MobileROS: A Wireless-Native Robot Operating System for Mobile Robotics

Boyi Liu, Qianyi Zhang, Yongguang Lu, Jianhao Jiao, Jagmohan Chauhan, Wen Wu, Jun Zhang, and Dimitrios Kanoulas

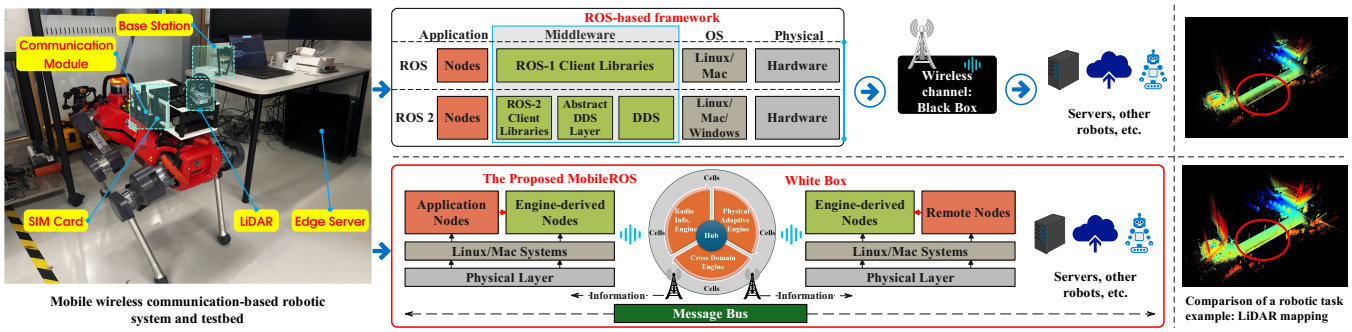


Fig. 1: The proposed wireless-native robot operating system vs. the legacy ROS-based framework [12]. The top section illustrates the foundational architecture of the ROS-based system, while the bottom section presents our proposed MobileROS framework. MobileROS transforms wireless communication from a traditional external "black-box" module into a transparent core resource within the robot operating system. The right side of the figure compares LiDAR-based mapping results, demonstrating the performance improvements enabled by MobileROS. MobileROS provides an open-source platform for the robotics community to study latency-sensitive tasks in wireless mobile communication environments.

Abstract*—The increasing deployment of mobile robots in dynamic outdoor environments necessitates robotic systems capable of maintaining reliability amidst fluctuating wireless connectivity. While the Robot Operating System (ROS) has established itself as the de facto standard for such networked robotics, its abstraction of communication as an opaque, best-effort utility creates a critical bottleneck: it fails to leverage physical layer (PHY) information, resulting in degraded performance and unreliable execution in fluctuating networks. To address this, this paper presents MobileROS, a wireless-native robot operating system that transforms wireless communication from an external service into a core system resource. Grounded in the Symbiotic Paradigm, MobileROS establishes a bidirectional exchange where network conditions inform robotic decisions and mission requirements guide network resource allocation. Based on service mesh principles and domain-driven design, our architecture implements a Hub-Engines-Cells (HEC) model. It features a central Hub for global optimization, three specialized engines (the Radio Information Engine, the Cross Domain Engine, and the Physical Adaptive Engine) for cross-layer intelligence, and distributed Cells as functional units. A key mechanism, Application-Driven Bidirectional Dynamic Slicing, allows robots to actively reconfigure network resources based on semantic urgency, transforming the robot from a passive observer into an active network controller. We systematically evaluate MobileROS across three cities (London, Hong Kong, and Shenzhen) in five scenarios: distributed visual SLAM, cross-domain LiDAR perception, V2X autonomous driving, hybrid multi-robot collaboration against WebRTC baselines, and partition recovery validating CAP-theorem-aware failsafe mechanisms. Results demonstrate that MobileROS maintains significantly more stable performance than standard ROS in mobile wireless deployments. We provide implementation details at https://github.com/MobileROS.

This work was partly supported by the Hong Kong Research Grants Council under the Areas of Excellence Scheme grant AoE/E-601/22-R, the NSFC/RGC Collaborative Research Scheme grant CRS_HKUST603/22, and the UKRI Future Leaders Fellowship [MR/V025333/1] (RoboHike).

Boyi Liu and Jun Zhang are with the Hong Kong University of Science and Technology, Hong Kong SAR, China (e-mail: bliubd@connect.ust.hk; eejzhang@ust.hk).

Qianyi Zhang, Jianhao Jiao, Jagmohan Chauhan, and Dimitrios Kanoulas are with the Department of Computer Science, University College London, London WC1E 6BT, U.K. (e-mail: qianyi.zhang@ucl.ac.uk; jianhao.jiao@ucl.ac.uk; jagmohan.chauhan@ucl.ac.uk; d.kanoulas@ucl.ac.uk).

Yongguang Lu and Wen Wu are with Peng Cheng Laboratory, Shenzhen 518055, China (e-mail: luyg5@mail2.sysu.edu.cn; wuw02@pcl.ac.cn).

Corresponding authors: Dr. Jianhao Jiao and Prof. Jun Zhang.

I. Introduction

The rapid advancement of outdoor mobile robotics, ranging from autonomous delivery [30] and precision agriculture [27] to large-scale inspection and environmental monitoring [11], is driving a stringent demand for high-performance wireless connectivity [15, 9, 31]. Meanwhile, with the rise of 5G and future 6G communication technologies, research has increasingly focused on achieving ultra-reliable, low-latency connections to support complex robotic missions [4]. However, bridging the gap between fluctuating wireless realities and rigid robotic control loops remains a fundamental architectural challenge.

ROS, and especially ROS 2 with its adoption of the Data Distribution Service (DDS), has significantly enhanced robotic communications by offering rich Quality of Service (QoS) policies such as Reliability, History, and Deadline. These mechanisms allow developers to tune middleware behavior for varied environments. However, while DDS provides powerful configurability at the middleware level, it fundamentally treats the underlying network as a black box, creating a critical visibility gap regarding the Physical Layer (PHY). In the standard architecture, rapid fluctuations of the

wireless channel (e.g., signal-to-noise ratio drops, modulation coding scheme changes, or resource block congestion in 5G) are abstracted away from the application layer. We describe this limitation not as a lack of configurability, but as an opacity to physical channel states. Without access to these PHY metrics, a robot cannot proactively switch its algorithmic strategy (e.g., from high-precision LiDAR mapping to sparse feature transmission) until after packet loss occurs at the middleware layer. This reactive paradigm inevitably leads to delays, dropped messages, and degraded decision-making capability [18], potentially leading to mission failures as communication conditions deteriorate.

Although advanced wireless systems research has produced several open source communication frameworks that provide flexible platforms for mobile communication, these are still primarily designed for general telecommunication services [6] and do not inherently consider the dynamic and task-specific requirements of robotic missions. Such frameworks fail to expose radio conditions in a manner suitable for real-time robot control and decision-making. Moreover, their telecom-centric focus imposes a steep learning curve for robotics researchers. The standard ROS ecosystem, while powerful for sensor fusion and modular software integration, does not natively incorporate real-time knowledge of the radio environment at the PHY level. This gap leaves robotics researchers without a straightforward means to leverage the full potential of wireless networks to improve robot autonomy.

To address this essential gap, we introduce MobileROS not as a replacement, but as a wireless-native extension to the ROS 2 ecosystem. MobileROS is grounded in what we term the **Symbiotic Paradigm**, a fundamental shift from the traditional decoupled "middleware-as-a-pipe" model toward a tightly integrated architecture where communication and robotic domains mutually inform each other. We align with the emerging "Communication-Control Co-Design" consensus [32]. We argue that while code modules must remain decoupled for maintainability, information flow must be symbiotic to handle physical uncertainty.

We explicitly acknowledge that MobileROS depends on the ROS middleware for distributing global policies. However, to prevent system deadlock during network failures, MobileROS employs a local "Circuit Breaker" mechanism. Local engines monitor the physical link state via hardware drivers independent of the ROS message queue, and can preemptively trigger local safety behaviors (Failsafe Mode) even if the middleware transport layer becomes unresponsive.

As depicted in Fig. 1, MobileROS introduces a "white box" approach to wireless integration, exposing key PHY-related insights that can be dynamically aligned with robotic tasks. Based on service mesh principles [25] and domain-driven design theory [10], our system integrates communication and robotics domains to enable dynamic adaptation to changing wireless conditions while optimizing task performance. Through this integration, MobileROS facilitates a synergistic relationship between communication capabilities and robotic operations, addressing the fundamental limitations of current approaches.

The key contributions of this paper are summarized as follows.

1. We present MobileROS, **the first open-source robot operating system to natively integrate cellular communication as a core information resource**. Departing from conventional "black-box" paradigms, our system transforms the wireless communication module into a transparent, PHY-layer-aware component. This approach is instantiated through the Symbiotic Paradigm, establishing a bidirectional information coupling that enables joint optimization unavailable to isolated domains.
2. **The Hybrid Coordination Architecture (HCA)** is proposed as a new centralized coordination and distributed execution framework based on the Hub-Engines-Cells model. HCA leverages service mesh principles for robust inter-module communication and domain-driven design for managing complexity across communication and robotic domains. Unlike traditional approaches that implement network adaptations at either the application layer (too high) or require hardware-specific solutions (too low), HCA provides a modular, system-level approach that bridges these domains through well-defined translation interfaces.
3. We design **Application-Driven Bidirectional Dynamic Slicing** as a core mechanism that enables robots to actively negotiate network resources based on semantic task urgency. This transforms MobileROS from a passive "5G Client" into an active "5G Controller", a capability absent in standard ROS.
4. **From a wireless systems perspective, MobileROS provides the first open-source implementation of Bidirectional Dynamic Slicing for robotic applications.** Unlike existing 5G stacks (e.g., OAI, srsRAN) that support only static slice configuration or unidirectional network-to-UE policies, our implementation enables real-time, application-initiated slice negotiation, allowing robots to actively request resource upgrades based on task-level semantics. This capability serves as the underlying enabler for the cross-layer adaptation mechanisms throughout the system.
5. **We conducted systematic evaluation through five targeted experiments.** These include: (1) Distributed SLAM demonstrating PHY-layer awareness benefits, (2) LiDAR perception validating cross-domain adaptation, (3) V2X autonomous driving with active slice control, (4) Hybrid multi-robot collaborative localization comparing against WebRTC baselines, and (5) Network partition recovery validating CAP-theorem-aware failsafe mechanisms. Conducted over 1.5 years across London, Hong Kong, and Shenzhen, these experiments validate MobileROS under diverse regulatory and spectrum environments.

Our evaluation methodology focuses on task-oriented

benchmarks rather than only traditional network metrics, emphasizing how network characteristics directly impact robotic task effectiveness. This approach acknowledges that in robotic systems, networking performance is meaningful primarily in the context of its effect on task-level execution quality.

The remainder of this paper is organized as follows. Section II reviews related works, highlighting gaps in current approaches. Section III describes the HCA design and its core components in detail. Section IV presents how MobileROS establishes a new communication-aware operational paradigm with its data flow. Section V shows the implementation details. Section VI demonstrates the effectiveness of our system through experiments on various robotic tasks. Section VII presents an analysis of the network performance of MobileROS. Finally, Section VIII concludes the work.

II. Related Works

A. Robot Operating Systems and CAP Trade-offs

The Robot Operating System (ROS) has evolved to address the complexities of distributed robotic computing. ROS 1 [28] employed a centralized architecture where a Master node managed service discovery and parameter synchronisation. In the context of the **CAP theorem** (Consistency, Availability, Partition Tolerance), this design prioritizes Consistency (CP): the system maintains a unified global state, but lacks Partition Tolerance — if the Master fails (Single Point of Failure), the entire network halts. Conversely, ROS 2 [23, 1] adopted a fully decentralized architecture based on the Data Distribution Service (DDS) [24]. By utilizing multicast discovery, ROS 2 shifts towards an AP architecture (Availability and Partition Tolerance), ensuring that distributed nodes can continue exchanging data even if parts of the network are severed [3]. A mechanism-focused comparison of these architectures is presented in Table I and II.

However, these architectures present a critical *Abstraction Gap*. Standard ROS/ROS 2 middleware operates strictly at the Transport Layer to preserve modularity, deliberately abstracting away the underlying network state. As noted by Kim et al. [17] and Kronauer et al. [19], this abstraction renders the system blind to Physical Layer dynamics. When the wireless channel fluctuates (e.g., SNR drops), the middleware cannot distinguish a temporary congestion event from a permanent failure, leading to reactive timeouts rather than proactive adaptation. Furthermore, the purely distributed nature of ROS/ROS 2 makes it difficult to implement global optimization strategies, as no single entity possesses a holistic view of the network resource state.

To address this architectural dichotomy, MobileROS introduces a **Hybrid Coordination Strategy**. Unlike standard ROS which is either purely centralized or purely distributed, MobileROS decouples *Execution* from *Information Flow*. We maintain **Distributed Execution** (via Engine-derived Cells running locally on robots) to ensure high Availability and real-time responsiveness. Simultaneously, we establish a **Centralized Information Flow** (via the Hub) to achieve the Consistency required for global resource optimization. This hybrid approach allows the system to leverage PHY-layer visibility for global planning while retaining local survivability during network partitions. This design does not conflict with the CAP theorem; rather, it strategically navigates it by separating the execution plane (AP-oriented) from the decision plane (CP-oriented), ensuring that local safety is never compromised by global partition. **Moreover, we further developed a corresponding mechanism to address the single-point-of-failure challenge motivated by the CAP theorem, as in Section III-G**.

B. Transport Protocols

Parallel to robotic frameworks, the networking community has developed robust protocols to address the **Connectivity Problem** in dynamic environments.

- **Mesh Networking:** Protocols such as *batman-adv* and OLSR establish multi-hop connectivity, extending operational range in non-line-of-sight scenarios [21].
- **WebRTC:** Originally designed for browser-based communication, WebRTC has become a standard for low-latency P2P streaming with built-in NAT traversal [22].
- **QUIC:** QUIC utilizes UDP to eliminate Head-of-Line (HoL) blocking, reducing connection latency for real-time applications [20].

While these technologies effectively solve the challenge of *"how to reach the peer,"* they suffer from a fundamental **Semantic Gap** in robotics. Standard congestion control algorithms (e.g., Google Congestion Control) are **Content-Agnostic**: they optimize for aggregate metrics like throughput or fairness, treating all data packets identically [8]. However, robotic data is **inherently semantic** and hierarchical. For instance, in real-time SLAM, losing a keyframe is catastrophic for map consistency. A transport-layer protocol may randomly discard high-value semantic packets during congestion to maintain bitrate.

To address this semantic void, MobileROS implements **Source-Side Elastic Adaptation** at the Application Layer. As compared in Table I, unlike transport protocols that passively react to congestion by dropping packets, MobileROS actively monitors PHY-layer capacity to modulate the data generation process itself. For example, it employs "Elastic Stream" mechanisms to dynamically adjust SLAM keyframe insertion rates or "Adaptive Voxel" filtering to reduce point cloud density before serialization. This ensures that the application logic proactively aligns with network reality, preserving semantic integrity (e.g., tracking continuity) even under constrained bandwidth.

C. Systems for Mobile Networked Robotics

Research in this domain spans two distinct but adjacent fields: *Robotic Network Systems* and *Cellular Infrastructure Tools*. A comprehensive comparison of these approaches against MobileROS is provided in Table I.

- **Robotic Network Systems:** Frameworks like **WiROS** [16]

TABLE I: Architectural Mechanism Comparison of Systems for Mobile Networked Robotics

System/Protocol	PHY-Layer Access	Adaptation Direction	Failure Semantics	ROS Integration	Execution Model
Robotic Middleware Systems					
MobileROS (Ours)	*Direct via 5G Stack*	*Bidirectional (Active Slicing)*	*CAP-Aware Degradation*	*Native Extension*	*Centralized + Distributed*
ROS 2 + DDS [23]	None (Abstracted)	None	Timeout-Based Retry	Native	Fully Distributed
FogROS [5]	None	Unidirectional (Offload)	Cloud Failover	ROS Bridge	Cloud-Centric
WiROS [16]	WiFi RSSI Only	Unidirectional (QoS Adjust)	Best-Effort	ROS 2 Plugin	Local Only
CHORD [13]	None	Unidirectional (Rate Limit)	Store-and-Forward	ROS 1/2 Hybrid	Mesh Distributed
Transport Protocols					
WebRTC Bridge [22]	None	Unidirectional (GCC)	Frame Drop	Custom Bridge	P2P Stream
QUIC Protocol [20]	None	Unidirectional (Flow Control)	Stream Reset	Library Binding	Client-Server/P2P
Mesh (batman-adv) [21]	Layer 2 Only	Unidirectional (Routing)	Route Rediscovery	OS Network Interface	Distributed Mesh
Cellular Infrastructure Tools					
O-RAN RIC [29]	Full RAN Access	Bidirectional (Internal)	Operator-Defined	None	Network-Centric
OpenAirInterface (OAI) [26]	Full RAN Access	Static Policy / Unidirectional	Radio Link Failure	None	Base Station Centric
srsRAN [14]	Full RAN Access	Static Policy	Radio Link Failure	None	Base Station Centric

Legend: PHY=Physical Layer; GCC=Google Congestion Control; CAP=Consistency-Availability-Partition Tolerance; RIC=RAN Intelligent Controller

TABLE II: Architectural Comparison: ROS 1 vs. ROS 2 vs. MobileROS.

Mechanism	ROS 1	ROS 2 (DDS)	MobileROS
Topology	Centralized Master (Single Point of Failure)	Distributed (Discovery via multicast)	**Hybrid** (Distributed Exec. + Logical Coord.)
Wireless Awareness	None (Black box)	**Low** (Abstracted by QoS policies)	**High** (Direct access to PHY/MAC metrics)
Adaptation	None	**Reactive** (Retransmission on loss)	**Proactive** (Predictive rate control)
Failure Handling	Node Disconnection	QoS Deadline Policies	**Degraded Mode** (Local safety fallback)

and **CHORD** [13] focus on adapting robot behavior to network conditions, typically optimizing QoS for WiFi or multi-robot exploration. Cloud-offloading frameworks like **FogROS** [5] focus on moving computation to remote servers.

- **Cellular Infrastructure:** On the telecommunications side, open-source stacks like **OpenAirInterface (OAI)** [26] and **srsRAN** [14] provide complete software-defined implementations of 3GPP standards (4G/5G). These platforms offer a fully programmable RAN and Core network, enabling advanced research into network slicing and scheduling algorithms that is impossible with commercial black-box hardware.

A critical disconnect exists between these two domains. Robotic systems often adhere to a "Strict Decoupling" philosophy, resulting in **Information Isolation** where the robot cannot access raw PHY-layer states from the cellular stack. Conversely, cellular tools like OAI are designed for operator-centric management with static policies. Crucially, they lack "Robot-facing" APIs and primarily support **unidirectional** or pre-configured slicing. They do not inherently support the **bidirectional dynamic slicing** required for a robot to programmatically trigger a slice upgrade in real-time based on semantic urgency.

To address this integration gap, MobileROS establishes a **Symbiotic Paradigm** that bridges robotics and cellular infrastructure. We argue that while *Operational Decoupling* (modular software interfaces) is essential for maintainability, *Information Coupling* is vital for performance. MobileROS serves as the translation layer: it exposes the PHY-layer visibility of tools like OAI to the robot (sensing) and translates the robot's semantic intent into network commands (control). This enables a bidirectional loop where the robot acts as an active network controller, a capability absent in isolated middleware or infrastructure stacks. This approach does not violate the "Golden Rule" of system decoupling; rather, it maintains the operational decoupling of execution nodes while enabling information coupling for state awareness, thereby enhancing robustness through cross-layer intelligence.

III. Architecture

Based on our analysis of existing systems' limitations, we recognize that a fundamental redesign is needed to bridge the gap between wireless communication and robotic operations. Previous approaches have either treated wireless as a black box or failed to establish meaningful bidirectional information flow between domains. To address these challenges, we introduce the HCA that underpins MobileROS, which transforms wireless from an external service to an integral system component while preserving ROS compatibility. In our discussion, we prioritize functional roles ("System Coordinator," "Local Adaptor") to emphasize utility, while the Hub-Engines-Cells (HEC) terminology serves as the formal, implementable blueprint. We begin with an overall architectural overview, then we delve into its hierarchical layering, and finally we detail the distributed execution mechanisms at the node level.

A. System Overview: the White Box Approach

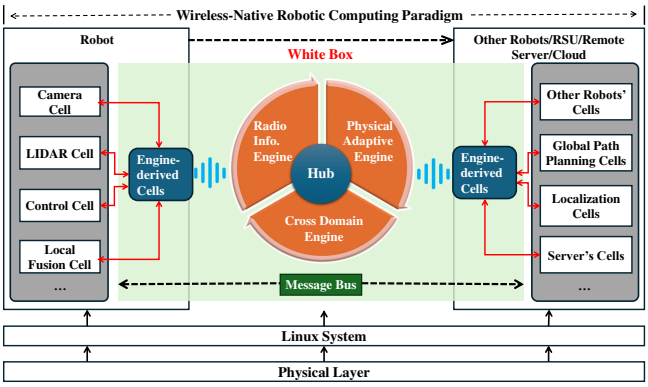


Fig. 2: MobileROS Architecture. (a) **Logical View**: The Hub-Engines-Cells hierarchy governing information flow. (b) **Physical Deployment View**: A distributed implementation example. The **Hub** runs on the Edge Server, coordinating global optimization. On the **Robot** side, local *Engine-derived Cells* interface with hardware to execute adaptation strategies (e.g., rate control) and report PHY metrics, communicating with the Hub via the wireless link.

Fig. 2 presents the conceptual overview of the HCA. Unlike conventional approaches that treat wireless as an external "black box", HCA implements a "white box" paradigm that exposes wireless information directly to robotic components. This transformation is achieved through a three-part structure we call the Hub-Engines-Cells (HEC) model. This model acts as the formal, implementable blueprint that maps functional roles (like the Coordinator) to technical execution units (like the Hub nodelet).

At the core of the architecture is the **Hub**, which functions as the central coordination mechanism. Positioned in the middle of Fig. 2, the Hub maintains global system awareness and coordinates responses across domains. Similar to the ROS master (`roscore`) in its coordination role, the Hub is a centralized logical entity responsible for global discovery and policy orchestration. Crucially, although the Hub is logically centralized for consistency, strictly decoupled local execution ensures partition tolerance (as detailed in Sec. III-G). It typically resides on the infrastructure side (e.g., Edge Server) to maintain a global view of the network state.

Surrounding the Hub are three specialized **Engines** that serve as bidirectional translators between technical domains: the **Radio Information Engine** processes physical layer wireless metrics, the **Physical Adaptive Engine** modifies transmission strategies based on both network and robot requirements, and the **Cross Domain Engine** performs spatio-temporal fusion of wireless and robotic information. The outer layer consists of **Cells** that implement concrete functionality within well-defined domains, both on the robotic side (Camera Cell, LiDAR Cell, etc.) and the communications side (PHY Cell, Medium Access Control (MAC) Cell).

Engine-derived Cells (Distributed Executors): Acting as local execution proxies, these lightweight agents reside on each robot. They locally execute the adaptation policies (e.g., downsampling sensor data) and upload compressed PHY metrics to the Hub, ensuring real-time responsiveness without overloading the wireless link. This separation between centralized policy-making and distributed execution is key to MobileROS's scalability.

The architecture draws from two key theoretical concepts in distributed systems engineering. **Service Mesh principles** enable a flexible communication framework through strategically positioned proxy components, creating an abstraction layer that facilitates decoupled service discovery, transparent traffic management, and distributed telemetry collection. **Domain-Driven Design (DDD)** methodology guides our architectural boundaries, establishing explicit bounded contexts that cleanly separate wireless and robotics domains while creating well-defined interfaces between them. This conceptual structure establishes a coherent information flow from physical wireless metrics to high-level robotic decision-making, creating what we term the "Wireless-Native Robotic Computing Paradigm".

B. Hierarchical Structure: Layers and Components

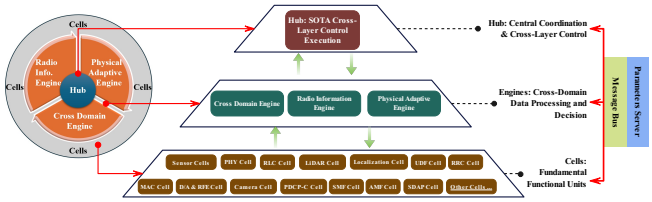


Fig. 3: The Hub-Engines-Cells hierarchy: a layered model implementing centralized coordination with distributed execution. The Hub manages global coordination policies, Engines provide specialized cross-domain intelligence, and Cells implement domain-specific functionalities within the ROS computational graph. The hierarchical structure facilitates information flow across layers while preserving domain integrity.

Fig. 3 illustrates the hierarchical organization of the HCA into three distinct layers. This layered structure balances the benefits of centralized coordination with the responsiveness of distributed execution.

Hub (Centralized Coordination): Positioned at the top of the hierarchy in Fig. 3, the Hub implements the central coordination functions of the system. As a specialized ROS node with privileged middleware access, it actively collects real-time metrics from all domains through aggregation channels. The Hub performs three critical functions: (1) it analyzes cross-cutting concerns spanning both wireless and robotic subsystems, identifying patterns and conditions invisible to individual components; (2) it dynamically formulates and updates system-wide policies based on comprehensive analysis of the global state; and (3) it distributes coordinated directives to appropriate Engines through prioritized message queues via the Message Bus shown at the bottom of Fig. 3.

The Hub's Global State Registry: The Hub is not a passive relay but an active state machine. It maintains a "Global State Registry" that synchronizes: (1) **Network Context:** Aggregated PHY metrics (SINR, PRB usage) from

the Radio Information Engine; and (2) **Robot Context:** Task criticality scores (e.g., "Safety-Critical" vs. "Logging") from the Application Layer. The Hub executes a 10Hz control loop: `Monitor` → `Analyze` → `Dispatch`. When a state transition occurs (e.g., `STABLE` → `CONGESTED`), it broadcasts a concise `PolicyUpdate` message to all distributed Cells, triggering immediate local adaptation.

Separation of Supervisory and Real-Time Control: It is important to clarify that the Hub's 10Hz policy loop is a *supervisory coordination cycle* that updates adaptation strategies (e.g., compression ratios, transmission modes), not a real-time control loop. Safety-critical control loops (e.g., motor commands, obstacle avoidance at 100–1000 Hz) execute entirely on the robot's local computing hardware using standard ROS 2 intra-process communication, and are never routed through the wireless link or the Hub. MobileROS only intercepts messages destined for wireless transmission via the Cell wrapper mechanism (Section III-C). Consequently, local ROS 2 advantages—including shared-memory zero-copy transport, intra-process communication, and DDS QoS policies—are preserved for all on-robot communication.

Engines (Cross-Domain Intelligence): The middle layer of Fig. 3 shows the three specialized Engines that bridge between technical domains. Each Engine implements advanced processing logic that transforms domain-specific information into actionable intelligence:

- *The Radio Information Engine* processes raw PHY and MAC layer metrics through sliding-window statistical analysis, converting technical parameters like Signal-to-Interference-plus-Noise Ratio (SINR) and block error rates into ROS messages with semantic meaning. This Engine creates an abstraction layer that shields robotic applications from the complexity of wireless technologies while exposing essential insights for network-aware operation.
- *The Physical Adaptive Engine* implements closed-loop controllers that modify transmission strategies based on both network feedback and robot task requirements. This Engine connects directly to both the wireless infrastructure and the robotic components, dynamically adjusting parameters such as message priorities, compression ratios, and channel allocation as conditions evolve, ensuring that communication resources are allocated optimally to support mission priorities.
- *The Cross Domain Engine* performs critical spatio-temporal fusion, combining wireless metrics with robotics state information to create enriched context. This fusion process enables integrated planning algorithms to optimize robot behavior considering both task objectives and connectivity constraints. For example, during path planning, this Engine allows the robot to consider both physical obstacles and regions of poor connectivity as constraints to be jointly optimized.

The Engines exemplify Domain-Driven Design principles by functioning as cross-domain middleware that preserve bounded contexts, transforming domain-specific data into actionable intelligence comprehensible across domains, and implementing bidirectional adaptation mechanisms.

Cells (Distributed Functional Units): At the lowest level of the hierarchy, shown at the bottom of Fig. 3, Cells implement concrete functionality within well-defined domains. These nodelet-compatible components execute specific tasks with clear modularity and domain boundaries:

- *On the robotics side*, Cells include functional units such as Camera Cell, LiDAR Cell, Control Cell, and Local Fusion Cell. These components handle conventional robotic tasks like sensor data acquisition and processing, but with network awareness, dynamically adjusting their operation based on communication conditions.
- *On the communications side*, specialized Cells expose traditionally opaque protocols, including PHY Cell, Radio Link Control (RLC) Cell, MAC Cell, and other communication stack components. Through these Cells, MobileROS achieves unprecedented visibility into the wireless communication stack.

The Cells implement DDD principles by executing domain-specific tasks with appropriate abstractions, exposing traditionally opaque protocols through standardized interfaces, and remaining responsive to cross-domain coordination while maintaining domain integrity.

C. Internal Mechanics: The Cell Wrapper Mechanism

To demystify the "black box" perception, we clarify that MobileROS operates via **Application-Level Interception** using the Decorator Pattern, not kernel modifications. When a user's ROS node calls `publish()`, the workflow proceeds as follows:

1) **Hook:** The SensorCell wrapper intercepts the message pointer before serialization.
2) **Policy Check:** The wrapper queries the local Engine-derived Policy Cache for the current adaptation strategy.
3) **Payload Mutation:** Based on the policy, the wrapper triggers a transformation function (e.g., `downsample_pointcloud()`, `compress_image()`).
4) **Transmission:** The modified payload is passed to the standard ROS publisher for serialization and transport.

This design preserves full compatibility with existing ROS nodes; they require no source code modification. Adaptation logic is injected transparently via the Cell wrapper layer, which users enable through launch file configuration.

D. Bidirectional Dynamic Slicing: From Passive Sensing to Active Control

A fundamental innovation in MobileROS is the transition from **passive network sensing** to **active network control**. Standard 5G middleware treats User Equipment (UE) as passive clients that accept whatever QoS the network assigns. MobileROS fundamentally inverts this relationship through **Application-Driven Bidirectional Dynamic Slicing**.

The Mechanism: Instead of relying solely on static network policies, we implement a lightweight middleware layer

based on a **UE-gNB Slice Manager coordination protocol**. This establishes a dedicated control channel that allows MobileROS to inject semantic urgency directly into the RAN scheduling logic:

- **Legacy Capability (Passive, Downlink):** The robot adapts its behavior to network constraints - e.g., reducing sensor resolution when bandwidth is limited.
- **New Capability (Active, bidirectional dynamic slicing):** When the Cross-Domain Engine detects a safety-critical context (e.g., collision risk prediction $> 80\%$), it triggers the **UE Slice Manager** to issue a high-priority `UE_STATE_REPORT`. This report is transmitted to the **gNB Slice Manager** embedded in the base station. The gNB immediately invokes the **Slice Policy Executor**, dynamically overriding the default scheduling weights and reallocating Physical Resource Blocks (PRBs) to promote the robot's data flow to a prioritized slice for the duration of the maneuver. This slice execution is bidirectional and dynamic.

Technical Breakthrough: Current state-of-the-art open-source systems, such as OAI and srsRAN, lack the capability for bidirectional dynamic slicing. We are the first to implement and open-source this functionality.

The Innovation: This constitutes an **Application-Driven Network Reconfiguration**. Unlike standard 5G slicing which assumes static service characteristics, this architecture exposes a "hook" for the robotic application layer to interact with the **Slice Policy Executor** in real-time. MobileROS bridges this gap, enabling the robot to "negotiate" for resources based on semantic urgency. This is the key "contribution on top of 5G" that transforms the robot from a passive network client into an active network controller. Experimental validation is provided in Section VI-C.

E. Distributed Execution: Feedback Loops and Adaptation

Fig. 4 details how the HCA architecture is implemented across distributed physical nodes while maintaining logical cohesion. This figure builds directly upon the hierarchical structure shown in Fig. 3, illustrating how components interact in practical deployment scenarios.

The left side of Fig. 4 shows the robot's local components, where sensor nodes and local computing resources interact with Engine-derived Cells. These Engine-derived Cells, shown in the center of both sides of the figure, are distributed execution units derived from the Engines (Radio Information Engine, Physical Adaptive Engine, and Cross Domain Engine). They implement policies defined by their parent Engines, which reside in the middle layer of the architecture as depicted in Fig. 3. For example, on the left side, Cells such as the Camera Cell and LiDAR Cell interact with Engine-derived Cells to process sensor data, extending the coordination capabilities of the centralized Engines to distributed environments. The Hub, as the central coordinator, orchestrates these interactions from a logical core position, though it is not explicitly shown in Fig. 4 as it operates across all nodes.

Physical network components are shown in the lower portion of the figure, with distinct nodes for different aspects of the wireless stack O-RU (Open Radio Unit) Node, O-DU (Open Distributed Unit) Node, O-CU-CP (Open Centralized Unit - Control Plane Node, etc.). These components represent communication-side Cells, such as the PHY Cell and MAC Cell, as introduced in Fig. 3. They provide direct access to traditionally opaque network parameters and are managed by the Engines, particularly the Radio Information Engine, which processes the physical layer information and transmits it to other engines.

The right side of Fig. 4 illustrates remote nodes such as cloud servers, Roadside Units (RSU), or other robots. These entities include additional Cells (e.g., cloud-based processing Cells) that participate in the distributed architecture through standardized interfaces. The Engine-derived Cells on this side, also shown in the center, execute strategies from the Engines, while the Hub maintains global optimization by coordinating across all these distributed components, ensuring logical cohesion across physically separate hardware.

This distributed execution model implements several crucial feedback loops:

- **Network-to-Robot Adaptation**: The Radio Information Engine's derived Cells continuously monitor network conditions and propagate quality metrics to robotic components, enabling proactive adaptation to changing wireless environments.
- **Robot-to-Network Optimization**: The Physical Adaptive Engine's derived Cells modify network parameters based on robotic task requirements, prioritizing communication resources according to mission needs.
- **Cross-Domain Coordination**: The Cross Domain Engine's derived Cells maintain spatial and temporal correlation between network conditions and robot position, establishing location-aware networking policies.

The connection between Fig. 3 and 4 is particularly evident in how the logical organization (Hub, Engines, Cells) is preserved across physical deployment boundaries. Engine-derived Cells, appearing in the middle sections of Fig. 4, are the practical implementation of the Engines' functionality in a distributed environment, extending the coordination capabilities to any location within the system.

Through this Hub-Engines-Cells architecture and its distributed execution model, MobileROS achieves both global optimization and local responsiveness. The system enables direct correlation between network metrics and robotic performance indicators, providing a framework for task-oriented evaluations that quantify how network characteristics translate to functional outcomes.

F. Design Patterns for Robotics Practitioners

To facilitate practical adoption, we formalize the adaptation logic into reusable **Design Patterns**. These patterns provide explicit mappings between network metrics and algorithm parameters, enabling practitioners to systematically integrate network awareness into their applications. The core patterns

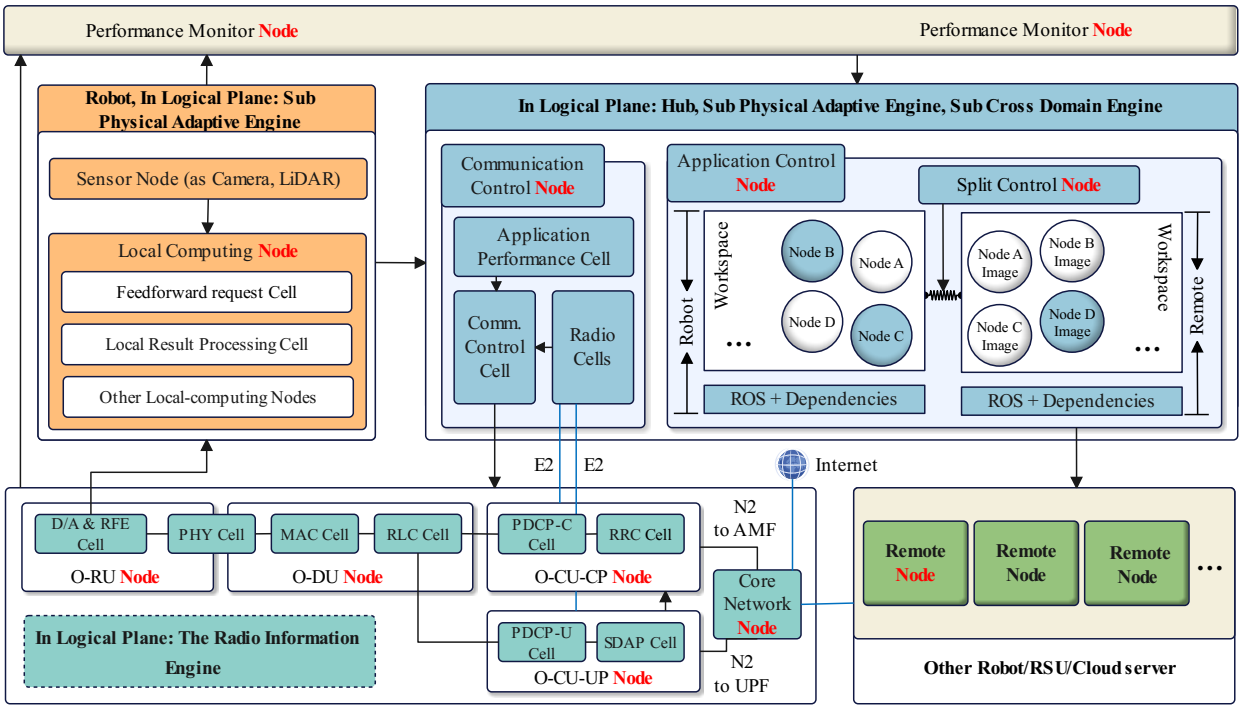


Fig. 4: Distributed execution and feedback loops in HCA: logical nodes and physical entities collaborate via Service Mesh-enabled messaging and DDD-based domain abstractions. The system spans across multiple physical nodes (left side: Robot computing components, center: Core networking components, right side: Remote nodes) while maintaining logical organization through Engine-derived Cells. This distributed architecture enables dynamic adaptation of transmission parameters, sensor modalities, and planning strategies based on real-time communication metrics and mission requirements.

include: (1) *Elastic Stream* for perception loops, (2) *Adaptive Voxel* for point cloud processing, (3) *Gated Execution* for planning tasks, and (4) *Priority Escalation* for safety-critical operations. Detailed formulations and implementation guidance are provided in Section IV-G.

G. CAP-Theorem-Aware Failsafe Design

A common concern with centralized architectures is the single point of failure. MobileROS addresses this through explicit CAP-theorem-aware design:

Normal Operation (CP Mode): When the Hub is reachable, MobileROS operates as a Consistency-Partition tolerant system. The Hub provides authoritative policy decisions, and Engine-derived Cells synchronize their state with the Hub. This ensures globally optimal resource allocation.

Partition Recovery (AP Mode): When the Hub becomes unreachable (detected via heartbeat timeout, default of 500 ms), Engine-derived Cells automatically transition to **Degraded Mode**:

- **Local Policy Cache:** Each Cell maintains a cached copy of the last-known-good policies from the Hub, enabling continued operation using stale but still valid parameters.
- **Conservative Fallback:** If the partition exceeds a configurable threshold (default 5s), Cells switch to predefined conservative policies (e.g., minimum transmission rate, maximum compression) to ensure safety-critical functionality.
- **Graceful Reconnection:** Upon Hub recovery, Cells resynchronize their state and resume normal operation without requiring restarts.

This design explicitly trades **Consistency** for **Availability** during partitions, ensuring that robots remain operational (albeit suboptimally) rather than failing completely. Experimental validation is provided in Section VI-E.

IV. Data Flow

This section describes the data flow within MobileROS's HCA, shown in Fig. 5. The data flow design implements a bidirectional communication pathway between wireless network conditions and robotic decision processes, enabling adaptive behavior in variable wireless environments while maintaining ROS compatibility.

A. Core Components and Message Propagation

The data flow is orchestrated by the Hub's central coordination logic (referred to as the Convergence Cell), which manages information exchange between network-aware and robotic components. This specialized ROS node coordinates the ingestion of network metrics, translation into robotic semantics, and distribution of adaptation policies. Rather than implementing a simple pipeline, MobileROS establishes a continuous feedback loop where network conditions inform robotic decisions, and robot requirements guide network resource allocation. The Convergence Cell manages message routing within the ROS computational graph, dynamically adjusting topic subscriptions and message priorities based on current mission requirements and network status. It maintains a runtime representation of the entire system augmented

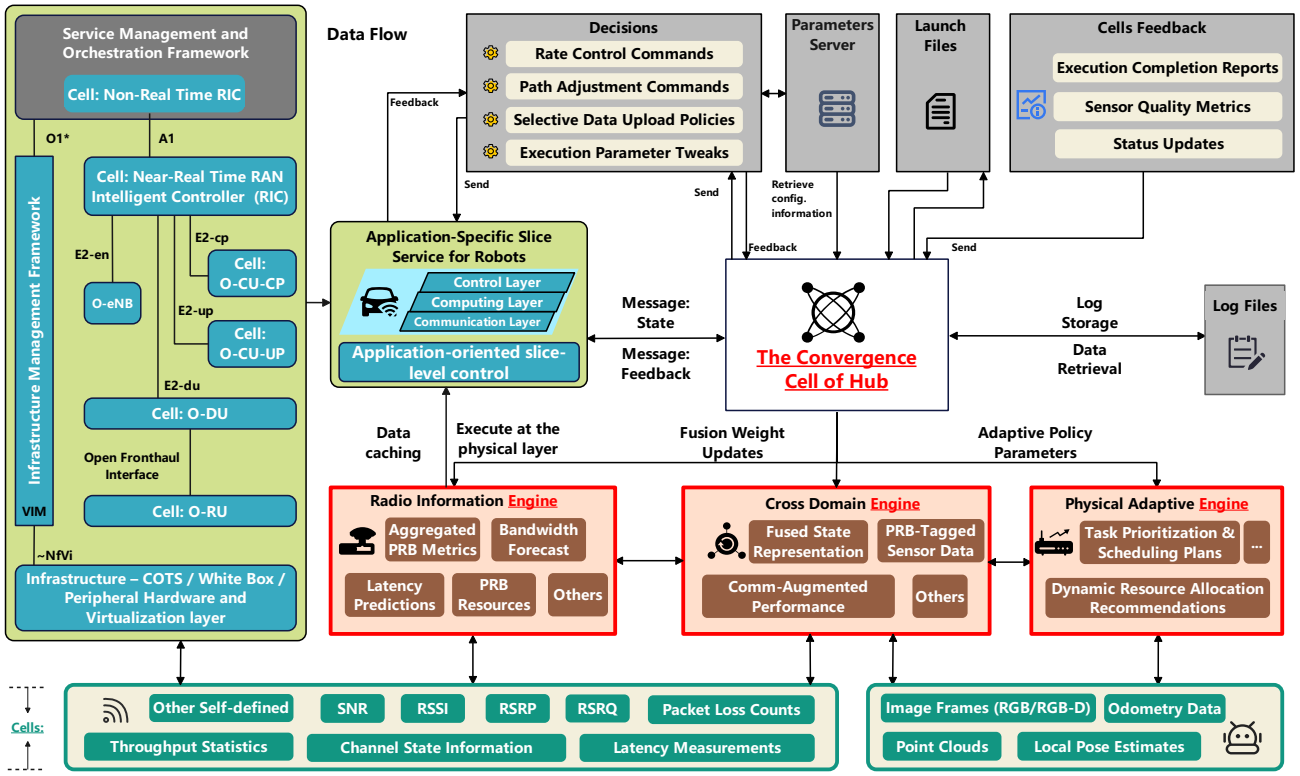


Fig. 5: Data flow in the Hybrid Coordination Architecture (Simplified). The diagram emphasizes cross-domain information flow rather than component boundaries. Network metrics flow upward through the Radio Information Engine, cross-domain translation occurs in the Convergence Cell, and adaptation policies flow downward to robotic Cells.

with network quality metadata, enabling global optimization decisions that balance communication resources against task demands.

MobileROS extends the standard ROS parameter server with capabilities for network awareness, maintaining multiple parameter sets optimized for different network conditions. When wireless quality changes, the Radio Information Engine publishes these changes to a dedicated topic, triggering the Convergence Cell to update parameters and reconfigure affected components. This dynamic parameter management enables rapid adaptation without requiring node restarts. Feedback from execution Cells flows back to the Convergence Cell through enhanced diagnostic messages and specialized topics, completing the adaptation loop. This feedback includes task completion status, sensor quality metrics, and resource utilization statistics, all tagged with network context to enable correlation between performance and wireless conditions.

B. Cross-Domain Translation Process

The Cross Domain Engine implements the critical function of translating between network and robotic semantics. This translation process transforms raw wireless metrics into actionable information for robotic algorithms through three sequential operations. First, the engine subscribes to network quality topics published by the Radio Information Engine, receiving continuous updates on signal strength, bandwidth availability, and connectivity reliability. These metrics are published as specialized ROS messages containing both instantaneous measurements and trend analyses. Next, it fuses this network information with robotic application data. For example, in visual SLAM systems, network jitter and latency metrics are translated into feature tracking parameters and keyframe insertion rates, while in LiDAR perception, bandwidth availability is mapped to point cloud density and structural preservation requirements. Finally, the engine publishes transformed data to cross-domain topics using standardized message types that robotic algorithms can directly incorporate. A SLAM system, for instance, receives network-aware parameters that automatically adjust feature detection thresholds based on current wireless conditions, maintaining localization accuracy without requiring the perception algorithms to be aware of network protocols.

C. Adaptive Execution Cycle

MobileROS implements a continuous adaptation cycle that responds to changing network conditions through four coordinated phases:

The **measurement phase** collects raw network metrics and robot state information through specialized monitoring nodes. Network monitoring Cells interface directly with wireless hardware to extract physical layer metrics, while system monitors track application performance and resource utilization. During the **analysis phase**, the Radio Information Engine processes raw measurements into meaningful assessments of current and predicted network quality. Simultaneously,

the Cross Domain Engine correlates these assessments with robotic state and task requirements to identify potential performance bottlenecks or optimization opportunities. During the **adaptation phase**, the Physical Adaptive Engine determines appropriate adjustments based on the analyzed information. These may include reconfiguring sensor data rates, adjusting compression levels, rescheduling bandwidth-intensive operations, or modifying navigation parameters to prioritize areas with better connectivity. Finally, during the **execution phase**, adaptation decisions propagate to operational Cells through parameter updates and control messages. Cells implement these adaptations through mechanisms such as dynamic resolution scaling, variable compression rates, and priority-based message filtering, maintaining critical functionality even as network conditions fluctuate.

Handling Complete Signal Loss (Failsafe Mechanism): To address extreme environments where network connectivity may be entirely severed, MobileROS implements a dedicated **Degraded Mode**. The *Radio Information Engine* continuously monitors the physical link status. If the signal quality drops below a critical floor (e.g., RSRP < -140 dBm) or a connection timeout occurs (> 500 ms), the Hub broadcasts a `SIGNAL_LOST` event.

Upon receiving this event, the system immediately switches to Failsafe policies:

1) **Computation Fallback:** Distributed Tasks (e.g., Cloud SLAM) are suspended, and the robot reverts to local dead-reckoning (e.g., Wheel Odometry + IMU) to maintain basic state estimation.
2) **Safety Actuation:** The Control Cell triggers a safety maneuver, such as "Stop and Hover" for Unmanned Aerial Vehicles (UAVs) or "Return to Last Known Good Signal" for Unmanned Ground Vehicles (UGVs), preventing the robot from blindly entering hazardous areas.

This ensures that even during a total communication blackout, the robot remains safe and recoverable, rather than stalling or crashing due to missing remote dependencies.

This closed-loop adaptation cycle creates measurable relationships between network conditions and task performance, allowing us to systematically evaluate system performance through task-specific metrics rather than isolated network statistics.

D. Application-Driven Network Slicing: From Passive Client to Active Controller

To address the limitation where robots are merely passive consumers of network quality, MobileROS introduces a **Bidirectional Control Loop**. Unlike standard middleware that relies on static QoS profiles, MobileROS implements a dynamic bridge to the 5G Control Plane.

1. The Mechanism: We integrated a custom interface to the OpenAirInterface (OAI) SD-RAN controller via the E2 interface. This allows the MobileROS Hub to inject policy commands directly into the gNB Scheduler.

- **Trigger:** When the *Cross-Domain Engine* identifies a "Safety-Critical" task state (e.g., collision risk prediction $> 80\%$), it issues a `SLICE_PROMOTE` request.
- **Execution:** The Hub translates this semantic request into a specific Radio Resource Management (RRM) command, overriding the default "Best Effort" slice and programmatically promoting the robot's data flow to a prioritized slice for the duration of the maneuver.

2. The Innovation: This constitutes an **Application-Driven Network Reconfiguration**. Standard 5G infrastructure allows for slicing, but it does not expose these controls to the robotic application layer. MobileROS bridges this gap, enabling the robot to "negotiate" for resources based on semantic urgency.

E. Network-Aware Data Transport

MobileROS enhances ROS communication infrastructure through three methodologies validated in our experiments. First, the system implements content-aware data encoding that selectively processes information based on semantic importance. This includes adaptive downsampling techniques for point clouds and intelligent keyframe selection for visual data, preserving critical structural information while reducing transmission volume. Second, it employs adaptive parameter tuning that automatically adjusts application parameters based on observed network conditions. For instance, SLAM feature density and keyframe insertion rates are modified according to available bandwidth and latency characteristics. Third, it implements cross-domain resource optimization to balance computational and network loads. For instance, the system performs additional data preprocessing locally when bandwidth is constrained, dynamically shifting the burden from the network to the onboard CPU. These mechanisms are implemented as extensions to standard ROS communication frameworks and operate transparently to application code. Our experiments in Section VI demonstrate how these approaches maintain application performance across varying network conditions, preserving structural integrity in perception tasks and ensuring operational continuity despite connectivity fluctuations.

F. Domain-Structured Message Organization

MobileROS organizes its data flow according to domain-driven design principles, establishing clear logical boundaries while maintaining semantic integrity across contexts. **First**, network-domain messages encapsulate wireless metrics in their native form, while robotic-domain messages adhere to standard ROS conventions. **Second**, cross-domain translation interfaces implement formal context mapping rules that preserve consistent interpretation of critical concepts between domains. **Third**, this structured approach prevents semantic drift when network conditions affect safety-critical operations, while allowing integration without extensive modifications to existing ROS components. **Finally**, through this architecture, MobileROS transforms wireless communication from an external service into an integral part of the robotic cognition loop, embedding network awareness directly into perception, planning, and control processes for reliable oper-

ation in challenging wireless environments.

G. Design Patterns for Network-Aware Robotics

To enable robotics practitioners to effectively leverage MobileROS, we formalize the adaptation strategies as reusable **Design Patterns**. Each pattern defines the mapping between network metrics and algorithm parameters, providing clear guidance on "what to change" and "how to change it."

Pattern 1: Elastic Stream (for Visual SLAM)

The Elastic Stream pattern maintains tracking continuity by trading spatial density for temporal stability. The Hub dynamically maps measured Uplink Throughput (T_{ul}) to the keyframe insertion rate (K_{rate}):

$$K_{rate} = \min(K_{max}, \max(K_{min}, \alpha \cdot T_{ul}))$$

where α is a tunable gain, and K_{min}/K_{max} define safe operating bounds. This ensures that when bandwidth drops, the system sacrifices map density (spatial) to preserve tracking continuity (temporal), preventing "tracking lost" failures.

Pattern 2: Adaptive Voxel (for LiDAR Perception)

The Adaptive Voxel pattern dynamically adjusts point cloud resolution based on available bandwidth. The Voxel Grid filter leaf size (V_{leaf}) is modulated as:

$$V_{leaf} = V_{base} \cdot (1 + \beta \cdot (1 - B_{avail}/B_{max}))$$

where B_{avail} is available bandwidth, B_{max} is maximum bandwidth, and β controls sensitivity. Lower bandwidth increases voxel size, reducing point count while preserving structural features through octree-based filtering.

Pattern 3: Gated Execution (for Planning Tasks)

The Gated Execution pattern implements QoS-aware task scheduling for operations requiring stable connectivity. The planner's `make_plan` service is only invoked when the *Predicted Stability Window* (W_{stable}) exceeds a threshold:

$$\text{Execute} = \begin{cases} \text{True} & \text{if } W_{stable} > \tau_{min} \\ \text{False} & \text{otherwise} \end{cases}$$

where τ_{min} is the minimum required stability duration (default 2.0s). This prevents the robot from initiating negotiations or planning cycles that are statistically likely to time out due to impending network degradation.

Pattern 4: Priority Escalation (for Safety-Critical Tasks)

The Priority Escalation pattern triggers network slice upgrades based on semantic context:

1) Cross-Domain Engine evaluates task criticality score C_{task}
2) If $C_{task} > \theta_{critical}$, issue `SLICE_PROMOTE` to Hub
3) Hub translates to RRM command, upgrading the slice
4) After maneuver completion ($C_{task} < \theta_{normal}$), downgrade slice to release resources

These patterns are implemented in MobileROS's `adaptation_policies/` module and can be customized or extended by users for domain-specific applications.

V. Implementation Details

5G Network Stack & Active Slicing: MobileROS interfaces with the 5G network via a customized **OpenAirInterface (OAI) v2.1.0** stack deployed on **USRP B210/X410** software-defined radios. We operate the gNB in a monolithic configuration using the 3GPP Band n78 (3.5 GHz) TDD profile with a channel bandwidth of 40 MHz. Crucially, to enable the Active Control mechanism described in Section III-D, we utilize the **E2 interface** primarily for real-time network state monitoring (telemetry collection). For the actual execution of slice reconfiguration, we integrated a custom **Slice Policy Executor** directly within the OAI MAC scheduler. This design allows the gNB to instantly update Resource Block (RB) allocation weights upon receiving semantic triggers, bypassing the latency of standard management interfaces.

PHY Metric Access Mechanism: The Radio Information Engine obtains PHY-layer metrics through **direct access** to the OAI stack's internal data structures. Specifically, metrics such as SINR, RSRP, RSRQ, PRB utilization, and MCS index are read via the E2 telemetry interface and custom shared-memory hooks, providing ground-truth physical layer measurements rather than application-layer inferences. To facilitate portability, the Radio Information Engine defines a standardized `IMetricProvider` driver interface. Porting to alternative backends (e.g., srsRAN or commercial modems via AT/QMI commands) requires implementing this interface for the vendor-specific API. The current implementation has been validated exclusively with the OAI stack.

Implementation of Bidirectional Dynamic Slicing: To realize the active control loop, we implemented a dual-sided coordination architecture:

1) **UE-Side Middleware (The Hook):** On the robot, a lightweight **UE Slice Manager** operates as user-space middleware. Utilizing transparent UDP tunneling, it intercepts robotic application traffic without firmware modifications. It monitors application criticality scores from the Cross-Domain Engine and generates **UE State Reports**, encapsulating semantic intents (e.g., "Urgent Obstacle Avoidance") into control messages.
2) **gNB-Side Orchestration:** We extended the OAI MAC layer with a custom **gNB Slice Manager**. Upon receiving a UE State Report, the internal Policy Executor maps the semantic request to physical layer parameters.
3) **Closed-Loop Reconfiguration:** The **PRB Manager** instantly adjusts the scheduler's weight metrics. This establishes a bidirectional feedback loop: the robot signals needs via State Reports, and the network provides immediate resource adjustments via **UE Updates**. The measured reconfiguration latency from State Report to PRB reallocation is approximately 120 ms in our testbed.

ROS Integration & Modularity: MobileROS comprises 15 ROS packages designed as **drop-in replacements** for standard communication nodes. We utilize `pluginlib` to inject network-aware behaviors without breaking the standard ROS API. The Hub operates as a **Nodelet Manager**. (Note:

A *Nodelet* is a ROS mechanism that enforces zero-copy data transfer between algorithms in the same process, significantly reducing serialization overhead.) This design minimizes local CPU overhead when coordinating Engines and Cells. To ensure modularity, Cells act as wrappers (Decorator Pattern) around standard ROS drivers, maintaining full backward compatibility. Developers can enhance existing applications simply by remapping topics in the launch file without modifying source code. This pattern extends to any ROS 2 node: developers wrap existing publishers with Cell adapters and register for network callbacks, requiring no modifications to core algorithm logic.

Scope of Adaptation: The Cell wrapper mechanism currently targets **topic-based (pub/sub)** data streams, which represent the dominant high-bandwidth data flows in perception and mapping pipelines. Standard ROS 2 services (request/reply) and actions (goal/feedback/result) operate through the unmodified DDS transport layer and remain fully functional. For service reliability in fluctuating networks, the Gated Execution pattern (Section IV-G) can defer service invocations until network conditions are stable. Extending the Cell wrapper to transparently adapt service and action payloads is an avenue for future work.

A. Integration and Usage Example

MobileROS offers a high-level Python/C++ API designed for rapid integration with existing ROS 2 nodes. Listing 1 illustrates a minimal example showing how to transform a standard camera driver into a network-aware component.

As shown above, the original application logic remains largely intact. Developers primarily interact with the `Hub` to register interest in network states, while the complex signal processing is handled internally by the Engines.

B. Adaptive Compression and Trade-off Analysis

While lossless compression is the default for stable networks, MobileROS activates **configurable lossy compression** when the Radio Information Engine detects bandwidth scarcity. These mechanisms implement the **Adaptive Voxel** and **Elastic Stream** patterns formalized in Section IV-G:

1) **Visual Data:** Dynamically adjusting JPEG quality factors (from 95 to 30) or resolution (scaling down by 50%) based on current throughput limits.
2) **LiDAR Data:** Implementing an adaptive voxel grid filter that increases leaf size (downsampling) as uplink PRBs become congested.

Trade-off Justification: We explicitly prioritize *temporal continuity* over *spatial precision*. While lossless compression is generally preferred in data transmission, mobile robotics presents a unique constraint: "late data is wrong data." Our baseline experiments confirm that enforcing lossless transmission during channel congestion causes bufferbloat, leading to latency spikes > 500 ms. This delay desynchronizes the control loop, causing mission-critical failures (e.g., SLAM tracking loss). Therefore, MobileROS treats controlled lossy compression not as a flaw, but as a necessary safety mechanism. By dynamically downsampling (e.g., discarding 40% of LiDAR points) during dips in signal quality, we ensure the control loop frequency remains stable.

Listing 1: MobileROS Integration Example

```
from mobile_ros import Hub, RadioInfoEngine,
    CrossDomainEngine, SensorCell
from mobile_ros.core import ChannelObserver
from rclpy.node import Node
from sensor_msgs.msg import Image

class NetworkAwareCamera(Node, ChannelObserver):
   def __init__(self):
      super().__init__('network_aware_camera')
      # 1. Initialize Hub and Engines
      self.hub = Hub()
      self.rie = self.hub.attach_engine(
          RadioInfoEngine(), mode='realtime')
      self.cde = self.hub.attach_engine(
          CrossDomainEngine(), mode='optimized')

      # 2. Deploy a Sensor Cell wrapper
      self.cam_cell = self.hub.deploy_cell(
          SensorCell, "front_cam")

      # 3. Register for network state updates
      self.rie.register_observer(self)
      self.sub = self.create_subscription(Image, "/
          cam/raw", self.cb, 10)
      self.pub = self.create_publisher(Image, "/cam
          /processed", 10)

   def on_channel_update(self, state):
      # 4. Receive PHY metrics (SNR, PRB)
          automatically
      strategy = self.cde.compute_constraints(state
          .snr, state.prb_util)
      self.cam_cell.update_policy(strategy)

   def cb(self, msg):
      # 5. Apply adaptive constraints (e.g.,
          compression)
      if self.cam_cell.should_transmit():
         processed_msg = self.cam_cell.apply_policy
             (msg)
         self.pub.publish(processed_msg)
```

C. ROS 2 Discovery over Cellular Networks

A practical deployment consideration is that ROS 2's default DDS discovery mechanism relies on UDP multicast (SPDP), which is not natively supported by the OAI GTP-U tunnel or typical cellular PDN bearers. We address this through two complementary strategies: (1) deploying a lightweight **WebSocket bridge node** that relays ROS 2 topic data over unicast TCP/WebSocket connections, bypassing the multicast requirement for cross-network discovery; and (2) adopting **Eclipse Zenoh** with the `zenoh-bridge-ros2dds` plugin [7], which replaces DDS multicast discovery with Zenoh's TCP-based scouting protocol while transparently bridging all ROS 2 topics, services, and actions. The Zenoh approach is particularly attractive as it reduces discovery traffic by orders of magnitude and

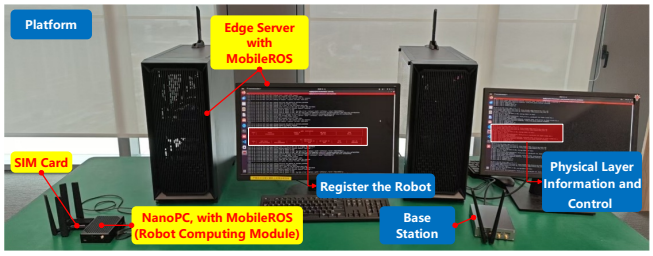


Fig. 6: MobileROS-enabled distributed visual SLAM system hardware setup. The experiment tests the "Elastic Stream" design pattern.

natively supports WAN-scale routing [2]. Both solutions are available in [7] and [2].

VI. Experiments

To rigorously validate the scientific contributions of the Symbiotic Paradigm, we designed five targeted experiments, each isolating a specific theoretical challenge:

1) **Exp A (Distributed SLAM):** Verifies the necessity of Information Coupling (vs. Strict Decoupling) in Cyber-Physical Systems.
2) **Exp B (LiDAR Perception):** Evaluates the efficacy of Source-Side Adaptation (vs. Network QoS) for high-bandwidth semantic data.
3) **Exp C (V2X Coordination):** Validates the value of Application-Driven Active Network Control (Bidirectional Slicing).
4) **Exp D (Hybrid Multi-Robot):** Demonstrates universality in heterogeneous Mesh networks against Transport-layer baselines (WebRTC/DDS).
5) **Exp E (Partition Recovery):** Tests resilience to CAP Theorem constraints and Single Point of Failure (SPOF) via the Dynamic CAP Strategy.

This structure moves beyond simple performance benchmarks to verify the architectural resilience of the system. We specifically target "Elastic" algorithms (SLAM, Point Cloud Streaming) that can tolerate variable data resolution but are sensitive to latency.

A. Experiment A: Information Coupling in Distributed SLAM

Hypothesis: Strict architectural decoupling leads to system blindness and failure under constrained resources; Information Symbiosis is required for survival.

Experimental Setup: As shown in Fig. 6, we implement a distributed visual SLAM system. We adopt an "Elastic Stream" design pattern, where the critical objective is to maintain real-time tracking frequency even under congestion. To achieve this, the Hub dynamically maps the measured Uplink Throughput (T_{ul}) to the ORB-SLAM3 `MaxKeyframesPerSecond` parameter (K_{rate}). The mapping follows a linear throttle logic:

$$K_{rate} = \min(K_{max}, \max(K_{min}, \alpha \cdot T_{ul}))$$

where α is a tunable gain (default: 2.0 keyframes/Mbps), and $K_{min} = 5$, $K_{max} = 20$ define safe operating bounds. This ensures that when bandwidth drops, the system sacrifices map density (spatial) to preserve tracking continuity (temporal).

TABLE III: Comparison of MobileROS ORB-SLAM3 performance under different slice-level PRB allocations. Results are reported as mean ± std over 3 runs. Arrows (↑/↓) indicate better direction. Best results are in **bold**.

Metric	Low PRB (30%) *(Mean±Std.)*	Medium PRB (60%) *(Mean±Std.)*	High PRB (90%) *(Mean±Std.)*
Network and Resource Configuration			
Localization Slice PRBs	10	20	30
Link Rate (Mbps) ↑	4.67 ± 0.15	7.20 ± 0.10	$\mathbf{9.90 \pm 0.10}$
UL Latency (ms) ↓	44.7 ± 1.5	28.3 ± 1.5	$\mathbf{15.3 \pm 0.6}$
Packet Loss Rate (%) ↓	3.20 ± 0.20	1.93 ± 0.15	$\mathbf{0.50 \pm 0.10}$
Jitter (ms) ↓	14.0 ± 1.0	8.0 ± 1.0	$\mathbf{4.3 \pm 0.6}$
ORB-SLAM3 Perception and Localization Metrics			
Mean Abs. Pose Error (m) ↓	0.35 ± 0.03	0.22 ± 0.02	$\mathbf{0.14 \pm 0.01}$
Std. Dev. Pos. Error (m) ↓	0.10 ± 0.01	0.063 ± 0.006	$\mathbf{0.033 \pm 0.006}$
Keyframe Rate (Hz) ↑	0.78 ± 0.03	1.10 ± 0.02	$\mathbf{1.32 \pm 0.03}$
Track Longevity (frames) ↑	19.0 ± 1.0	24.0 ± 1.0	$\mathbf{31.0 \pm 1.0}$

Results and Analysis: Table III presents the performance of MobileROS across varying PRB allocations. Leveraging MobileROS's network awareness capabilities, the system continuously measures network conditions and adaptively adjusts SLAM parameters.

To visually demonstrate this impact, Fig. 7 illustrates how varying slice-level Physical Resource Block (PRB) allocations directly affect map quality and trajectory accuracy. The figure confirms that higher resource allocation (High PRB) leads to denser point clouds and significantly reduced drift compared to constrained scenarios (Low PRB), validating the effectiveness of MobileROS's resource management.

TABLE IV: Baseline Comparison: Standard ROS vs. MobileROS under Constrained Resources (30% PRB).

Metric	Standard ROS *(Strictly Decoupled)*	MobileROS *(Info Symbiosis)*
Behavior	Blind Transmission	Adaptive Throttling
UL Latency	465 ± 130 ms	$\mathbf{45 \pm 2}$ **ms**
Packet Loss	19.8%	**3.2%**
Tracking Status	Frequent Loss	**Stable**
Mean Pose Error	$>$ 2.0 m (Drift)	**0.35 m**

Discussion: To validate the necessity of adaptation, we compared MobileROS against a Standard ROS baseline under identical low-bandwidth conditions (30% PRB). As detailed in Table IV, the standard system failed to adjust its keyframe insertion rate, saturating the uplink. This resulted in severe

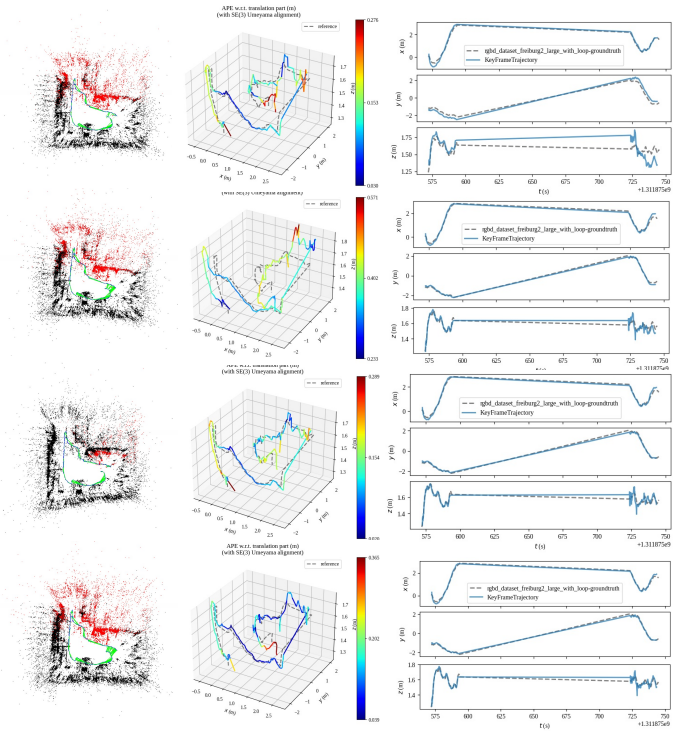


Fig. 7: Representative ORB-SLAM3 outcomes under varying slice-level PRB allocations. **Left:** Reconstructed point cloud maps and estimated trajectories for low vs. high PRB runs, showing the direct correlation between network resources and map quality. **Center:** 3D trajectory comparisons highlighting reduced drift as PRB resources increase, demonstrating improved localization accuracy. **Right:** Time series plots of X, Y, and Z pose dimensions against ground truth, showing closer alignment under higher PRB allocations, which indicates more reliable pose estimation.

bufferbloat, causing a **10× increase in latency** (465 ms vs. 45 ms). Consequently, the SLAM backend triggered "Tracking Lost" resets multiple times. In contrast, MobileROS throttled the keyframe rate to ≈ 0.78 Hz, successfully maintaining continuous tracking with a bounded error of 0.35 m. This comparison conclusively demonstrates that "Decoupling" in the presence of physical constraints is not a virtue but a vulnerability.

B. Experiment B: Source-Side Adaptation in LiDAR Perception

Hypothesis: Source-Side Adaptation (Application Layer) significantly outperforms Middleware-only QoS in handling high-bandwidth semantic data.

Experimental Setup: This experiment addresses the challenge of transmitting high-density LiDAR data (Livox Mid360, ∼300k points/sec) over a fluctuating 5G uplink for remote object detection. We implement a latency-driven feedback loop targeting the Voxel Grid Filter's `leaf_size` parameter. Unlike standard ROS which uses a static value (e.g., 0.1 m), MobileROS adjusts the leaf size (L_{curr}) based on the current Channel Buffer Occupancy (B_{occ}):

$$L_{curr} = L_{base} \cdot (1 + \beta \cdot B_{occ})$$

where L_{base} is the default leaf size (0.1 m), $B_{occ} \in [0, 1]$ is the channel buffer occupancy ratio, and β controls sensitivity (default: 1.5). As network contention rises, the leaf size increases, effectively reducing the point cloud density at the source to guarantee that the transmission latency remains within safety bounds (< 100 ms).

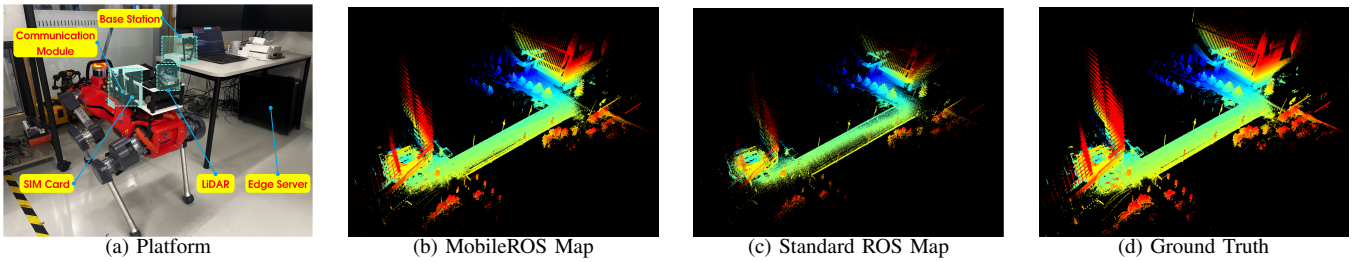
We compare three configurations: (1) **Standard ROS** with default CycloneDDS settings and no adaptation, (2) **Static Allocation** with a fixed high-priority network slice but no source-side compression, and (3) **MobileROS** with full source-side adaptation enabled.

Results and Analysis: We subjected all three systems to a synthetic interference pattern that periodically reduced the available uplink bandwidth from 20 Mbps to 3 Mbps. As shown in the aggregated results in Table V, MobileROS reduces per-frame latency by ≈66% (from 111 ms to 37.7 ms) compared to Standard ROS. The qualitative impact is visible in Fig. 8: Standard ROS (Fig. 8c) exhibits severe map fragmentation because the middleware indiscriminately dropped packets once the latency budget was exceeded. In contrast, MobileROS (Fig. 8b) proactively thinned the point cloud *before* transmission. Although the resolution dropped, the structural integrity was preserved, allowing the SLAM backend to maintain loop closure.

Discussion: The key insight is that Static Allocation alone is insufficient. While it improved latency (65.3 ms vs. 111 ms) by reserving network resources, it still suffered from frame drops (2.10%) because the raw data volume exceeded the allocated capacity during bandwidth dips. MobileROS, by contrast, actively reduced the data volume at the source, achieving the lowest frame drop rate (0.67%) and the highest detection accuracy (AP@IoU=0.5: 89.6% vs. 77.2%). This demonstrates that **source-side semantic adaptation** is essential for high-bandwidth robotic data, complementing but not replaceable by network-layer QoS.

TABLE V: Performance comparison: Standard ROS vs. MobileROS (Aggregated Statistics). Arrows (↑/↓) indicate better direction.

Metric	**Standard ROS** *(Mean±Std)*	**Static Alloc.** *(Mean±Std)*	**MobileROS (Ours)** *(Mean±Std)*
Network Performance			
Avail. UL Rate (Mbps) ↑	3.17 ± 0.15	6.40 ± 0.10	$\mathbf{9.53 \pm 0.15}$
UL Latency (ms) ↓	52.3 ± 2.5	29.0 ± 1.0	$\mathbf{15.0 \pm 1.0}$
Packet Loss Rate (%) ↓	3.50 ± 0.30	1.87 ± 0.15	$\mathbf{0.50 \pm 0.10}$
Jitter (ms) ↓	19.0 ± 1.0	9.3 ± 0.6	$\mathbf{4.7 \pm 0.6}$
3D Detection Accuracy (AP)			
AP@IoU=0.5 (All) ↑	77.2 ± 0.9	84.5 ± 0.6	$\mathbf{89.6 \pm 0.5}$
AP@IoU=0.7 (Cars) ↑	64.2 ± 0.8	72.8 ± 0.3	$\mathbf{80.5 \pm 0.5}$
System Efficiency			
Detection FPS ↑	9.2 ± 0.3	14.6 ± 0.2	$\mathbf{17.8 \pm 0.4}$
Frame Drop Rate (%) ↓	5.77 ± 0.25	2.10 ± 0.10	$\mathbf{0.67 \pm 0.15}$
Per-Frame Latency (ms) ↓	111.0 ± 3.6	65.3 ± 2.5	$\mathbf{37.7 \pm 2.5}$



(a) Platform (b) MobileROS Map (c) Standard ROS Map (d) Ground Truth

Fig. 8: LiDAR-based mapping performance. MobileROS (b) uses adaptive voxel filtering to preserve structure, while Standard ROS (c) suffers fragmentation due to bufferbloat.

C. Experiment C: Active Network Control in V2X

Hypothesis: Robotic applications can act as "Network Controllers" to actively reconfigure 5G slices for safety-critical tasks.

Experimental Setup: The experiment evaluates MobileROS's ability to perform network-aware decision-making. We validate the **Application-Driven Bidirectional Slicing** mechanism described in Section III-D. The MobileROS Hub orchestrates the system, managing interaction between vehicle nodes and infrastructure units. The testbed (Fig. 9) simulates an infrastructure-assisted autonomous driving scenario where an edge server (RSU) provides global localization corrections to a vehicle.

Baseline Configurations: We compare three configurations: (1) **Standard ROS 2** with default DDS settings and no network awareness, (2) **MobileROS (Passive)** with Tier 1 reactive adaptation only (no active slicing), and (3) **MobileROS (Active)** with both tiers enabled including bidirectional slice negotiation.

Results and Analysis: First, we analyze the impact of latency on localization. Fig. 10 presents a temporal analysis of localization performance. MobileROS maintains substantially better performance than standard approaches as latency increases (Fig. 11).

Two-Tiered Adaptation Strategy: MobileROS employs a hierarchical defense against network instability, implementing the design patterns introduced in Section IV-G.

- **Tier 1: Reactive Adaptation (Learning Phase).** As shown in Fig. 12, the vehicle initially experiences high jitter. At the point marked by the star, the Hub's *Policy Engine* identifies the pattern of network instability and switches the transmission strategy from "Raw Data" to "Feature Only," instantly stabilizing the control loop. This handles routine fluctuations without requiring network reconfiguration.
- **Tier 2: Proactive Slicing (Emergency Phase).** To validate the "Active Control" contribution on top of 5G, we simulated an "Emergency Stop" scenario. We injected 15 Mbps of background video traffic to saturate the default Best-Effort slice, then triggered an obstacle detection event (collision risk $> 80\%$). Upon detecting the obstacle, MobileROS bypassed the Tier 1 adaptation and triggered a **Slice Upgrade** via the UE-gNB Slice Manager protocol. Network logs confirm that the gNB reallocated Physical Resource Blocks (PRBs) within **120 ms**, preempting background traffic to prioritize the robot's safety command.

TABLE VI: Active Slicing Performance under Network Congestion (Emergency Stop Scenario).

Metric	Standard ROS 2	MobileROS (Passive Only)	MobileROS (Active Slicing)
Network Response			
Slice Upgrade Capability	✗	✗	✓
Slice Reconf. Time	N/A	N/A	**120 ± 15**ms
PRB Allocation (Peak)	10% (Best Effort)	10% (Best Effort)	**45% (Priority)**
End-to-End Performance			
E2E Latency (Normal) ↓	85 ± 12 **ms**	62 ± 8 **ms**	$\mathbf{58 \pm 6}$ **ms**
E2E Late. (Congestion) ↓	520 ± 95 **ms**	285 ± 35 **ms**	**148 ± 18**ms
Latency Spike Ratio	6.1×	4.6×	**2.6**×
Safety-Critical Task Delivery			
Emergency Cmd Success Rate ↑	62.5%	87.5%	**100**%
Average Cmd Delivery Time ↓	380 ± 85 **ms**	195 ± 40 **ms**	$\mathbf{92 \pm 12}$ **ms**
Obstacle Response (Pass/Fail)	Fail (3/8)	Pass (7/8)	**Pass (8/8)**

Discussion: Table VI quantifies the benefit of Active Slicing. The key observation is the **Latency Spike Ratio** during congestion: Standard ROS 2 experienced a 6.1× latency increase (from 85 ms to 520 ms), while MobileROS (Active) limited this to 2.6× (from 58 ms to 148 ms) by dynamically securing priority PRBs.

Critically, the **Emergency Command Success Rate** demonstrates the safety implication: Standard ROS 2 failed to deliver 3 out of 8 emergency stop commands within the 200 ms deadline, while MobileROS (Active) achieved 100% delivery with an average latency of only 92 ms. A standard 5G middleware, lacking the semantic trigger from the robotic application layer, would have queued the safety command behind background traffic. This active reconfiguration capability is further illustrated in Fig. 13, showing how dynamic allocation maintains end-to-end latency below 150 ms even during congestion peaks.

D. Experiment D: Hybrid Multi-Robot Collaborative Localization

Hypothesis: MobileROS's Source-Side Adaptation generalizes beyond single-agent cellular scenarios to heteroge-

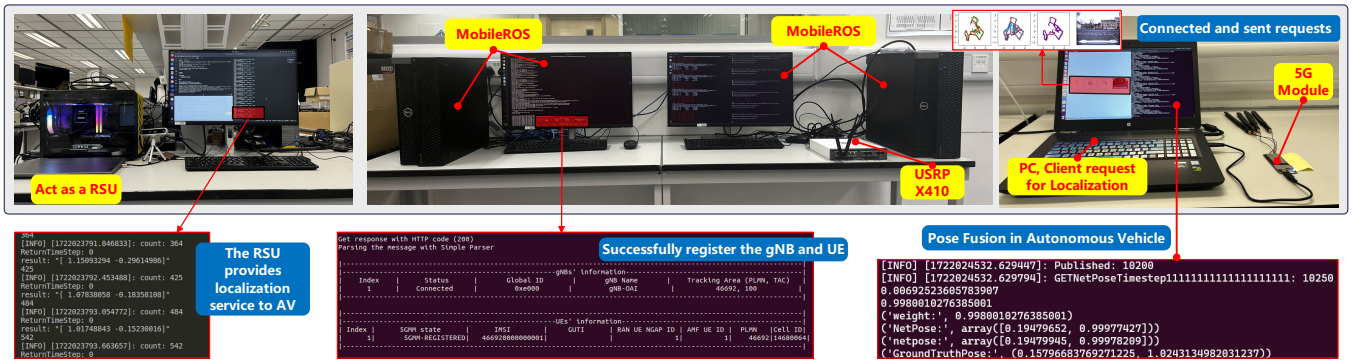
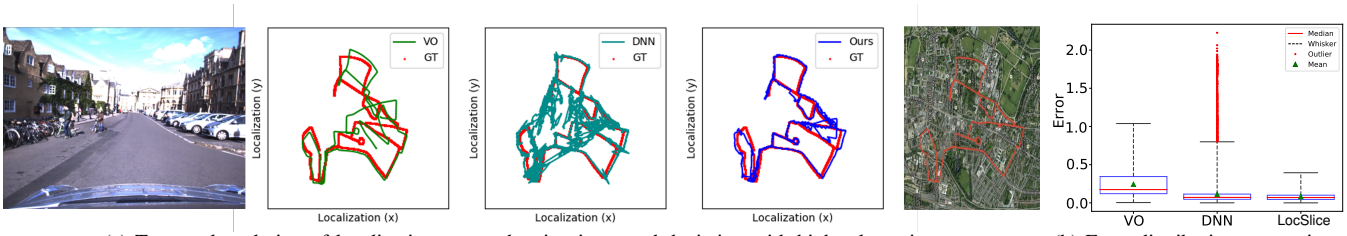


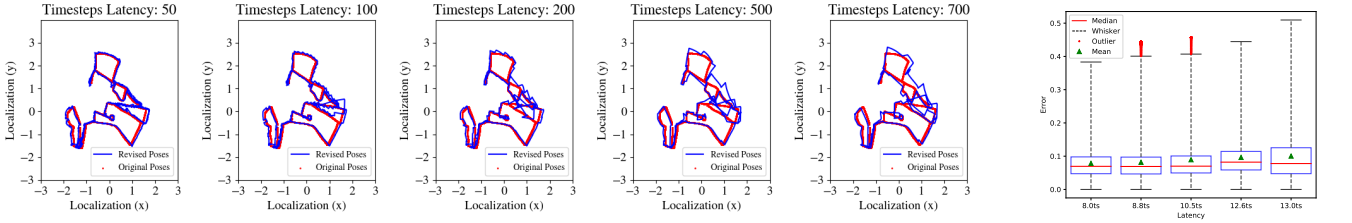
Fig. 9: V2X Experiment Deployment. Validating the Application-Driven Bidirectional Slicing capability.



(a) Temporal evolution of localization errors showing increased deviation with higher latencies.

(b) Error distribution comparison.

Fig. 10: Analysis of localization performance. The three trajectory visualizations represent VO, DNN, and combined MobileROS estimates.



(a) Trajectory analysis showing positioning performance degradation under increasing network latencies from 50 ms to 700 ms.

(b) Latency-localization accuracy correlation.

Fig. 11: Impact of network latency on localization accuracy.

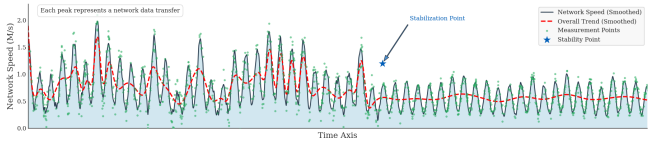


Fig. 12: Latency fluctuations in the online learning process. The star marker indicates when MobileROS adapts.

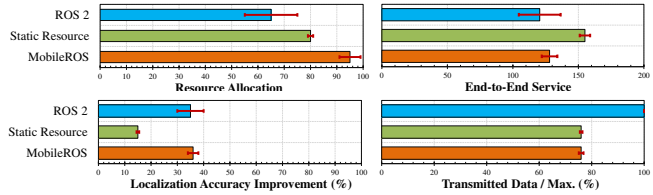


Fig. 13: System-level performance metrics. MobileROS consistently outperforms both standard ROS 2 and static resource allocation.

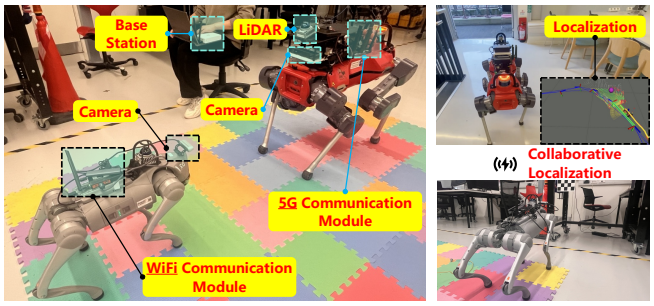


Fig. 14: (Left): Setup of the WiFi and 5G robots; (right-top): visual localization trajectory via matching; (right-bottom): the WiFi-equipped robot.

neous (WiFi + 5G) multi-robot coordination.

Experimental Setup: To validate the extensibility of MobileROS beyond single-agent cellular scenarios, we conducted a collaborative localization experiment involving heterogeneous networks. The setup involves two robots: Robot A (Outdoor, 5G-connected) and Robot B (Indoor, WiFi-connected). Robot A acts as a "scout," sending visual feature maps to Robot B to assist in relocalization. This requires reliably transmitting feature descriptors over a hybrid, potentially unreliable link. The WiFi link (2.4 GHz, 802.11n) exhibited a measured packet loss rate of 8–12% and average RTT of 25

ms under nominal conditions, degrading significantly during interference.

We compared MobileROS against three baselines: (1) **Legacy ROS 1** using TCP transport, (2) **Standard ROS 2** with CycloneDDS (UDP multicast), and (3) **WebRTC Bridge** using Google Congestion Control (GCC). Each system transmitted identical visual feature data (ORB descriptors, $\sim$2 MB/s) over a 5-minute trial with 10 repetitions.

Results and Analysis: Table VII presents the quantitative comparison across all four configurations.

- **The DDS Failure (ROS 2):** The standard ROS 2 stack relies on UDP multicast with reliable delivery semantics. In lossy WiFi environments, this triggered a **retransmission storm**, causing severe bufferbloat that spiked latency to 480 ms and increased localization error to 0.42 m.
- **The Transport Failure (WebRTC):** While the WebRTC bridge maintained low latency (55 ms) through aggressive rate adaptation, its **content-agnostic** congestion control (GCC) randomly dropped packets without considering semantic importance. This resulted in corrupted feature descriptors, leading to a high localization error of 0.38 m despite good network metrics.
- **The MobileROS Advantage:** By proactively downsampling the visual stream at the source (Source-Side Adaptation), MobileROS achieved comparable latency (45 ms) while maintaining significantly higher precision (0.06 m) because it preserved the *structural integrity* of transmitted features.

TABLE VII: Multi-Robot Heterogeneous Network Performance Comparison. Results are reported as mean $\pm$ std over 10 trials.

Metric	ROS 1 (TCP)	ROS 2 (DDS)	WebRTC (P2P/GCC)	MobileROS
Localization Accuracy				
Trans. Error E_t (m) $\downarrow$	0.49 ± 0.08	0.42 ± 0.06	0.38 ± 0.07	$\mathbf{0.06 \pm 0.02}$
Rot. Error E_r (deg) $\downarrow$	8.11 ± 1.2	4.38 ± 0.9	5.12 ± 1.1	$\mathbf{0.64 \pm 0.15}$
Network Performance				
Transport Latency (ms) $\downarrow$	>500	480 ± 55	55 ± 12	$\mathbf{45 \pm 8}$
Eff. Throughput (Mbps) $\uparrow$	0.8 ± 0.2	1.2 ± 0.3	1.6 ± 0.2	$\mathbf{1.8 \pm 0.15}$
Data Quality				
Feature Match Rate (%) $\uparrow$	45 ± 8	52 ± 10	61 ± 7	$\mathbf{94 \pm 3}$
Data Integrity	Delayed	Delayed	Corrupted	**Preserved**

Discussion: The 7$\times$ improvement in translation error (0.06 m vs. 0.42 m for ROS 2) and 7$\times$ improvement in rotation error (0.64° vs. 4.38° for ROS 2) demonstrate the critical importance of **semantic-aware adaptation**. The key insight is revealed by the **Feature Match Rate**: despite WebRTC achieving low latency, its random packet drops resulted in only 61% successful feature matches, whereas MobileROS achieved 94% by intelligently selecting which data to transmit.

This experiment confirms that generic transport protocols optimized for streaming media (WebRTC) or reliable delivery (DDS) are insufficient for robotic data that requires *semantic consistency*. Application-layer adaptation that understands the structure of robotic data is essential for multi-robot coordination over heterogeneous, unreliable networks.

E. Experiment E: Network Partition and Failsafe Recovery

Hypothesis: A Dynamic CAP Strategy allows the system to survive Single Point of Failure (SPOF) scenarios by shifting consistency requirements at runtime.

Experimental Setup: To rigorously evaluate the system's resilience to the CAP theorem constraints raised in Section III-G, we simulated a network partition where the connection to the central Hub was abruptly severed. The experiment proceeded in three phases:

- **Normal Phase** (t = 0–15 s): Full connectivity with Hub coordination.
- **Partition Phase** (t = 15–45 s): Hub link severed; robot operates autonomously.
- **Recovery Phase** (t = 45–60 s): Link restored; system resynchronizes.

We compared Standard ROS 2 (Nav2 Stack with default DDS timeout settings) against MobileROS equipped with the local Circuit Breaker mechanism described in Section III. Each configuration was tested over 10 trials on a ground robot navigating a 50 m indoor corridor.

Results and Analysis: Table VIII presents the quantitative comparison of system resilience under network partition.

- **Standard ROS 2 Failure (Freeze):** When the link was cut, the standard ROS 2 local planner continued to wait for global path updates until a DDS timeout (5.0 s), eventually entering an abort state as recovery behaviors failed without global guidance. The robot stopped indefinitely, requiring manual intervention.
- **MobileROS Recovery (Degraded Mode):** MobileROS detected the PHY-layer signal loss in $<$ 100 ms via the Radio Information Engine's heartbeat monitor. It immediately triggered the Circuit Breaker, switching to **Degraded Mode**: falling back to Visual Odometry (VO) for state estimation and executing a **Safe-Hold maneuver** (controlled braking) to prevent collision.
- **Graceful Reconnection:** Upon link restoration at $t = 45$ s, MobileROS automatically resynchronized with the Hub within 350 ms, resuming normal CP-oriented operation without requiring a system restart.

Discussion: From a CAP theorem perspective, this experiment validates the Dynamic CAP Strategy introduced in Section III-G. The key results are:

- **Detection Speed:** MobileROS detected the partition 59$\times$ faster (85 ms vs. 5000 ms) by monitoring PHY-layer signals rather than waiting for middleware timeouts.
- **Safety Guarantee:** Standard ROS 2 caused 2 collision events (out of 10 trials) due to the robot continuing on

TABLE VIII: System Resilience under Network Partition (Single Point of Failure Simulation). Results over 10 trials.

Metric	Standard ROS 2 (Nav2 + DDS)	MobileROS (Circuit Breaker)
Detection and Response		
Partition Detection Time ↓	5000 ± 120 ms (Timeout)	$\mathbf{85 \pm 12}$ **ms** (PHY)
Mode Transition Time ↓	N/A (No Fallback)	$\mathbf{45 \pm 8}$ **ms**
Behavior during Outage	Freeze → Abort	**Degraded (Safe-Hold)**
Safety and Availability		
Collision Events	2 / 10 trials	**0 / 10 trials**
System Liveness	0% (Full Stop)	**100% (Local Active)**
Safe-Hold Success Rate ↑	N/A	**100%**
Recovery Performance		
Auto-Recovery Success ↑	0% (Manual Reset)	**100%**
Resync Time after Reconnect ↓	N/A	$\mathbf{350 \pm 45}$ **ms**
State Drift during Partition ↓	N/A (Aborted)	$\mathbf{0.12 \pm 0.03}$ **m**
CAP Trade-off Strategy		
Normal Mode	CP (Consistency)	CP (Consistency)
Partition Mode	*Sacrificed Availability*	**AP (Availability)**

stale trajectories before abort. MobileROS achieved zero collisions through immediate Safe-Hold activation.

- **Bounded Drift:** During the 30 s partition, MobileROS's local VO fallback limited state estimation drift to only 0.12 m, enabling seamless resynchronization.

This experiment demonstrates that while the Hub remains a *logical* center for global optimization, it is **not** a physical single point of failure. MobileROS dynamically shifts from CP-oriented (Normal) to AP-oriented (Degraded) behavior, ensuring that **local safety is never compromised by global partition**, a critical requirement for deploying centralized coordination in real-world mobile robotics.

VII. Network Analysis of MobileROS

To complement the application-level experiments, we conducted a focused micro-benchmark evaluation to characterize the system's fundamental networking behavior. Network performance evaluation was not conducted concurrently in the previous experiments, as the complex robotic applications running concurrently would significantly confound fundamental network measurements. Therefore, this section presents a controlled analysis designed to isolate the "Symbiotic" adaptation mechanisms from robotic task logic.

Experimental Setup: The setup employed a USRP B210 software-defined radio controlled by the **OpenAirInterface (OAI)** soft-modem stack, running on an Intel Core i7-9700K workstation (32GB RAM) as the network controller. MobileROS was compared against a standard ROS 2 Foxy distribution (using the default CycloneDDS profile) to quantify the benefits of Physical Layer awareness over standard middleware QoS. The objective was to measure MobileROS's adaptation latency, overhead, and throughput efficiency in isolation from heavy SLAM or perception algorithms.

In each experiment, a single MobileROS node transmitted messages to a receiving node at a fixed frequency of 10 Hz for 10 minutes per network condition. Five predefined message categories were used: Control Messages ($\approx$ 100 bytes), Perception Data ($10-50$ KB), Mapping Data ($20-40$ KB), Planning Data ($5-15$ KB), and Telemetry Data ($\approx$ 200 bytes).

Test Duration and Worst-Case Characterization: Each micro-benchmark ran for 10 minutes per network condition at 10 Hz, yielding approximately 6,000 samples per configuration. Beyond mean$\pm$std statistics reported throughout this section, we characterize tail latency to assess worst-case behavior. MobileROS achieves P95 = 22 ms, P99 = 38 ms, and a maximum observed latency of 65 ms, compared to Standard ROS 2's P95 = 185 ms, P99 = 420 ms, and maximum of 1,250 ms. The absence of extreme tail spikes in MobileROS is attributed to the source-side adaptation preventing buffer buildup during channel congestion.

Through systematic measurements compared against standard ROS 2 (Fig. 15), four critical aspects were evaluated:

1) **Network Adaptation:** MobileROS validates the "Symbiotic Paradigm" by dynamically adjusting compression ratios. Unlike ROS 2's middleware-only approach which achieved static efficiency (45-60%), MobileROS utilized PHY-layer feedback to achieve 73% higher effective throughput and maintain allocation efficiency above 85% across varying channel conditions.
2) **Minimal System Overhead:** Despite the active monitoring by the Radio Information Engine, system overhead remains minimal with only 8% increased CPU utilization and $15 - 20$ MB memory consumption. Crucially, the "Application-Driven" adaptation loop allows detection of network changes in $50 - 100$ ms and implementation in $150 - 200$ ms, significantly outperforming ROS 2's $800 - 1200$ ms timeout-based response time.
3) **Cross-Layer Optimization:** By bridging the gap between the Application and Physical layers, MobileROS maintains above 85% completion rates at 80% network utilization, while standard ROS 2 drops to 40% due to bufferbloat opaque to the DDS middleware.
4) **Content-Aware Encoding:** Through content-aware encoding, MobileROS reduces total data volume by 62% while maintaining comparable task performance. This confirms the efficacy of the "Adaptive Compression" strategy, where the system intelligently trades spatial resolution for temporal continuity to preserve structural information.

Table IX quantifies these capabilities. The data shows significant variations in bandwidth allocation patterns (e.g., Control Messages ranging from $16.2 \pm 1.8\%$ to $4.8 \pm 0.9\%$, while Mapping Data shifts from $24.3 \pm 1.7\%$ to $36.3 \pm 1.8\%$). The system dynamically adjusts message priorities based on

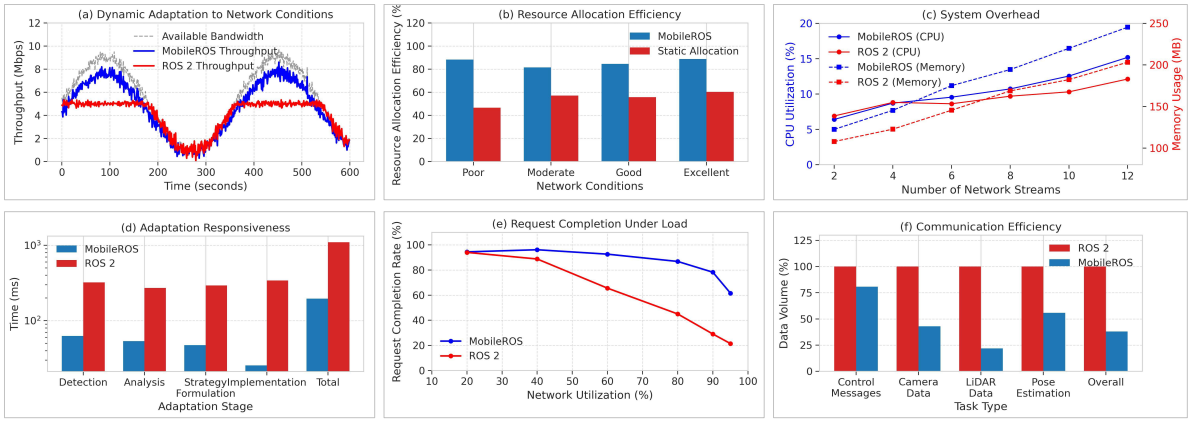


Fig. 15: Network performance analysis of MobileROS: Comparing (a) dynamic adaptation to network conditions, (b) resource allocation efficiency, (c) adaptation responsiveness, (d) request completion under load, (e) system overhead, and (f) communication efficiency with ROS 2.

TABLE IX: MobileROS Network Resource Management Analysis

Message Type	**Bandwidth Allocation (%)**			**Priority by Network Condition (0-100)**				
	High	**Medium**	**Low**	**V. Poor**	**Poor**	**Moderate**	**Good**	**Excellent**
Control Messages	16.2 ± 1.8	9.7 ± 1.2	4.8 ± 0.9	94 ± 3	89 ± 2	83 ± 3	78 ± 2	73 ± 4
Perception Data	34.6 ± 2.3	31.2 ± 1.5	24.1 ± 2.1	42 ± 3	52 ± 4	68 ± 3	87 ± 3	93 ± 2
Mapping Data	24.3 ± 1.7	29.5 ± 2.2	36.3 ± 1.8	21 ± 2	32 ± 3	61 ± 2	83 ± 4	96 ± 2
Planning Data	14.8 ± 1.5	19.2 ± 1.7	26.4 ± 1.9	37 ± 2	46 ± 3	67 ± 4	79 ± 3	91 ± 3
Telemetry Data	10.3 ± 0.8	9.6 ± 1.1	9.8 ± 0.7	76 ± 3	71 ± 2	64 ± 3	61 ± 2	57 ± 3

network quality; control messages maintain critical priority (94 ± 3) in very poor conditions but gradually reduce priority (to 73 ± 4) as conditions improve, allowing perception data priority to increase dramatically (from 42 ± 3 to 93 ± 2). These measurements validate the core architecture: transforming opaque wireless communication into a transparent resource that enables intelligent cross-domain optimization impossible in traditional black-box approaches.

VIII. Conclusion

This paper introduces MobileROS, a wireless-native robotic operating system grounded in the Symbiotic Paradigm that transforms cellular communication from an external utility into a core robotic resource. By establishing bidirectional information coupling between the physical layer and application logic, MobileROS enables robots to function as active network controllers capable of semantics-driven resource negotiation and proactive adaptation. Our experimental validation confirms that this cross-layer co-design significantly enhances system robustness and safety in challenging network environments compared to standard strictly decoupled architectures. Our findings suggest that balancing architectural modularity with runtime information symbiosis is essential for the next generation of mobile robotics, and we provide the complete open-source implementation to facilitate community adoption at https://github.com/MobileROS.

References

[1] Andrea Bonci, Francesco Gaudeni, Maria Chiara Giannini, and Sauro Longhi. Robot operating system 2 (ros2)-based frameworks for increasing robot autonomy: A survey. *Applied Sciences*, 13(23):12796, 2023. doi: 10.3390/app132312796.

[2] Lorenzo Bravi. *5G-Enabled UAV System: Implementation, Deployment and Analysis*. PhD thesis, Politecnico di Torino, 2024.

[3] Eric Brewer. Cap twelve years later: How the "rules" have changed. *Computer*, 45(2):23–29, 2012. doi: 10.1109/MC.2012.37.

[4] He Chen, Rana Abbas, Peng Cheng, et al. Ultra-reliable low latency cellular networks: Use cases, challenges and approaches. *arXiv preprint arXiv:1709.00560*, 2016. doi: 10.48550/arXiv.1709.00560.

[5] Kaiyuan Eric Chen, Yafei Liang, Nikhil Jha, et al. Fogros: An adaptive framework for automating fog robotics deployment. In *2021 IEEE 17th International Conference on Automation Science and Engineering (CASE)*, pages 2035–2042, 2021. doi: 10.1109/CASE49439.2021.9551644.

[6] Chien-Fu Cheng and Bo-Ming Chen. Wireless sensor networks: Target-barrier coverage with static and mobile sensors. In *Proceedings of the 2023 12th International Conference on Networks, Communication and Computing*, pages 1–5, 2023.

[7] Angelo Corsaro, Luca Cominardi, and Gabriele Baldoni. Eclipse zenoh: A next-generation protocol for iot and ros 2. In *Workshop on Open-Source Software for Robot Operating Systems (ROSCon)*, 2023. URL https://github.com/eclipse-zenoh/zenoh-plugin-ros2dds.

[8] Luca De Cicco, Gaetano Carlucci, and Saverio Mascolo. Understanding the Google congestion control for RTC video. *ACM Transactions on Multimedia Computing, Communications, and Applications (TOMM)*, 14(1s):1–23, 2018.

[9] Kaikai Deng, Dong Zhao, Qiaoyue Han, et al. Midas: Generating mmwave radar data from videos for training pervasive and privacy-preserving human sensing tasks. *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies*, 7(1):1–26, 2023.

[10] Eric Evans. *Domain-Driven Design: Tackling Complexity in the Heart of Software*. 2003. ISBN 0321125215.

[11] Alessio Fascista. Toward integrated large-scale environmental monitoring using wsn/uav/crowdsensing: A review of applications, signal processing, and future perspectives. *Sensors*, 22(5):1824, 2022. doi: 10.3390/s22051824.

[12] Ramkumar Gandhinathan and Lentin Joseph. *ROS Robotics Projects*. Sebastopol, CA, 2020. ISBN 9781838649326. URL https://www.oreilly.com/library/view/ros-robotics-projects/9781838649326/.

[13] M. Ginting, K. Otsu, J. Edlund, et al. Chord: Distributed data-sharing via hybrid ros 1 and 2 for multi-robot exploration of large-scale complex environments. *IEEE Robotics and Automation Letters*, 6:5064–5071, 2021. doi: 10.1109/LRA.2021.3061393.

[14] Ismael Gomez-Miguelez, Andres Garcia-Saavedra, Paul D. Sutton, et al. srsLTE: An open-source platform for LTE evolution and experimentation. In *the 2016 ACM Workshop on Wireless Network Testbeds, Experimental evaluation & Characterization (WiNTECH '16)*, pages 25–32, 2016. doi: 10.1145/2980159.2980163.

[15] Han Hao, Wei Xi, Andreas Kuster, et al. Mc-lora: Multi-node concurrent localization for lorawan indoors and outdoors. *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies*, 9(1):1–32, 2025.

[16] Bishal Jaiswal, Himanshu Tyagi, Aditya Gopalan, et al. Wiros: A qos software solution for ros2 in a wifi network. *2023 15th International Conference on COMmunication Systems & NETworkS (COMSNETS)*, pages 216–218, 2023.

[17] Jongkil Kim, J. Smereka, Calvin Cheung, et al. Security and performance considerations in ros 2: A balancing act. *ArXiv*, abs/1809.09566, 2018.

[18] B Ravi Kiran, Ibrahim Sobh, Victor Talpaert, Patrick Mannion, Ahmad A Al Sallab, Senthil Yogamani, and Patrick Pérez. Deep reinforcement learning for autonomous driving: A survey. *IEEE transactions on intelligent transportation systems*, 23(6):4909–4926, 2021.

[19] Tobias Kronauer, Joshwa Pohlmann, Maximilian Matthé, et al. Latency analysis of ros2 multi-node systems. *2021 IEEE International Conference on Multisensor Fusion and Integration for Intelligent Systems (MFI)*, pages 1–7, 2021. doi: 10.1109/MFI52462.2021.9591166.

[20] Adam Langley, Alistair Riddoch, Alyssa Wilk, et al. The QUIC transport protocol: Design and Internet-scale deployment. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication (SIGCOMM '17)*, pages 183–196, 2017. doi: 10.1145/3098822.3098842.

[21] Bin Li, Zesong Fei, and Yan Zhang. Uav communications for 5g and beyond: Recent advances and future trends. *IEEE Internet of Things Journal*, 6(2):2241–2263, 2019.

[22] Salvatore Loreto and Simon Pietro Romano. Real-time communication with WebRTC. *IEEE Communications Magazine*, 52(4):100–109, 2014.

[23] Steven Macenski, Tully Foote, Brian Gerkey, Chris Lalancette, and William Woodall. Robot operating system 2: Design, architecture, and uses in the wild. *Science Robotics*, 7(66):eabm6074, 2022. doi: 10.1126/scirobotics.abm6074.

[24] Yuya Maruyama, S. Kato, and Takuya Azumi. Exploring the performance of ros2. *2016 International Conference on Embedded Software (EMSOFT)*, pages 1–10, 2016. doi: 10.1145/2968478.2968502.

[25] Sam Newman. *Building Microservices: Designing Fine-Grained Systems*. 2015. ISBN 1491950358.

[26] Navid Nikaein, Mahesh K. Marina, Saravana Manickam, et al. Openairinterface: A flexible platform for 5g research. *ACM SIGCOMM Computer Communication Review*, 44(5):33–38, 2014.

[27] Benjamin Nowak. Precision agriculture: Where do we stand? a review of the adoption of precision agriculture technologies on field crops farms in developed countries. *Agricultural Research*, 10(4):515–522, 2021. doi: 10.1007/s40003-021-00539-x.

[28] Morgan Quigley, Ken Conley, Brian Gerkey, et al. Ros: An open-source robot operating system. *The International Conference on Robotics and Automation(CIRA) workshop on open source software*, 3(3.2):5, 2009.

[29] Wim Rouwet. *Open radio access Network (O-RAN) systems architecture and design*. 2025.

[30] Eyad Shaklab, Areg Karapetyan, Arjun Sharma, et al. Towards autonomous and safe last-mile deliveries with ai-augmented self-driving delivery robots. *arXiv preprint arXiv:2305.17705*, 2023. doi: 10.48550/arXiv.2305.17705.

[31] Anna Wojciechowska, Jeremy Frey, Esther Mandelblum, et al. Designing drones: Factors and characteristics influencing the perception of flying robots. *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies*, 3(3):1–19, 2019.

[32] Guodong Zhao, Muhammad Ali Imran, Zhibo Pang, et al. Toward real-time control in future wireless networks: Communication-control co-design. *IEEE Communications Magazine*, 57(2):138–144, 2019. doi: 10.1109/MCOM.2018.1800163.