[110] A.-M. Basem and H. Ali, "Data Distribution Service (DDS) based implementation of smart grid devices using ANSI C12.19 standard," *Procedia Computer Science*, vol. 110, pp. 394–401, 2017.

[111] 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," *Int. J. of Distributed Sensor Networks*, vol. 11, no. 7, p. 863123, 2015.

[112] V. Bode, D. Buettner, T. Preclik, C. Trinitis, and M. Schulz, "Systematic analysis of DDS implementations," in *Proc. 24th Int. Middleware Conf.*, 2023, pp. 234–246.

[113] L. J. Dust, E. Persson, M. Ekström, S. Mubeen, and E. Dean, "Quantitative analysis of communication handling for centralized multi-agent robot systems using ROS 2," in *Proc. 2022 IEEE 20th Int. Conf. on Industrial Informatics (INDIN)*. IEEE, 2022, pp. 624–629.

[114] Y. He and W. Shi, "A faster and more reliable middleware for autonomous driving systems," *arXiv preprint arXiv:2510.11448*, 2025.

[115] L. J. Dust and E. Persson, "Dynamic connection handling for scalable robotic systems using ROS 2," 2022.

[116] M. Pöhnl, A. Tamisier, and T. Blass, "A middleware journey from microcontrollers to microprocessors," in *Proc. 2022 Design, Automation & Test in Europe Conf. & Exhibition (DATE)*. IEEE, 2022, pp. 282–286.

[117] M. De Marchi and N. Bombieri, "Orchestration-aware optimization of ROS 2 communication protocols," in *Proc. 2024 Design, Automation & Test in Europe Conf. & Exhibition (DATE)*. IEEE, 2024, pp. 1–6.

[118] J. Zhang, X. Yu, S. Ha, J. Peña Queralta, and T. Westerlund, "Comparison of middlewares in edge-to-edge and edge-to-cloud communication for distributed ROS 2 systems," *J. of Intelligent & Robotic Systems*, vol. 110, no. 4, pp. 1–20, 2024.

[119] A. Hakiri, P. Berthou, A. Gokhale, D. C. Schmidt, and T. Gayraud, "Supporting SIP-based end-to-end Data Distribution Service QoS in WANs," *J. of Systems and Software*, vol. 95, pp. 100–121, 2014.

[120] A. S. Rhoades, G. Schrader, and P. Poulin, "Benchmarking publish/subscribe middleware for radar applications," in *Proc. Eleventh Annual HPEC Workshop*, 2007.

[121] T. A. Youssef, A. T. Elsayed, and O. A. Mohammed, "DDS-based interoperability framework for smart grid testbed infrastructure," in *Proc. 2015 IEEE 15th Int. Conf. on Environment and Electrical Engineering (EEEIC)*. IEEE, 2015, pp. 219–224.

[122] T. A. Youssef, M. El Hariri, A. T. Elsayed, and O. A. Mohammed, "A DDS-based energy management framework for small microgrid operation and control," *IEEE Trans. on Industrial Informatics*, vol. 14, no. 3, pp. 958–968, 2017.

[123] K. Peeroo, P. Popov, and V. Stankovic, "A survey on experimental performance evaluation of Data Distribution Service (DDS) implementations," *arXiv preprint arXiv:2310.16630*, 2023.

[124] ——, "Exploring the effects of multicast communication on DDS performance," in *Proc. Student Forum of the 18th European Dependable Computing Conf. (EDCC)*, 2022.

[125] S. Jones, E. Milner, M. Sooriyabandara, and S. Hauert, "DOTS: An open testbed for industrial swarm robotic solutions," *arXiv preprint arXiv:2203.13809*, 2022.

[126] K.-h. Lee, C.-k. Kim, K. T. Kim, and W.-t. Kim, "Router design for DDS: Architecture and performance evaluation," in *Proc. 2014 Int. Conf. on Big Data and Smart Computing (BIGCOMP)*. IEEE, 2014, pp. 250–254.

[127] B. Zieba and M. van Sinderen, "Preservation of correctness during system reconfiguration in Data Distribution Service for real-time systems (DDS)," in *Proc. 26th IEEE Int. Conf. on Distributed Computing Systems Workshops (ICDCSW)*. IEEE, 2006, pp. 30–30.

[128] H. Teper, T. Betz, G. Von Der Brüggen, K.-H. Chen, J. Betz, and J.-J. Chen, "Timing-aware ROS 2 architecture and system optimization," in *Proc. 2023 IEEE 29th Int. Conf. on Embedded and Real-Time Computing Systems and Applications (RTCSA)*. IEEE, 2023, pp. 206–215.

[129] S. Dehnavi, M. Koedam, A. Nelson, D. Goswami, and K. Goossens, "CompROS: A composable ROS 2 based architecture for real-time embedded robotic development," in *Proc. 2021 IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*. IEEE, 2021, pp. 6449–6455.

[130] G. Sciangula, D. Casini, A. Biondi, C. Scordino, M. Di Natale *et al.*, "Bounding the data-delivery latency of DDS messages in real-time applications," *Leibniz Int. Proc. in Informatics*, vol. 262, 2023.

[131] S. Kim, J. Song, K. Lee, S. Oh, and H. S. Chwa, "CRoS-RT: Cross-layer priority scheduling for predictable inter-process communication in ROS 2," in *Proc. 2025 IEEE 31st Real-Time and Embedded Technology and Applications Symp. (RTAS)*. IEEE, 2025, pp. 202–214.

[132] C.-S. Shih, H.-J. Lin, Y. Yuan, Y.-H. Kuo, and W.-Y. Liang, "Scalable and bounded-time decisions on edge device network using Eclipse Zenoh," in *Proc. 2022 IEEE 28th Int. Conf. on Embedded and Real-Time Computing Systems and Applications (RTCSA)*. IEEE, 2022, pp. 170–179.

[133] H. Abaza, D. Roy, B. Trach, W. Chang, S. Saidi, A. Motakis, W. Ren, and Y. Liu, "Managing end-to-end timing jitters in ROS 2 computation chains," in *Proc. 32nd Int. Conf. on Real-Time Networks and Systems*, 2024, pp. 229–241.

[134] C. Randolph, "Improving the predictability of event chains in ROS 2," *Ph.D. dissertation, Delft Univ. of Technology*, 2021.

[135] M. G. Valls, I. R. López, and L. F. Villar, "iLAND: An enhanced middleware for real-time reconfiguration of service oriented distributed real-time systems," *IEEE Trans. on Industrial Informatics*, vol. 9, no. 1, pp. 228–236, 2012.

[136] H. Pérez and J. J. Gutiérrez, "Modeling the QoS parameters of DDS for event-driven real-time applications," *J. of Systems and Software*, vol. 104, pp. 126–140, 2015.

[137] J.-B. Castillo-Sánchez, E. González-Parada, and J.-M. Cano-García, "Swarm robot communications in ROS 2: An experimental study," *IEEE Access*, vol. 12, pp. 142 930–142 943, 2024.

[138] S. Lee, J. Kang, and K.-J. Park, "Dependency chain analysis of ROS 2 DDS QoS policies: From lifecycle tutorial to static verification," *arXiv preprint arXiv:2509.03381*, 2025.

[139] P. Thulasiraman, Z. Chen, B. Allen, and B. Bingham, "Evaluation of the Robot Operating System 2 in lossy unmanned networks," in *Proc. 2020 IEEE Int. Systems Conf. (SysCon)*. IEEE, 2020, pp. 1–8.

[140] T. Kim *et al.*, "Mitigating DDS discovery bursts in ROS 2 multi-robot system," Ph.D. dissertation, DGIST, 2026.

[141] H.-S. Park, S. Lee, D. Um, H. Ryu, and K.-J. Park, "An analytical latency model of the Data Distribution Service in ROS 2," in *Proc. IEEE INFOCOM 2025 – IEEE Conf. on Computer Communications*. IEEE, 2025, pp. 1–10.

[142] C. Plasberg, H. Nessau, D. Timmermann, C. Eichmann, A. Roennau, and R. Dillmann, "Towards distributed real-time capable robotic control using ROS 2," in *Proc. 2022 IEEE 18th Int. Conf. on Automation Science and Engineering (CASE)*. IEEE, 2022, pp. 2205–2210.

[143] A.-l. Chisăliță and A. Korodi, "Stepping towards Zenoh protocol in automotive scenarios," *IEEE Access*, 2025.

[144] M. F. Ginting, K. Otsu, J. A. Edlund, J. Gao, and A.-A. Agha-Mohammadi, "CHORD: Distributed data-sharing via hybrid ROS 1 and 2 for multi-robot exploration of large-scale complex environments," *IEEE Robotics and Automation Letters*, vol. 6, no. 3, pp. 5064–5071, 2021.

[145] V. DiLuoffo, W. R. Michalson, and B. Sunar, "Robot Operating System 2: The need for a holistic security approach to robotic architectures," *Int. J. of Advanced Robotic Systems*, vol. 15, no. 3, pp. 1–11, 2018.

[146] B. Al-madani, A. Al-Roubaiey, and T. Al-shehari, "Wireless video streaming over Data Distribution Service middleware," in *Proc. 2012 IEEE Int. Conf. on Computer Science and Automation Engineering*. IEEE, 2012, pp. 263–266.

[147] P. Thulasiraman, Y. K. D. Cheng, and B. Allen, "Evaluation of the Data Distribution Service for a lossy autonomous hybrid system," in *Proc. 2022 IEEE Int. Systems Conf. (SysCon)*. IEEE, 2022, pp. 1–8.

[148] K. Hu, L. Zou, Z. Chen, J. Jiang, F. Jiang, and X. Tao, "Wireless multi-robot collaboration: Communications, perception, control, and planning," *IEEE Network*, vol. 39, no. 4, pp. 302–311, 2024.

[149] B. Almadani, M. Alsaeedi, and A. Al-Roubaiey, "QoS-aware scalable video streaming using Data Distribution Service," *Multimedia Tools and Applications*, vol. 75, no. 10, pp. 5841–5870, 2016.

[150] E. Dey, M. Walczak, M. S. Anwar, N. Roy, J. Freeman, T. Gregory, N. Suri, and C. Busart, "A novel ROS 2 QoS policy-enabled synchronizing middleware for co-simulation of heterogeneous multi-robot systems," in *Proc. 2023 32nd Int. Conf. on Computer Communications and Networks (ICCCN)*. IEEE, 2023, pp. 1–10.

[151] G. Szabó, "Toward the automatic network resource management of Robot Operating System in programmable mobile networks," *IEEE Access*, vol. 11, pp. 65 934–65 955, 2023.

[152] B. Li, Y. Zhu, Y. Zhao, K. Cui, X. Zhong, and K. Lu, "Fault tolerance DDS communications in Time-Sensitive Networking using IEEE 802.1 CB redundancy mechanisms," *J. of Systems Architecture*, p. 103756, 2026.

[153] D. Jo and Y. Kwon, "Generation of critical information and sharing mechanism for multi-robot mission success," *IEEE Access*, 2025.

[154] D. Calisi, F. Fedi, A. Leo, and D. Nardi, "Software development for networked robot systems," *IFAC Proc. Volumes*, vol. 43, no. 16, pp. 605–610, 2010.

[155] J. Wytrębowicz, K. Cabaj, and J. Krawiec, "Messaging protocols for IoT systems — a pragmatic comparison," *Sensors*, vol. 21, no. 20, p. 6904, 2021.

[156] S. Asjad, E. Harber, V. Santos, and D. Tyler, "NeuroReality: A Data Distribution Service-based inter-process communication middleware," in *Proc. 2024 IEEE Conf. on Telepresence*. IEEE, 2024, pp. 168–171.

[157] G. Toffetti, L. Militano, R. Tharaka, and M. Straub, "ROS-based robotic applications orchestration in the compute continuum: Challenges and approaches," in *Proc. IEEE/ACM 16th Int. Conf. on Utility and Cloud Computing*, 2023, pp. 1–6.

[158] H. Teper, M. Günzel, N. Ueter, G. von der Brüggen, and J.-J. Chen, "End-to-end timing analysis in ROS 2," in *Proc. 2022 IEEE Real-Time Systems Symp. (RTSS)*. IEEE, 2022, pp. 53–65.

[159] A. Jalil and J. Kobayashi, "Efficacy of local cache for performance improvement of reliable data transmission in aggregated robot processing architecture," in *Proc. 2022 22nd Int. Conf. on Control, Automation and Systems (ICCAS)*. IEEE, 2022, pp. 1339–1344.

[160] A. Jalil, J. Kobayashi, and T. Saitoh, "Performance improvement of multi-robot data transmission in aggregated robot processing architecture with caches and QoS balancing optimization," *Robotics*, vol. 12, no. 3, p. 87, 2023.

[161] R. Li, Z. Song, M. Lv, J.-M. Wu, C. J. Xue, J. Wang, and N. Guan, "ATER: Adaptive task execution rate regulation for enhanced real-time performance in ROS 2," in *Proc. 2025 IEEE 31st Int. Conf. on Embedded and Real-Time Computing Systems and Applications (RTCSA)*. IEEE, 2025, pp. 90–101.

[162] A. Romero-Garcés, J. F. Inglés-Romero, J. Martínez, and C. Vicente-Chicote, "Self-adaptive quality-of-service in distributed middleware for robotics," in *Proc. Workshop on Recognition and Action for Scene Understanding (REACTS)*, 2013.

[163] J. Sanchez-Monedero, J. Povedano-Molina, J. M. Lopez-Vega, and J. M. Lopez-Soler, "Bloom filter-based discovery protocol for DDS middleware," *J. of Parallel and Distributed Computing*, vol. 71, no. 10, pp. 1305–1317, 2011.

[164] H. Lee, D. Kim, and S. Moon, "Optimizing Data Distribution Service discovery for swarm unmanned aerial vehicles through preloading and network awareness," *Drones*, vol. 9, no. 8, pp. 1–20, 2025.

[165] W.-P. Nwadiugwu, D.-S. Kim, W. Ejaz, and A. Anpalagan, "MAD-DDS: Memory-efficient automatic discovery data distribution service for large-scale distributed control network," *IET Communications*, vol. 17, no. 12, pp. 1432–1446, 2023.

[166] S. Sang and Y. Ling, "Service discovery-based hybrid network middleware for efficient communication in distributed robotic systems,"

in *Proc. 2025 IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*. IEEE, 2025, pp. 1357–1364.

[167] L. P. Chovet, G. M. Garcia, A. Bera, A. Richard, K. Yoshida, and M. A. Olivares-Mendez, "Performance comparison of ROS 2 middlewares for multi-robot mesh networks in planetary exploration," *J. of Intelligent & Robotic Systems*, vol. 111, no. 1, p. 18, 2025.

[168] Ö. Köksal and B. Tekinerdogan, "Obstacles in Data Distribution Service middleware: A systematic review," *Future Generation Computer Systems*, vol. 68, pp. 191–210, 2017.

[169] F. Desbiens, *Building enterprise IoT solutions with Eclipse IoT technologies: An open source approach to edge computing*. Springer, 2023.

[170] D. Portugal, R. P. Rocha, and J. P. Castilho, "Inquiring the Robot Operating System community on the state of adoption of the ROS 2 robotics middleware," *Int. J. of Intelligent Robotics and Applications*, vol. 9, no. 2, pp. 454–479, 2025.

[171] Real-Time Innovations, "Data-centric middleware: Modular software infrastructure to monitor, control and collect real-time equipment analytics," Whitepaper. [Online]. Available: https://www.rti.com/hubfs/docs/RTI_Data_Centric_Middleware.pdf, 2014, accessed: 2026-04-02.

[172] K. Chen, N. Tian, C. Juette, T. Qiu, L. Ren, J. Kubiatowicz, and K. Goldberg, "FogROS2-PLR: Probabilistic latency-reliability for cloud robotics," in *Proc. 2025 IEEE Int. Conf. on Robotics and Automation (ICRA)*. IEEE, 2025, pp. 16 290–16 297.

[173] B. Jaiswal, H. Tyagi, A. Gopalan, and V. Sevani, "WiROS: A QoS software solution for ROS 2 in a WiFi network," in *Proc. 2023 15th Int. Conf. on COMmunication Systems & NETworkS (COMSNETS)*. IEEE, 2023, pp. 216–218.

[174] S. Parra, S. Schneider, and N. Hochgeschwender, "Specifying QoS requirements and capabilities for component-based robot software," in *Proc. 2021 IEEE/ACM 3rd Int. Workshop on Robotics Software Engineering (RoSE)*. IEEE, 2021, pp. 29–36.

[175] J. Hoffert, D. Schmidt, and A. Gokhale, "A QoS policy configuration modeling language for publish/subscribe middleware platforms," in *Proc. 2007 Inaugural Int. Conf. on Distributed Event-Based Systems*, 2007, pp. 140–145.

[176] K. An, T. Kuroda, A. Gokhale, S. Tambe, and A. Sorbini, "Model-driven generative framework for automated OMG DDS performance testing in the cloud," *ACM SIGPLAN Notices*, vol. 49, no. 3, pp. 179–182, 2013.

[177] A. Jalil, J. Kobayashi, and T. Saitoh, "QoS balancing optimization in aggregated robot processing architecture: Rate and buffer," *J. of Advances in Artificial Life Robotics*, vol. 3, no. 4, pp. 209–213, 2023.

[178] A. Fasano, "Optimizing inter-process communication for robotics applications," Ph.D. dissertation, Politecnico di Torino, 2025.

[179] C. S. V. Gutiérrez, L. U. S. Juan, I. Z. Ugarte, and V. M. Vilches, "Towards a distributed and real-time framework for robots: Evaluation of ROS 2.0 communications for real-time robotic applications," *arXiv preprint arXiv:1809.02595*, 2018.

[180] 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.

[181] D. P. Klüner, M. Molz, A. Kampmann, S. Kowalewski, and B. Alrifaee, "Modern middlewares for automated vehicles: A tutorial," *IEEE Intelligent Transportation Systems Magazine*, 2026.

[182] M. Richart, F. Velázquez, F. Ciuffardi, J. Visca, and J. Baliosian, "CoCoSim: A tool for co-simulation of mobile cooperative robots," in *Proc. Int. Conf. on Software Engineering and Formal Methods*. Springer, 2022, pp. 258–268.

[183] S. Acharya, A. Bharadwaj, Y. Simmhan, A. Gopalan, P. Parag, and H. Tyagi, "CorNet: A co-simulation middleware for robot networks," in *Proc. 2020 Int. Conf. on COMmunication Systems & NETworkS (COMSNETS)*. IEEE, 2020, pp. 245–251.

[184] S. Moon, J. J. Bird, S. Borenstein, and E. W. Frew, "A Gazebo/ROS-based communication-realistic simulator for networked SUAS," in *Proc. 2020 Int. Conf. on Unmanned Aircraft Systems (ICUAS)*. IEEE, 2020, pp. 1819–1827.

[185] J. Shi, A. Ma, X. Cheng, M. Sha, and P. Xi, "Adapting wireless network configuration from simulation to reality via deep learning-based domain adaptation," *IEEE/ACM Trans. on Networking*, vol. 32, no. 3, pp. 1983–1998, 2023.

[186] S. Agarwal, A. Asija, S. K. Kaul, and S. Anand, "Sim-to-real transfer for estimation over wireless networks," *ACM Trans. on Cyber-Physical Systems*, vol. 9, no. 4, pp. 1–25, 2025.

[187] M. Calvo-Fullana, D. Mox, A. Pyattaev, J. Fink, V. Kumar, and A. Ribeiro, "ROS-NetSim: A framework for the integration of robotic and network simulators," *IEEE Robotics and Automation Letters*, vol. 6, no. 2, pp. 1120–1127, 2021.

[188] J. Kim, J. M. Smereka, C. Cheung, S. Nepal, and M. Grobler, "Security and performance considerations in ROS 2: A balancing act," *arXiv preprint arXiv:1809.09566*, 2018.

[189] S. Sandoval and P. Thulasiraman, "Cyber security assessment of the Robot Operating System 2 for aerial networks," in *Proc. 2019 IEEE Int. Systems Conf. (SysCon)*. IEEE, 2019, pp. 1–8.

[190] M. Aartsen, K. Banga, K. Talko, D. Touw, B. Wisman, D. Meïnsma, and M. Björkqvist, "Analyzing interoperability and security overhead of ROS 2 DDS middleware," in *Proc. 2022 30th Mediterranean Conf. on Control and Automation (MED)*. IEEE, 2022, pp. 976–981.

[191] S. Takemoto, K. Nishida, Y. Nozaki, M. Yoshikawa, S. Honda, and R. Kurachi, "Performance evaluation of CAESAR authenticated encryption on SROS 2," in *Proc. 2019 2nd Artificial Intelligence and Cloud Computing Conf.*, 2019, pp. 168–172.

[192] L. Xia, X. Gao, and W. Shi, "Investigating security threats in multi-tenant ROS 2 systems," in *Proc. 2025 IEEE Int. Conf. on Robotics and Automation (ICRA)*. IEEE, 2025, pp. 16 441–16 448.

[193] B. Wang, H. Li, and J. Guan, "A formal analysis of Data Distribution Service security," in *Proc. 19th ACM Asia Conf. on Computer and Communications Security*, 2024, pp. 716–727.

[194] J. Du, C. Gao, and T. Feng, "Formal safety assessment and improvement of DDS protocol for industrial Data Distribution Service," *Future Internet*, vol. 15, no. 1, p. 24, 2022.

[195] G. Deng, G. Xu, Y. Zhou, T. Zhang, and Y. Liu, "On the (in)security of secure ROS 2," in *Proc. 2022 ACM SIGSAC Conf. on Computer and Communications Security*, 2022, pp. 739–753.

[196] Y. Patel, P. H. Rughani, and D. Desai, "Analyzing security vulnerability and forensic investigation of ROS 2: A case study," in *Proc. 8th Int. Conf. on Robotics and Artificial Intelligence*, 2022, pp. 6–12.

[197] L. David, R. Vasconcelos, L. Alves, R. André, and M. Endler, "A DDS-based middleware for scalable tracking, communication and collaboration of mobile nodes," *J. of Internet Services and Applications*, vol. 4, no. 1, p. 16, 2013.

[198] R. White, H. I. Christensen, G. Caiazza, and A. Cortesi, "Procedurally provisioned access control for robotic systems," in *Proc. 2018 IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*. IEEE, 2018, pp. 1–9.

[199] Q. Wang, B. Lee, and Y. Qiao, "Support remote attestation for decentralized Robot Operating System (ROS) using trusted execution environment," in *Proc. 2024 IEEE Int. Conf. on Blockchain and Cryptocurrency (ICBC)*. IEEE, 2024, pp. 693–695.

[200] P. G. Wagner, P. Birnstill, and J. Beyerer, "DDS Security+: Enhancing the Data Distribution Service with TPM-based remote attestation," in *Proc. 19th Int. Conf. on Availability, Reliability and Security*, 2024, pp. 1–11.

[201] G. Sciangula, D. Casini, A. Biondi, C. Scordino, and M. Di Natale, "End-to-end response-time analysis of DDS-based real-time applications," *Internet of Things*, p. 101853, 2025.

[202] J. Yu, S. Lee, Y. Choi, and K.-J. Park, "ros2probe: Non-intrusive, kernel-selective observability for Robot Operating System 2 middleware," *arXiv preprint arXiv:2606.10746*, 2026.

[203] A. Hakiri, P. Berthou, S. Slim Abdellatif, M. Diaz, and T. Gayraud, "Supporting end-to-end internet QoS for DDS-based large-scale distributed simulation," in *Proc. 1st ACM SIGSIM Conf. on Principles of Advanced Discrete Simulation*, 2013, pp. 397–402.

[204] B. Almadani, S. Abudalfa *et al.*, "QoS adaptation for publish/subscribe middleware in real-time dynamic environments," *Int. Arab J. of Information Technology (IAJIT)*, vol. 14, no. 2, 2017.

[205] R. Zhao, G. Yang, J. Fleming, and B. Chen, "Distributed state estimation for discrete-time LTI systems: The design trilemma and a novel framework," *arXiv preprint arXiv:2603.20144*, 2026.

[206] M. Bosk and J. Ott, "Towards domain-specific time-sensitive information-centric networking architecture," in *Proc. 2024 IFIP Networking Conf. (IFIP Networking)*. IEEE, 2024, pp. 720–725.

[207] J. Hoffert, D. Schmidt, and A. Gokhale, "DQML: A modeling language for configuring distributed publish/subscribe Quality of Service policies," in *Proc. OTM Confederated Int. Conf. on the Move to Meaningful Internet Systems*. Springer, 2008, pp. 515–534.

[208] J. Hoffert, A. Gokhale, and D. C. Schmidt, "Timely autonomic adaptation of publish/subscribe middleware in dynamic environments," *Int. J. of Adaptive, Resilient and Autonomic Systems (IJARAS)*, vol. 2, no. 4, pp. 1–24, 2011.

[209] P. Garcia Lopez, R. Gracia, M. Espelt, G. Paris, M. Arrufat, and R. Messeguer, "Topology-aware group communication middleware for MANETs," in *Proc. 4th Int. ICST Conf. on COMmunication System softWAre and middlewaRE*, 2009, pp. 1–10.

[210] G. Szabó, "Enhancing vertical application and network co-design: A solution for multipath channel switching in ROS 2 with 3GPP integration," *IEEE Access*, 2025.

[211] G. Orsini, D. Bade, and W. Lamersdorf, "CloudAware: A context-adaptive middleware for mobile edge and cloud computing applications," in *Proc. 2016 IEEE 1st Int. Workshops on Foundations and Applications of Self* Systems (FAS*W)*. IEEE, 2016, pp. 216–221.

[212] P. Bellavista, A. Corradi, and C. Giannelli, "Context-aware middleware for reliable multi-hop multi-path connectivity," in *Proc. IFIP Int. Workshop on Software Technologies for Embedded and Ubiquitous Systems*. Springer, 2008, pp. 66–78.

[213] J. Balasubramanian, S. Tambe, C. Lu, A. Gokhale, C. Gill, and D. C. Schmidt, "Adaptive failover for real-time middleware with passive replication," in *Proc. 2009 15th IEEE Real-Time and Embedded Technology and Applications Symp.* IEEE, 2009, pp. 118–127.

[214] J. Balasubramanian, A. Gokhale, D. C. Schmidt, and N. Wang, "Towards middleware for fault-tolerance in distributed real-time and embedded systems," in *Proc. IFIP Int. Conf. on Distributed Applications and Interoperable Systems*. Springer, 2008, pp. 72–85.

[215] E. Pitoura and B. Bhargava, "Data consistency in intermittently connected distributed systems," *IEEE Trans. on Knowledge and Data Engineering*, vol. 11, no. 6, pp. 896–915, 2002.

[216] S. Paul, D. Lephuoc, and M. Hauswirth, "Performance evaluation of ROS 2–DDS middleware implementations facilitating cooperative driving in autonomous vehicle," *arXiv preprint arXiv:2412.07485*, 2024.

[217] D. Yoshino, Y. Watanobe, and K. Naruse, "High communication performance Internet of Robotic Things systems based on RT-Middleware," *IEEE Access*, vol. 13, pp. 207 527–207 540, 2025.

[218] R. Kurte, Z. Salcic, and K. I.-K. Wang, "Decentralised global service discovery for the Internet of Things," *Sensors*, vol. 24, no. 7, p. 2196, 2024.

[219] S. Ebadi and L. M. Khanli, "A new distributed and hierarchical mechanism for service discovery in a grid environment," *Future Generation Computer Systems*, vol. 27, no. 6, pp. 836–842, 2011.

[220] P. Detzner, J. Gödeke, L. Tönning, P. Laskowski, M. Hörstrup, O. Stolz, M. Brehler, and S. Kerner, "SOLA: A decentralized communication middleware developed with ns-3," in *Proc. 2023 Workshop on ns-3*, 2023, pp. 78–85.

[221] J. Jung, S. Lee, N. Kim, and H. Yoon, "Efficient service discovery mechanism for wireless sensor networks," *Computer Communications*, vol. 31, no. 14, pp. 3292–3298, 2008.

[222] R. Harbird, S. Hailes, and C. Mascolo, "Adaptive resource discovery for ubiquitous computing," in *Proc. 2nd Workshop on Middleware for Pervasive and Ad-Hoc Computing*, 2004, pp. 155–160.

[223] R. Mokadem, A. Hameurlain, and A. M. Tjoa, "Resource discovery service while minimizing maintenance overhead in hierarchical DHT systems," in *Proc. 12th Int. Conf. on Information Integration and Web-based Applications & Services*, 2010, pp. 630–638.

[224] W. Tang, L. Wang, and G. Xue, "Design of high availability service discovery for microservices architecture," in *Proc. 2019 3rd Int. Conf. on Management Engineering, Software Engineering and Service Sciences*, 2019, pp. 253–257.

[225] P. Gomes, E. Cavalcante, T. Rodrigues, T. Batista, F. C. Delicato, and P. F. Pires, "A federated discovery service for the Internet of Things," in *Proc. 2nd Workshop on Middleware for Context-Aware Applications in the IoT*, 2015, pp. 25–30.

[226] S. Kim, H. Choi, S. Lee, M. Kim, H. Shin, and C. Moon, "A dynamic bridge architecture for efficient interoperability between AUTOSAR adaptive and ROS 2," *Electronics*, vol. 14, no. 18, p. 3635, 2025.

[227] Open Robotics, "ROS 2 domain bridge design and implementation," [Online]. Available: https://docs.ros.org/en/kilted/p/domain_bridge/doc/design.html, 2024, accessed: Apr. 6, 2026.

[228] B.-Y. Song and H. Choi, "ROS gateway: Enhancing ROS availability across multiple network environments," *Sensors*, vol. 24, no. 19, p. 6297, 2024.

[229] W. Sim, B. Song, J. Shin, and T. Kim, "Data Distribution Service converter based on the OPC UA publish–subscribe protocol," *Electronics*, vol. 10, no. 20, p. 2524, 2021.

[230] M. Boari, E. Lodolo, S. Monti, and S. Pasini, "Middleware for automatic dynamic reconfiguration of context-driven services," *Microprocessors and Microsystems*, vol. 32, no. 3, pp. 145–158, 2008.

[231] M. Aragão, P. Moreno, and A. Bernardino, "Middleware interoperability for robotics: A ROS–YARP framework," *Frontiers in Robotics and AI*, vol. 3, p. 64, 2016.

[232] Eclipse Zenoh, "Zenoh bridge for DDS," [Online]. Available: https://docs.ros.org/en/kilted/p/zenoh_bridge_dds/, 2024, accessed: Apr. 6, 2026.

[233] B. Song, X. Hu, J. Xiao, G. Zhang, S. Wang, and Q. Zhou, "Integration of Data Distribution Service into partitioned real-time embedded systems," in *Proc. 2020 15th IEEE Conf. on Industrial Electronics and Applications (ICIEA)*. IEEE, 2020, pp. 1606–1611.

[234] Zenoh longwang release. Zenoh. [Online]. Available: https://zenoh.io/blog/2026-04-16-zenoh-longwang/