applied sciences

MDPI

Article

A Scalable Heterogeneous Interoperable Communication Framework for Multi Unmanned Systems

Sojeong Kim [1], Seeon Lee [2] and Ok-kyoon Ha [2,*]

[1] Department of Industrial AI Convergence Engineering, Kyungwoon University, Gumi 39160, Republic of Korea; ksj20010@naver.com
[2] Department of Software, Kyungwoon University, Gumi 39160, Republic of Korea; skskt123499@gmail.com
* Correspondence: okha@ikw.ac.kr

Abstract

Robotics technology is rapidly advancing across various fields, including manufacturing, logistics, defense, surveillance, and disaster response. However, the integrated operation of heterogeneous robotic systems remains challenging due to differences in communication protocols and system architectures. This study proposes the Scalable Heterogeneous Interoperable Communication Framework (SHICF), which adapts the interoperability concepts of STANAG 4586 to unmanned ground vehicle systems. The proposed framework employs a unified binary message structure incorporating a common header, message identifiers, and vehicle identifiers to enable data exchange across heterogeneous systems. Platform-specific interfaces collect robot status information, including position, velocity, battery level, and inertial sensor data, and encode it into standardized binary messages for transmission to a web-based Ground Control System over UDP. SHICF was implemented on heterogeneous UGVs running ROS Melodic and ROS 2 Humble and evaluated against corresponding Native communication configurations based on rosbridge, JSON serialization, and WebSocket transport. The experimental evaluation compared the Native and SHICF configurations independently within each ROS environment rather than directly comparing the two ROS platforms. The results demonstrated that a single GCS could identify and monitor heterogeneous UGVs in real time. In ROS Melodic, the measured message loss rate decreased from 0.45% in the Native configuration to 0.20% with SHICF. Similarly, in ROS 2 Humble, the message loss rate decreased from 1.42% to 0.20%. These findings demonstrated the feasibility of integrated communication and monitoring among heterogeneous UGVs.

Keywords: heterogeneous multi-robot systems; unmanned ground vehicle (UGV); ground control system (GCS); ROS interoperability; STANAG 4586; UDP communication

Check for updates

Academic Editor: Yutaka Ishibashi

Received: 17 September 2026
Revised: 16 October 2026
Accepted: 19 October 2026
Published: 24 October 2026

Copyright: © 2026 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license.

1. Introduction

Robotics technology is rapidly advancing across various fields, including manufacturing, logistics, defense, surveillance, and disaster response. Unmanned Ground Vehicles (UGVs) are particularly valuable because they can perform tasks such as reconnaissance, transportation, facility inspection, and disaster response in environments that are difficult or hazardous for humans to access. As robotic applications continue to expand, the operational paradigm is shifting from single-robot operation toward the coordinated deployment of multiple heterogeneous robots with different hardware and software environments [1]. This transition requires an integrated communication framework capable of collecting and

managing robot status and sensor data in real time. For the effective operation of multiple UGVs, a Ground Control System (GCS) must be capable of collecting and managing key status information, including vehicle position, speed, battery status, and Inertial Measurement Unit (IMU) data, in real time. The Robot Operating System (ROS) is a widely used robotics software framework that supports data exchange among sensors, controllers, and applications [2]. Its topic-based publish–subscribe architecture provides a modular structure for developing and extending robotic systems. However, heterogeneous robotic systems may use different communication architectures and protocols. ROS relies on ROS Master and communication protocols such as TCPROS and UDPROS, whereas ROS 2 adopts a distributed communication architecture based on the Data Distribution Service (DDS) [3,4].

These architectural differences generally require bridges, message conversion modules, or additional middleware to integrate ROS- and ROS 2-based robots into a common GCS. Existing ROS-based integration methods typically convert or relay data while preserving individual ROS topic structures. When position, velocity, battery, and IMU data are transmitted through separate topics, each topic requires its own message structure and serialization and deserialization processes. In addition, direct integration of ROS topics with a web-based GCS generally requires middleware such as rosbridge and additional data conversion processes, including JSON serialization. Although these approaches provide flexibility and implementation convenience, they can increase data redundancy, message processing overhead, and communication complexity, particularly for UGVs with limited computational and network resources. As the number of robots and status messages increases, the number of communication connections and data conversion processes also increases, further complicating system integration. Various communication frameworks and standards have been investigated to support the interoperability of heterogeneous unmanned systems, including UGVs and unmanned aerial vehicles (UAVs) [5,6]. Among them, STANAG 4586 defines standardized message structures and levels of interoperability to support communication between UAVs and control systems. Its standardized messaging approach provides a useful basis for establishing interoperable unmanned systems independent of specific manufacturers and platforms. However, STANAG 4586 was primarily developed for UAV operations, and its original message structures and operational concepts are not directly optimized for real-time status collection from ROS-based UGVs or integration with web-based ground control environments [6–8]. In multi-UGV systems, increasing numbers of robots and sensor data types further increase the number of topic connections, message formats, and conversion processes that must be handled by the GCS. Therefore, a standardized communication structure that integrates essential operational data into common messages is required to reduce dependence on individual ROS topics and simplify heterogeneous system integration [9].

This study proposes the Scalable Heterogeneous Interoperable Communication Framework (SHICF), a lightweight communication framework that adapts the messaging and interoperability concepts of STANAG 4586 to heterogeneous UGV systems. SHICF converts status data generated by UGVs using different communication protocols into a unified binary message format and transmits the data to a GCS. Position, velocity, battery, and IMU data are consolidated into a single status report message, enabling the GCS to manage multiple UGVs independently of their underlying communication protocols and ROS versions. In addition, SHICF employs a common message structure containing a message identifier, sequence number, timestamp, and vehicle identifier to distinguish individual UGVs and manage message transmission. The main contributions of this study are as follows:

- This study proposes a lightweight and platform-independent communication framework that converts status data from heterogeneous UGVs into a common message format, enabling their integrated management through a single GCS.

- SHICF consolidates position, velocity, battery, and IMU data into a unified status message, reducing the structural complexity and data redundancy associated with transmitting operational data through multiple individual ROS topics.
- SHICF is experimentally evaluated using heterogeneous UGVs based on different ROS versions, demonstrating reliable real-time status collection through a single GCS.
- The framework is designed to support the future integration of additional robotic platforms and communication protocols without requiring fundamental changes to the overall communication architecture.

The remainder of this paper is organized as follows: Section 2 reviews related work on ROS-based robot communication, heterogeneous robot integration, and communication protocols for unmanned systems. Section 3 presents the overall architecture of SHICF and describes the design of its common header and status report messages. Section 4 describes the experimental environment, including heterogeneous UGVs and the web-based GCS, as well as the performance evaluation methodology. Section 5 presents and analyzes the comparative experimental results between native ROS topic-based communication and SHICF. Finally, Section 6 summarizes the findings and discusses future research directions.

2. Related Work

2.1. Communication Architectures of ROS and ROS 2

ROS is a widely used robotics software framework that supports data exchange among sensors, controllers, and applications. ROS employs a publish–subscribe communication model in which publisher nodes transmit data through specific topics and subscriber nodes receive the corresponding messages. This architecture enables individual functions of a robotic system to be implemented as independent nodes, supporting modular system development and functional extensibility [2,3]. Figure 1 illustrates the communication architecture of ROS Melodic. When a sensor node publishes data to a topic [2,5], the Publisher Node registers its node and topic information with the ROS Master using XML-RPC (eXtensible Markup Language–Remote Procedure Call). When a Subscriber Node subscribes to the topic, the ROS Master provides the Publisher's network information, allowing the Subscriber to establish a direct connection. Topic data are then transmitted directly between nodes via TCPROS or UDPROS. Thus, the ROS Master manages node and topic registration, discovery, and connection establishment rather than relaying topic data.

Figure 2 illustrates the communication architecture of ROS 2 Humble. Like ROS, ROS 2 uses a publish–subscribe model but differs in node discovery and connection establishment. Unlike ROS, which requires a centralized ROS Master, ROS 2 employs a distributed architecture based on the Data Distribution Service (DDS), enabling communication without a centralized Master. The DDS discovery mechanism uses the Participant Discovery Protocol (PDP) to discover participating DDS Participants and the Endpoint Discovery Protocol (EDP) to exchange and match Publisher and Subscriber endpoint information. Communication relationships are established by matching topic names, message types, and Quality of Service (QoS) policies. Topic data are subsequently transmitted directly between nodes via DDS/RTPS [2–5,9].

Because ROS and ROS 2 use different node discovery mechanisms, communication middleware, and data transmission architectures, direct communication between the two versions generally requires an additional bridge or message conversion layer. Therefore, integrating ROS- and ROS 2-based robots into a single GCS requires additional integration mechanisms to accommodate their different communication architectures and topic structures [10,11].

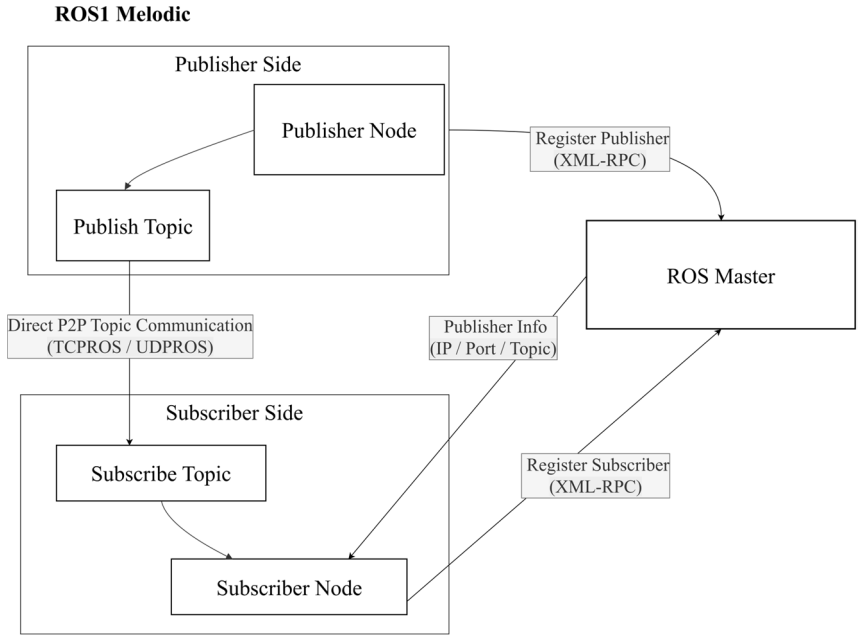


Figure 1. ROS Melodic Communication Architecture.

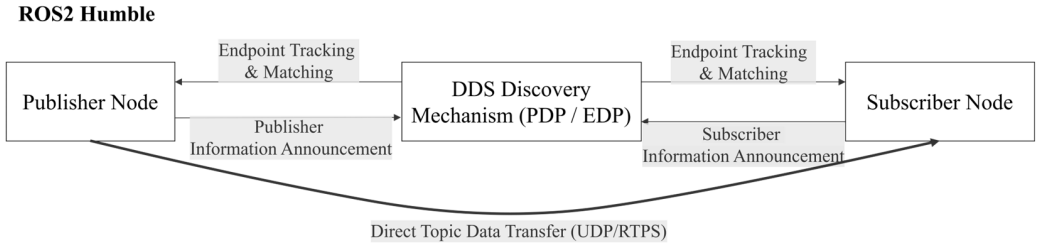


Figure 2. ROS 2 Humble Communication Architecture.

2.2. Main Drawbacks of Communication in Heterogeneous Robot Integration

Table 1 compares the structural characteristics of various communication methods. ros1_bridge supports interoperability between ROS and ROS 2 by converting and mapping compatible message types between the two environments [12]. In contrast, rosbridge provides an interface between ROS and external applications, particularly web-based systems, by exposing ROS data through standard web communication formats. MAVLink provides a lightweight message-based communication protocol for exchanging telemetry data and control commands between unmanned vehicles and a GCS. Within ROS environments, MAVLink can be integrated with ROS topics through additional interfaces such as MAVROS. Existing approaches for integrating ROS- and ROS 2-based robots generally rely on bridges, message conversion mechanisms, or middleware-specific interfaces. ros1_bridge enables ROS and ROS 2 nodes to exchange topic data while preserving their existing ROS-based structures [13]. This approach allows existing ROS nodes and topics to be reused across different ROS versions. Moreover, integration with a web-based GCS generally requires an additional interface, such as rosbridge, to convert and transmit ROS data in a format accessible to web applications. As the number of heterogeneous robots, sensors, and message types increases, the number of topics, message mappings, and communication interfaces that must be managed may also increase, resulting in greater system complexity. MAVLink addresses similar communication requirements through a lightweight and predefined message structure for telemetry and command exchange between unmanned vehicles and a GCS [14]. However, exchanging data between MAVLink-based systems and ROS topics requires an integration layer, such as MAVROS or a custom adapter, to map

and convert messages between the two interfaces. MAVLink and STANAG 4586 demonstrate the effectiveness of standardized message definitions for supporting interoperability between unmanned vehicles and control systems [15]. However, their message structures and operational models were primarily developed for UAV-oriented environments and are not directly optimized for the status information and operational characteristics of heterogeneous ROS-based UGVs [9].

Table 1. Comparison of Heterogeneous Robot Communication Approaches.

Category	ROS–ROS 2 Bridge [12]	Rosbridge-Based Approach [13]	MAVLink [14,15]
Primary purpose	Topic conversion between ROS and ROS 2	Integration between ROS and web applications	Communication between unmanned vehicles and GCSs
Transmission unit	Individual ROS topic messages	ROS topic messages encoded as JSON by default	Binary messages defined by a MAVLink dialect
ROS dependency	Requires ROS and ROS 2	Requires a ROS backend	-
Web integration	Requires additional implementation	WebSocket interface provided	Requires additional implementation
Multi-vehicle identification	Requires namespaces or similar mechanisms	Managed through topics and namespaces	System ID

2.3. STANAG 4586 and Unmanned System Interoperability

As shown in Figure 3, STANAG 4586 is a NATO standard that supports interoperability between heterogeneous unmanned aircraft systems (UASs) and control systems through standardized interfaces and message structures [16]. Its architecture includes a UAV Control System (UCS), which incorporates the Core UAV Control System (CUCS) and a Vehicle Specific Module (VSM). The CUCS provides common control functions, while the VSM serves as a platform-specific interface that translates vehicle-dependent data and commands into standardized representations. This separation enables heterogeneous unmanned platforms to be operated through a common control environment. STANAG 4586 also defines Levels of Interoperability (LOIs), which specify the extent of information access and control capabilities available to a control system. Previous studies have applied STANAG 4586 to UAV navigation and control systems [17] and investigated interoperability between STANAG 4586 and MAVLink [18]. However, STANAG 4586 was primarily developed for UAV operations, and its message definitions and operational requirements do not directly address the characteristics of ground-based UGV systems.

Based on these interoperability principles, SHICF adapts the functional separation between common control functions and platform-specific interfaces to heterogeneous UGV environments. In the proposed architecture, the web-based GCS performs a role conceptually similar to the CUCS, while the platform-specific communication modules on each UGV provide functions analogous to those of the VSM by converting ROS-dependent data into a common binary message format. The LOI concept is adapted to distinguish interoperability functions associated with vehicle status monitoring, control command exchange, and sensor data utilization. The LOI field in the SHICF common header identifies the intended interoperability level of a message and supports the classification of communication functions. Unlike STANAG 4586, SHICF employs independently defined Message IDs and binary message structures tailored to UGV operation. Therefore, SHICF should be regarded as a framework inspired by selected STANAG 4586 interoperability concepts rather than a fully compliant implementation of the standard.

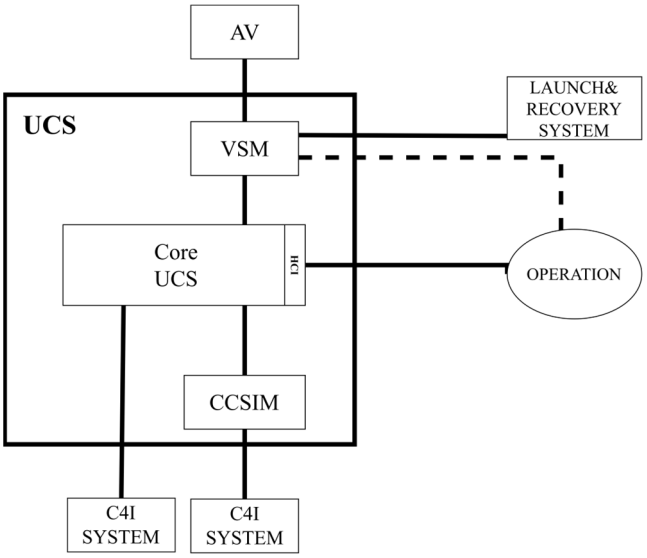


Figure 3. Functional architecture of the UCS in STANAG 4586, showing centralized supervision through the GCS and standardized interfaces that decouple platform-specific air vehicle implementations from higher-level monitoring and control functions. The dashed line indicates real-time interfaces through the Vehicle Specific Module (VSM) for time-critical operations.

3. SHICF Framework Design

3.1. Overall Framework Architecture

Figure 4 illustrates the overall architecture of SHICF, which enables the integrated monitoring and control of heterogeneous UGVs through a single web-based GCS [19]. The framework consists of four functional layers: the Heterogeneous ROS Multi-UGV Layer, Common Message Translation Layer, Windows-based GCS Server Layer, and Web-based GCS Application Layer. These layers separate platform-specific data acquisition, message conversion, communication processing, and monitoring and control functions.

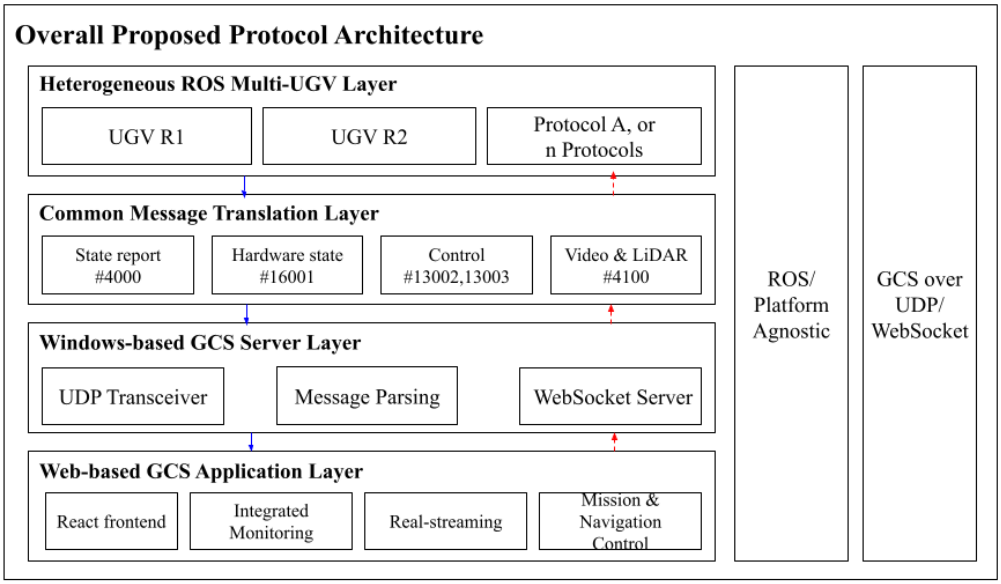


Figure 4. Proposed system architecture for integrating heterogeneous ROS-based multi-UGVs with a web-based GCS. Blue arrows indicate data transmission from UGVs to the GCS, and red arrows indicate control commands from the GCS to UGVs.

The Heterogeneous ROS Multi-UGV Layer includes UGV R1 running ROS Melodic and UGV R2 running ROS 2 Humble, while the additional protocol block represents extensibility to other robotic platforms. The Common Message Translation Layer converts platform-dependent data into a unified binary message format. Message #4000 transmits vehicle status information, Message #16001 reports hardware conditions, Messages #13002 and #13003 provide autonomous driving and emergency stop commands, and Message #4100 carries sensor stream metadata. Large-volume camera data are transmitted through a separate UDP-based binary streaming path. The Windows-based GCS Server Layer, implemented using Node.js, consists of a UDP Transceiver, Message Parsing component, and WebSocket Server. These components handle binary message transmission and reception, identify message types and vehicle IDs, and exchange data with the web application. The React-based GCS Application Layer provides integrated UGV monitoring, sensor stream visualization, and mission and navigation control.

The blue arrows indicate status and sensor data transmission from the UGVs to the GCS, whereas the red arrows represent control commands transmitted from the GCS to the UGVs. The ROS/Platform Agnostic block indicates that heterogeneous platforms can be integrated through platform-specific interfaces without modifying the common message structure. The GCS over UDP/WebSocket block represents the communication architecture, in which UDP supports binary message exchange between the UGVs and the GCS server, while WebSocket enables real-time communication between the Node.js server and React application. This layered architecture supports consistent data exchange and scalable integration of heterogeneous UGVs.

3.2. Overall Packet Structure

Figure 5 shows the common message packet structure of SHICF, which consists of four fields: Length, Header, Payload, and Checksum. The Length field indicates the total message size, while the 10-byte Header contains common information for message identification and UGV management. The Payload has a variable length of 0–57 bytes depending on the message type, allowing different types of status and control data to be transmitted. The 2-byte Checksum is used to detect transmission errors. The receiver recalculates the checksum from the received data and compares it with the transmitted value to verify message integrity [6,7,19].

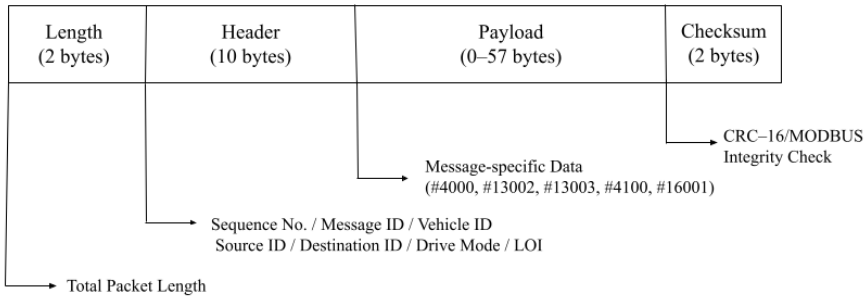


Figure 5. Overall packet structure of the SHICF.

3.3. Common Header Structure

Figure 6 illustrates the 10-byte common header of SHICF, which contains the fields required for message identification and integrated operation of heterogeneous UGVs [6,7,19]. The Sequence Number indicates the transmission order and supports the detection of missing messages, while the Message ID identifies the message type. The Vehicle ID distinguishes individual UGVs, and the Source ID and Destination ID identify the sender and receiver, respectively. The Drive Mode indicates the operating mode of the UGV, while the

LOI field specifies the interoperability level between the UGV and the GCS. Based on the interoperability concept of STANAG 4586, SHICF defines three LOI levels tailored to UGV operations. LOI 0 (Status Reporting) supports basic vehicle monitoring and state reporting. LOI 1 (Autonomous Navigation) represents waypoint-based navigation and command execution. LOI 2 (Video and Sensor Data Sharing) covers real-time video streaming and high-precision sensor data transmission. These levels distinguish the monitoring, control, and sensor data sharing functions supported by the proposed framework. Unlike the original LOI definitions in STANAG 4586, the three levels are independently defined for SHICF and reflect the operational requirements of heterogeneous UGVs. By combining these interoperability levels with a common binary message structure, SHICF provides a consistent communication interface for monitoring and controlling heterogeneous UGVs through a single GCS. All SHICF message fields are serialized in little-endian byte order using fixed-width data types. The fields are packed sequentially without implicit padding or alignment bytes. Therefore, the total message size is determined by the sum of the common header, payload, and CRC16 checksum fields.

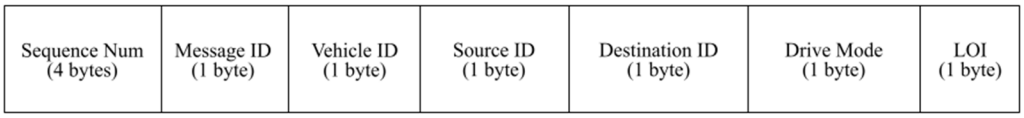


Figure 6. Structure of the common header.

3.4. Message and Payload Definitions

SHICF defines function-specific Message IDs for vehicle status reporting, subsystem monitoring, control commands, and sensor stream management. Status information and sensor metadata are transmitted from the UGVs to the GCS, while control commands are transmitted from the GCS to the UGVs. All messages share a common header containing essential information, such as the Message ID, Vehicle ID, and Sequence Number, enabling consistent message identification, vehicle distinction, and transmission sequence tracking. The payload structure varies according to the Message ID and contains only the data required for its corresponding function. Specifically, Messages #4000 and #16001 provide vehicle and subsystem status information, Messages #13002 and #13003 support autonomous driving and emergency stop control, and Message #4100 carries sensor stream metadata separately from large-volume sensor data. This message-based design enables the GCS to identify and process data from heterogeneous UGVs using a unified packet structure, while allowing additional message types and robotic platforms to be incorporated without modifying the existing common message format [6–8,19].

3.4.1. Payload Structure of Message #4000

Table 2 presents the State Report Message #4000, which periodically transmits key status information from the UGV to the GCS. The payload contains essential operational data, including geographic position (latitude, longitude, and altitude), vehicle velocity, three-axis acceleration measurements from the inertial measurement unit (IMU), and battery status. These data enable the GCS to continuously monitor the UGV's location, motion, and operational condition in real time. The message contains the Vehicle ID for UGV identification, position data including latitude, longitude, and altitude, speed, IMU data, and battery status. Each UGV converts its platform-specific status data into the same #4000 message format before transmission. This allows the GCS to consistently monitor heterogeneous UGVs regardless of their ROS versions or internal data structures [19].

Table 2. Payload structure of message #4000.

Field	Size (Bytes)	Data Type	Description
Latitude	8	float64	Current latitude in decimal degrees
Longitude	8	float64	Current longitude in decimal degrees
Altitude	4	float32	Current altitude in meters
Speed	4	float32	Current speed in meters per second
Acceleration X	4	float32	Linear acceleration along the X-axis
Acceleration Y	4	float32	Linear acceleration along the Y-axis
Acceleration Z	4	float32	Linear acceleration along the Z-axis
Quaternion X	4	float32	X component of the orientation quaternion
Quaternion Y	4	float32	Y component of the orientation quaternion
Quaternion Z	4	float32	Z component of the orientation quaternion
Quaternion W	4	float32	W component of the orientation quaternion
Battery Level	1	unit8	Remaining battery level in percent
Timestamp	4	uint32	Indicates the message generation time
Total	57	—	—

3.4.2. Payload Structure of Message #13002 (Autonomous Command)

Table 3 presents the Autonomous Command Message #13002, which transmits autonomous driving commands from the GCS to the UGV. The payload contains target geographic coordinates (latitude and longitude), target driving speed, and an autonomous driving control flag indicating whether autonomous operation is enabled or disabled. These parameters allow the GCS to specify the destination and desired speed of the UGV while controlling the activation and deactivation of autonomous driving functions. Target Latitude and Target Longitude specify the destination coordinates, while Target Speed defines the commanded travel speed. The Autonomous Command field controls the activation of autonomous driving, where a value of 1 enables the command and a value of 0 disables it [19].

Table 3. Payload structure of message #13002.

Field	Size (Bytes)	Data Type	Description
Target Latitude	8	float64	Target latitude in decimal degrees
Target Longitude	8	float64	Target longitude in decimal degrees
Target Altitude	4	float32	Target altitude in meters
Commanded Speed	4	float32	Commanded driving speed in m/s
Autonomous Command Flag	1	uint8	Autonomous navigation enable/disable (1: Enable, 0: Disable)
Total	25	-	-

3.4.3. Payload Structure of Message #13003 (Emergency Command)

Table 4 presents the Emergency Stop Message #13003, which transmits emergency stop commands from the GCS to the UGV. The Emergency Stop field controls the movement of the UGV, where a value of 1 activates the emergency stop, and a value of 0 deactivates it. Message #13003 is defined separately from autonomous driving commands, allowing the GCS to independently process emergency stop control [19].

Table 4. Payload structure of message #13003.

Field	Size (Bytes)	Data Type	Description
Emergency Stop	1	uint8	Indicates whether emergency stop is activated
Total	1	-	-

3.4.4. Payload Structure of Message #4100

Table 5 presents the Sensor Streaming Message #4100, which is designed to transmit sensor stream metadata from the UGV to the GCS independently of the corresponding sensor data. The payload contains the Frame ID and Timestamp to identify individual sensor frames and their acquisition times, the Sensor ID, Sensor Type, and Sensor Role to distinguish sensor sources and their functions, and the Frame Size to indicate the size of the corresponding sensor data. Rather than embedding large-volume sensor data, such as camera images and LiDAR measurements, directly into the message payload, SHICF adopts a metadata-based approach that separates sensor stream information from actual sensor data transmission. In the implemented architecture, camera data are transmitted through a separate UDP-based binary streaming path, while Message #4100 carries only the metadata required for sensor identification, frame tracking, and stream management. This separation allows the GCS to manage heterogeneous sensor streams using a common message structure without incorporating large-volume sensor data into the control and status messaging protocol [19].

Table 5. Payload structure of message #4100.

Field	Size (Bytes)	Data Type	Description
Frame ID	4	uint32	Frame identifier
Timestamp	8	uint64	Frame creation time
Sensor ID	1	uint8	Sensor identifier
Sensor Type	1	uint8	Sensor type
Sensor Role	1	uint8	Sensor function
Reserved	1	uint8	For future use
Frame Size	4	uint32	Frame size in bytes
Total	20	—	—

3.4.5. Payload Structure of Message #16001

Table 6 presents the Subsystem Status Message #16001, which transmits hardware and subsystem status information from the UGV to the GCS. The payload includes a Presence Vector that indicates the validity of individual status fields, a Subsystem ID that identifies the monitored subsystem, and a Subsystem State that represents its current operating condition. The Component and Subcomponent fields identify specific hardware components, while CPU, RAM, and GPU Usage indicate resource utilization, and Temperature represents the system temperature. These measurements are encoded as integer values scaled by a factor of 10 to preserve one decimal place and are restored to their original values at the GCS. By identifying valid status fields through the Presence Vector, the GCS can exclude invalid measurements from processing and consistently monitor hardware conditions across heterogeneous UGV platforms [19].

Table 6. Payload structure of message #16001.

Field	Size (Bytes)	Data Type	Description
Presence Vector	2	uint16	Indicates a valid field
Subsystem ID	1	uint8	Identifies the subsystem
Subsystem State	1	uint8	Indicates operational status
Report Reference	4	uint32	References for a related report
Controlled station	2	uint16	Identify the controlled station
Component Number	2	uint16	Identifies the component
Subcomponent Number	2	uint16	Identifies the subcomponent
CPU Usage	2	uint16	CPU utilization
RAM Usage	2	uint16	RAM utilization
GPU Usage	2	uint16	GPU utilization
Temperature	2	int16	Temperature
Total	22	—	—

3.4.6. Checksum (CRC-16-Based Data Integrity Verification)

Table 7 presents the CRC-16/MODBUS-based data integrity verification method used in SHICF to detect packet corruption during transmission. The CRC is calculated over the Common Header and Payload, excluding the CRC-16 field [20], and the resulting 2-byte value is stored at the end of the packet in little-endian format. Upon reception, the GCS recalculates the CRC using the same method and compares it with the received value. Packets with matching CRC values are accepted as valid, while packets with mismatched values are discarded. This mechanism enables SHICF to verify data integrity without an additional retransmission process [19].

Table 7. Checksum Field Structure.

Field	Size (Bytes)	Data Type	Description
Checksum	2	uint16	Detects packet transmission errors

3.5. Bidirectional Framework Communication Flow

SHICF consists of three functional layers: the UGV layer, GCS server layer, and React-based GCS layer. The UGV layer collects ROS topic data and executes control commands. The GCS server processes UDP packets, manages UGV status data, and relays data through WebSocket. The React-based GCS visualizes UGV status information and provides an interface for control commands. For communication from the UGV to the GCS, status data such as position, speed, battery level, and IMU data are collected from ROS topics and converted into common binary messages. The messages are transmitted to the GCS server via UDP, where they are processed based on the Message ID and Vehicle ID. The processed data are then transmitted through WebSocket and visualized in real time on the React-based GCS. For communication from the GCS to the UGV, control commands entered through the React-based GCS are transmitted to the GCS server via WebSocket and converted into the corresponding binary messages. The messages are then transmitted to the target UGV via UDP. Message #13002 provides autonomous driving commands, while Message #13003 provides emergency stop commands.

Figure 7 illustrates the communication and data processing flow from the UGVs to the GCS. Each UGV encapsulates the collected data into binary messages according to

predefined message specifications and transmits them to the GCS server via UDP. The GCS server verifies the CRC of each received message and identifies the message type and source UGV based on the Message ID and Vehicle ID. Validated messages are then processed according to their respective message specifications, and the resulting data are forwarded to the React-based GCS frontend via WebSocket [20]. Message #4000 transmits essential UGV operational information, including position, driving status, and battery status. The GCS server parses the received binary data according to the predefined message specification and converts the parsed data into JSON-formatted Robot Status Data. This information is updated whenever a new message is received and is displayed on the map and vehicle status interface in the Dashboard & Map View. Message #16001 transmits hardware status information, including CPU, RAM, and GPU utilization, as well as temperature information. The GCS server parses the received message and forwards the processed information to the frontend via WebSocket, enabling real-time hardware status monitoring on the Robot Detail Page. Message #4100 transmits sensor stream metadata associated with image and LiDAR data, including the Frame ID, Timestamp, Sensor ID, Sensor Type, Sensor Role, and Frame Size. The actual sensor data are processed separately from the Message #4100 payload, while camera images are encapsulated in a separate binary structure and transmitted via UDP. The GCS server processes the received metadata through a dedicated streaming processing path and forwards the relevant information to the frontend via WebSocket to support sensor stream management and real-time visualization in Operation View [21]. In addition, the GCS server generates and stores communication logs based on the received messages, allowing operators to review UGV-specific status changes and communication histories. Through this process, SHICF integrates data generated by UGVs operating in different ROS environments into a unified message structure and processes the data according to their Message IDs. This enables consistent monitoring of the operational status, sensor information, and hardware status of heterogeneous UGVs through a single web-based GCS.

Figure 8 illustrates the communication and control flow from the GCS to the UGVs. Control commands generated through the React-based GCS frontend are transmitted to the GCS server via WebSocket communication and subsequently forwarded to the designated UGV through UDP. When an operator selects a destination on the Google Maps interface, the GCS generates an autonomous driving command based on the selected target coordinates. Message #13002 transmits autonomous driving commands, including the target latitude, longitude, driving speed, and autonomous operation control flag, while Message #13003 provides emergency stop commands. Each command is encapsulated into a predefined binary message structure, and a CRC-16 checksum calculated using the algorithm described in the referenced study is included to detect transmission errors. The GCS server identifies the command type and target UGV based on the Message ID and Vehicle ID, records the command information in the Command Log, and transmits the message to the corresponding UGV through Robot IP Routing [22,23].

Upon receiving the command, the UGV verifies the CRC-16 checksum and interprets the payload according to its Message ID. The received command is then converted into the corresponding robot control operation, enabling autonomous navigation or emergency stop execution. Through this process, SHICF supports consistent and independent control of heterogeneous UGVs using a unified binary message structure, regardless of their underlying ROS environments.

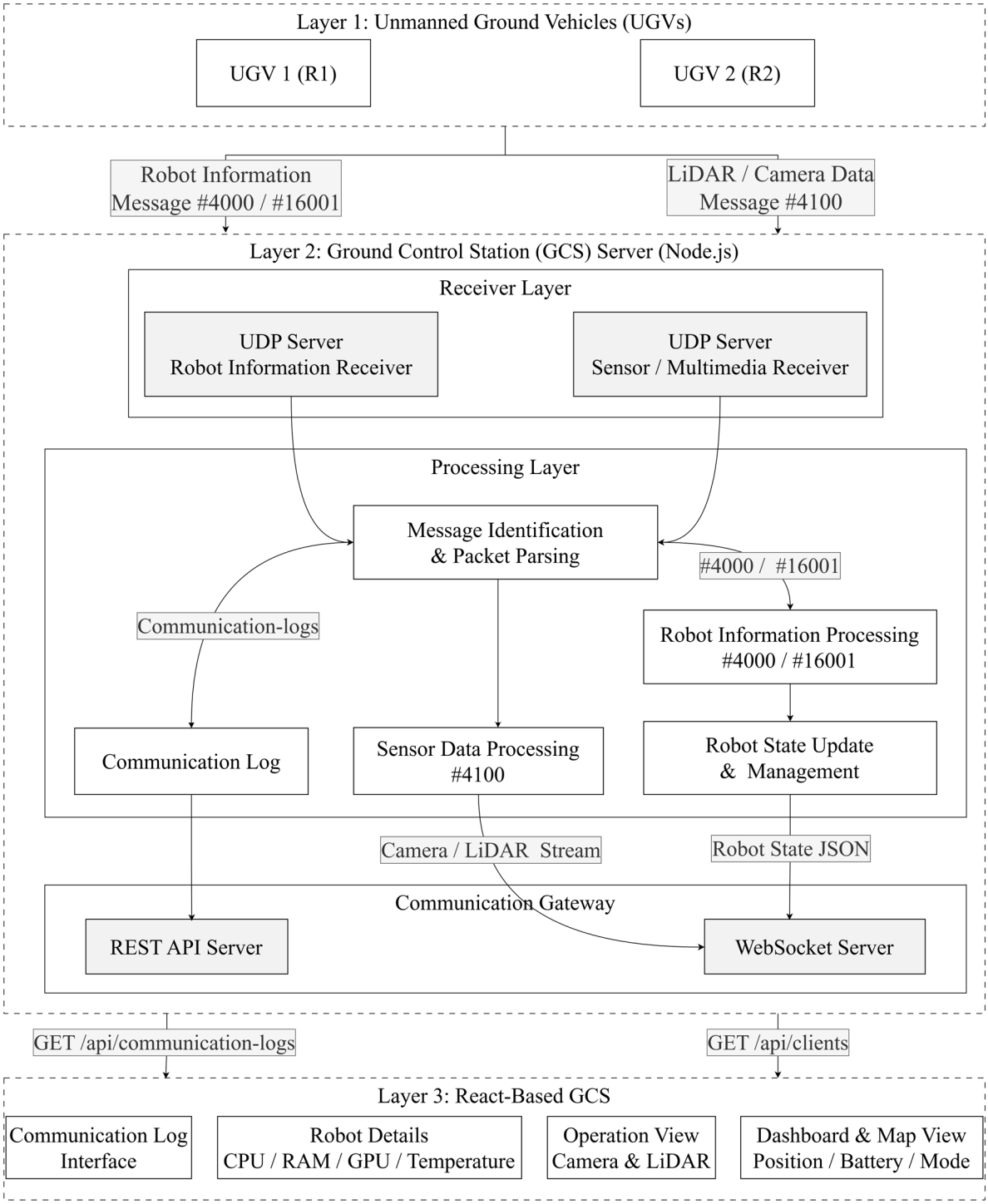


Figure 7. UGV-to-GCS Communication Framework Flow. The dashed boxes distinguish the UGVs, GCS server, and GCS layers, while the arrows indicate the directions of information exchange between these layers.

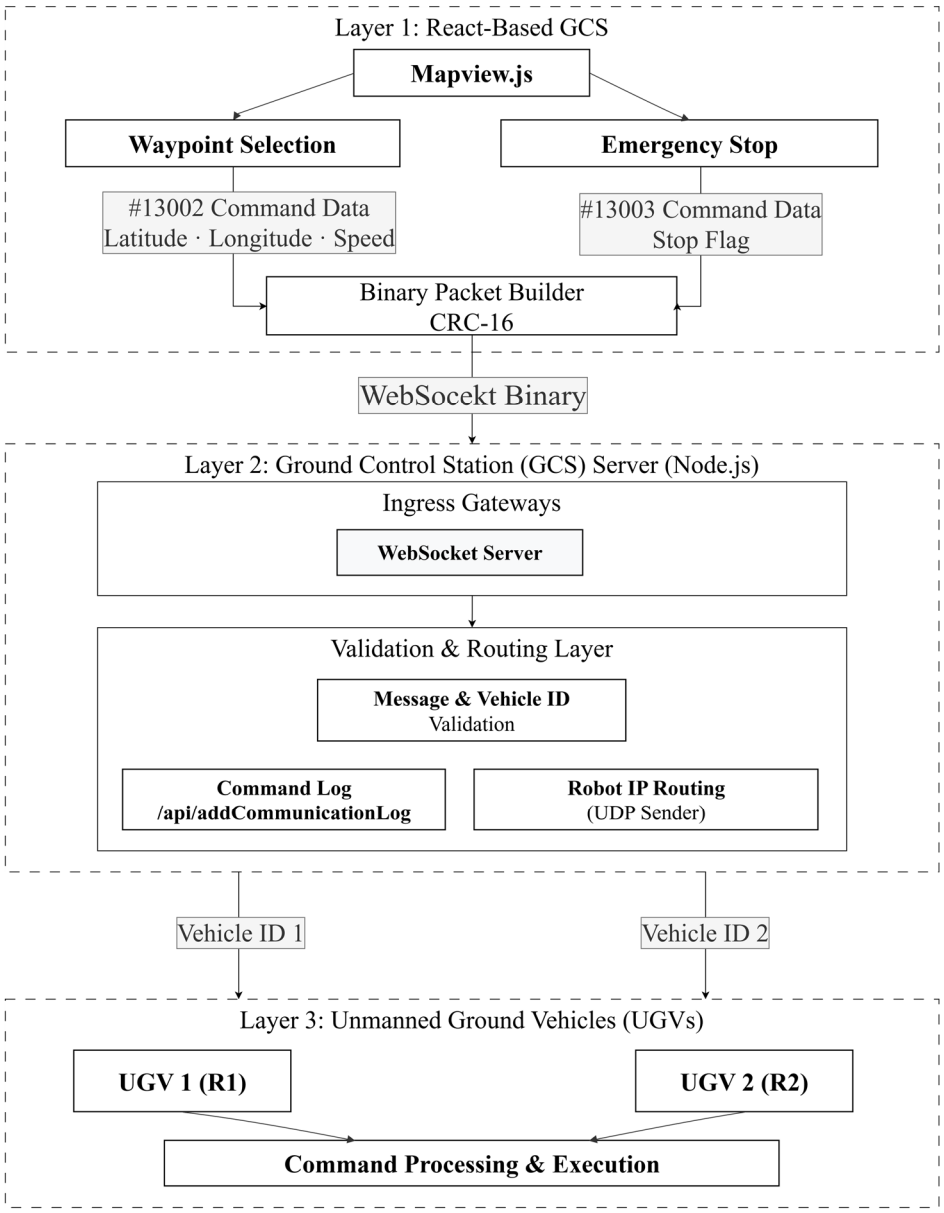


Figure 8. GCS-to-UGV communication Framework flow. The dashed boxes distinguish the GCS, GCS server, and UGVs layers, while the arrows indicate the directions of information exchange between these layers.

4. Experimental Environment and Setup

4.1. Experimental Setup

Table 8 presents the experimental environment used to evaluate the communication performance of SHICF across different ROS versions and hardware platforms. The ROS Melodic-based UGV (R1) was equipped with a Jetson Nano, while the ROS 2 Humble-based UGV (R2) used a Jetson Orin Nano Super. Both UGVs transmit position, speed, battery, and IMU data to the same GCS. The Native ROS Topic and SHICF communication configurations were evaluated under the same wireless network conditions using identical status data, transmission intervals, and measurement durations to ensure a consistent performance comparison. Both UGVs operated over an IEEE 802.11n wireless network at 2.4 GHz, with theoretical link rates of 150 Mbps for R1 and 135 Mbps for R2. The measured network throughputs were 15.6 Mbps and 21.0 Mbps, respectively. The message transmission rate was set to 10 Hz, and each experiment was conducted for approximately 20 min.

Bandwidth utilization was calculated by dividing the sum of the average transmission (TX) and reception (RX) data rates by the theoretical link rate and expressing the result as a percentage. In R1, the bandwidth utilization was 1.65% for the Native configuration and 1.53% for SHICF, whereas in R2, the corresponding values were 0.37% and 0.24%, respectively. These results indicate that SHICF exhibited lower bandwidth utilization than the corresponding Native configuration in both ROS environments. The bandwidth utilization values represent the measured bidirectional network traffic normalized by the theoretical link rate rather than the actual wireless channel occupancy.

Table 8. Experimental Environment setup.

Parameter	R1 (Native/SHICF)		R2 (Native/SHICF)	
Network type	IEEE 802.11n (2.4 GHz)			
Theoretical link rate	150 Mbps		135 Mbps	
Measured throughput	15.6 Mbps		21.0 Mbps	
Number of UGVs	2			
Message rate	10 Hz			
Bandwidth utilization	1.65%	1.53%	0.37%	0.24%
Test duration	Approximately 20 min			
OS	Ubuntu 18.04		Ubuntu 22.04	

4.2. Experimental Configuration

Table 9 summarizes the experimental setup used to evaluate SHICF in heterogeneous ROS environments. Two UGVs were employed: R1, equipped with a Jetson Nano running ROS Melodic, and R2, equipped with a Jetson Orin Nano Super running ROS 2 Humble. The experiments considered three communication configurations: native ROS Melodic communication, native ROS 2 Humble communication, and the proposed SHICF communication framework. For the native configurations, each UGV operated a rosbridge server to transmit ROS topic data directly to the web client through WebSocket communication [24,25].

Table 9. Experimental Environment Configuration.

Parameter	ROS Native Configuration	ROS 2 Native Configuration	SHICF
ROS Version	ROS melodic	ROS 2 humble	ROS/ROS 2
Communication method	TCPROS	DDS/RTPS (Best Effort QoS)	Custom Binary Protocol over UDP
Data source	ROS topics		
Serialization	ROS message serialization	CDR Serialization	Custom binary serialization
Transport layer	TCP	DDS/RTPS over UDP	UDP
Packet structure	ROS message format	DDS/RTPS Message format	Custom packet format
Error detection	Not explicitly applied		Checksum
Sequence control	Not explicitly applied		Sequence number

In contrast, SHICF converted ROS topic data into unified binary messages and transmitted them to the GCS server via UDP, where the received data were processed and forwarded to the web application through WebSocket communication. The performance evaluation consisted of two platform-specific comparisons: native ROS Melodic versus

SHICF implemented on ROS, and native ROS 2 Humble versus SHICF implemented on ROS 2. ROS Melodic uses TCPROS for native topic communication, whereas ROS 2 Humble supports UDP-based communication through DDS/RTPS. Considering these differences, the ROS 2 Humble environment was selected as the primary platform for quantitative performance evaluation, while the integrated operation of ROS and ROS 2 UGVs under a single SHICF-based GCS was used to verify interoperability across heterogeneous ROS environments. To ensure a fair performance comparison, the native and SHICF configurations were evaluated using identical data, transmission intervals, network conditions, and measurement durations, with each configuration tested independently. The performance metrics included end-to-end latency, jitter, packet loss rate, application-layer throughput, total network usage, CPU utilization, and memory utilization. These measurements were used to evaluate the communication performance and resource utilization of SHICF relative to the corresponding native ROS configurations.

4.3. Experimental Results of GCS Monitoring

To verify the operation of SHICF, the web-based GCS was evaluated for UGV status monitoring, vehicle control, and system status monitoring. The GCS processes the received data separately for each UGV and provides three main interfaces: the Total Page, Operation Page, and Robot Detail Page. These interfaces allow users to monitor multiple UGVs, control individual UGVs, and view detailed system status information in real time.

Figure 9 shows the Total Page of the GCS, which displays the status information of multiple UGVs received through SHICF. Each UGV is identified by its Vehicle ID, allowing the GCS to display the connection status and key operational information of each vehicle. The received position data are displayed on the map to monitor the current location and movement of each UGV in real time. The results confirm that status data from UGVs using different ROS environments can be processed in a common message format and monitored through a single GCS. The user can select a target UGV based on its Vehicle ID and transmit control commands from the GCS to the selected UGV. Message #13002 transmits the target position and speed for autonomous driving, while Message #13003 transmits the emergency stop command. The Operation Page also displays status information received from the UGV, allowing users to monitor vehicle status and control individual UGVs through the same interface.

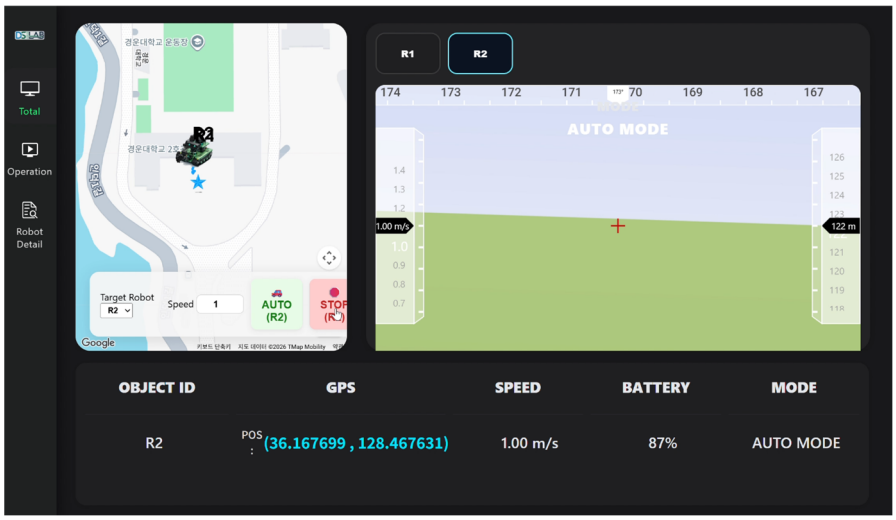


Figure 9. Experimental results of GCS (Total page screen).

Figure 10 shows the Operation Page of the GCS, which provides real-time monitoring and visualization functions for individual UGVs. The left side displays real-time camera

images of the area in front of the vehicle. The right side visualizes 2D LiDAR sensor data as points representing surrounding obstacles, enabling real-time observation of obstacle locations. By presenting camera images and LiDAR data within an interface, the Operation Page allows the operator to simultaneously monitor the forward view and the spatial distribution of obstacles. This integrated visualization enhances situational awareness and demonstrates GCS's capability for real-time sensor monitoring through a unified interface.

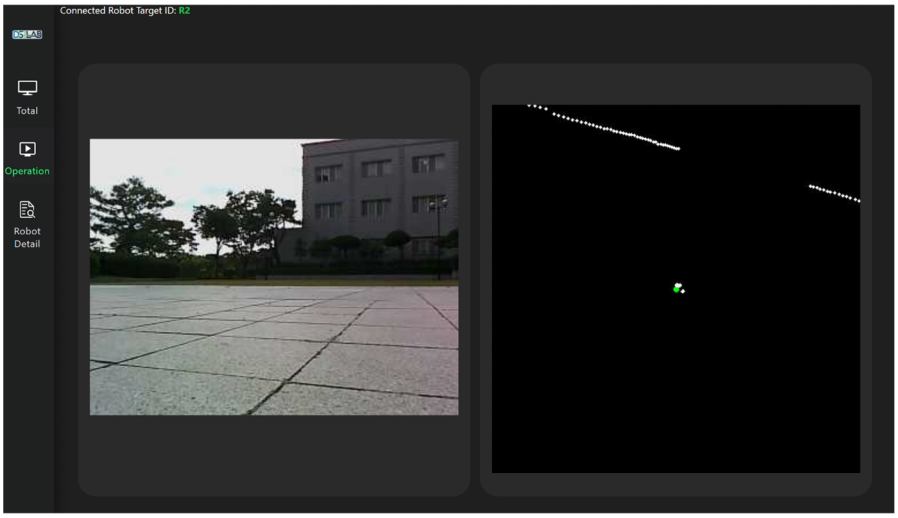


Figure 10. Experimental results of GCS (Operation screen). The left panel displays the real-time camera view, while the dashed lines in the right panel indicate obstacles, and the central icon represents the robot's current position.

Figure 11 shows the Robot Detail Page of the GCS, which provides detailed communication and connection information for each UGV. The Connected Robot Report displays the connection and communication status of each UGV based on its Vehicle ID. The Robot Communication Logs record exchanged messages by timestamp, transmission direction, Message ID, and message type. During the experiment, Message #4000 transmitted from the UGV to the GCS and Messages #13002 and #13003 transmitted from the GCS to the UGV were successfully recorded. Connection and disconnection events were also recorded, allowing the GCS to monitor the communication status and message exchange history of each UGV.

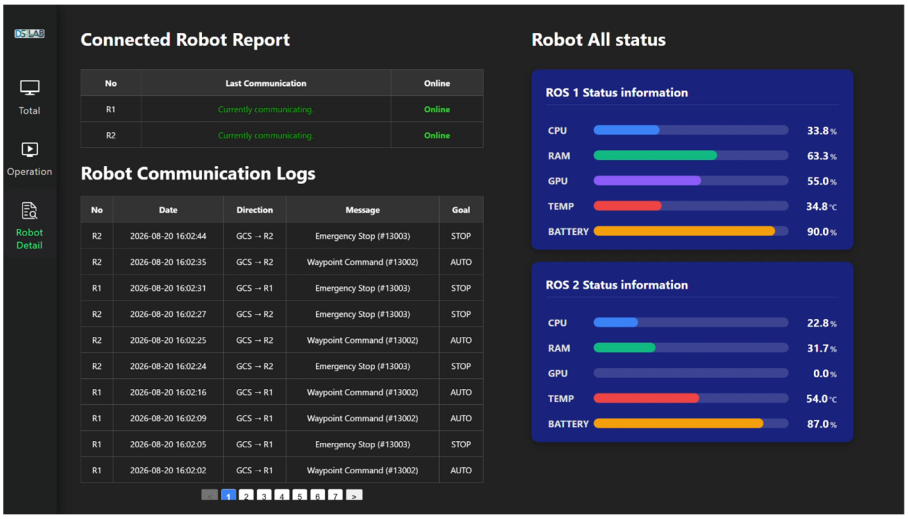


Figure 11. Experimental results of GCS (Robot Detail screen).

4.4. Experimental Scenarios

This experiment compares the communication performance of the Native ROS Topic method and SHICF for transmitting UGV status data to the React-based GCS. Both methods were evaluated using the same status data, transmission intervals, network conditions, and measurement duration. The experiments were conducted under the same conditions on ROS Melodic- and ROS 2 Humble-based UGVs to evaluate performance across different ROS versions. In the Native ROS Topic scenario, position, speed, battery, and IMU data were published as ROS topics and transmitted through rosbridge_server. The React-based GCS connected directly to the rosbridge_server of each UGV through WebSocket to receive and visualize the status data. The Node.js GCS server and SHICF binary message conversion were not used in this scenario, and the Native ROS Topic structure was used as the reference method. In the SHICF scenario, position, speed, battery, and IMU data collected from ROS topics were integrated into a common binary status message and transmitted to the GCS server via UDP. The server processed the received messages based on the Message ID and Vehicle ID and forwarded the status data to the React-based GCS through WebSocket. This scenario evaluates the performance of SHICF without requiring the GCS to directly process ROS-version-specific topic structures.

5. Communication Evaluation and Results

To evaluate the communication performance of SHICF, comparative experiments were conducted in two ROS environments: ROS Melodic and ROS 2 Humble. In the ROS Melodic environment, the Native TCPROS-based communication configuration was compared with the proposed UDP-based SHICF, while in the ROS 2 Humble environment, the Native communication configuration using Best Effort QoS was compared with SHICF. Each experimental configuration was evaluated over a period of 1200 s using five common performance metrics to quantitatively assess communication latency, stability, and network efficiency [26]. The experimental results were analyzed independently within each ROS environment, focusing on comparisons between the Native and SHICF communication configurations rather than direct performance comparisons between ROS Melodic and ROS 2 Humble. Specifically, the ROS Melodic-based Native communication configuration was compared with SHICF on a Jetson Nano, while the ROS 2 Humble-based Native communication configuration was compared with SHICF on a Jetson Orin Nano Super. Each comparison was conducted under identical hardware conditions within the respective ROS environment.

5.1. Performance Metrics—Latency

Figure 12 presents the round-trip time (RTT) and jitter measurements obtained using the Native and SHICF communication configurations in ROS Melodic and ROS 2 Humble environments. Figures 12a and 12b present the RTT and jitter measurements for the Native and SHICF configurations in ROS Melodic, respectively. The Native configuration recorded an average RTT of 13.22 ms and jitter of 4.88 ms, whereas SHICF achieved 7.95 ms and 4.03 ms, representing reductions of approximately 39.9% and 17.5%, respectively. These results indicate lower communication delay and reduced delay variation under the evaluated conditions. Figure 12c,d show the corresponding results for ROS 2 Humble. In Figure 12c, the Native configuration with Best Effort QoS recorded an average RTT of 29.98 ms and jitter of 19.81 ms, while Figure 12d presents the SHICF measurements for comparison. However, these differences should be interpreted considering the distinct communication architectures rather than attributed solely to the proposed message format. Specifically, the Native configuration employs ROS topic communication through rosbridge, JSON serialization, and WebSocket transport, whereas SHICF uses a custom binary message

format transmitted over UDP to the GCS backend. Consequently, differences in RTT and jitter may reflect the combined effects of serialization overhead, transport protocols, middleware processing, and communication-path implementation. Furthermore, because experiments were conducted over a Wi-Fi-based wireless network, variations in signal conditions, network contention, and packet scheduling may also have influenced the measurements. Therefore, the results demonstrate the communication performance of the evaluated end-to-end configurations rather than isolating the performance contribution of any individual protocol component.

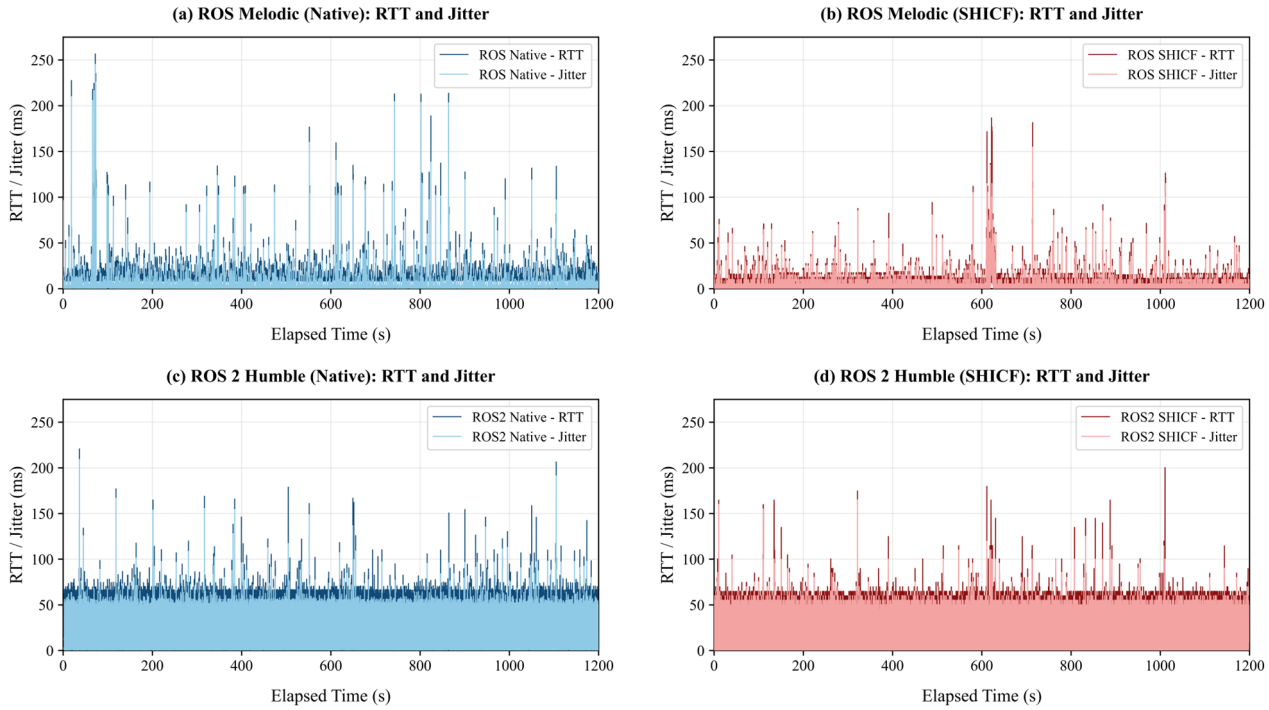


Figure 12. RTT and jitter measurements for the Native and SHICF communication configurations: (**a**) ROS Melodic Native; (**b**) ROS Melodic SHICF; (**c**) ROS 2 Humble Native; (**d**) ROS 2 Humble SHICF.

5.2. Performance Metrics—System Resource Usage

Figure 13 presents the CPU and memory utilization measurements obtained using the Native and SHICF communication configurations in ROS Melodic and ROS 2 Humble environments. Figures 13a and 13b present the CPU and memory utilization measurements for the Native and SHICF configurations in ROS Melodic, respectively. The Native configuration recorded average CPU and memory utilization of 20.23% and 59.21%, respectively, whereas SHICF recorded 20.79% and 53.21%, respectively. Figure 13c,d show the corresponding results for ROS 2 Humble. In ROS 2 Humble, average CPU utilization increased from 9.67% for the Native configuration to 11.15% for SHICF, whereas memory utilization decreased from 57.34% to 33.91%. Although SHICF requires processing for ROS data acquisition, binary serialization, checksum calculation, and UDP transmission, lower memory utilization was observed in both ROS environments. These differences may reflect the combined effects of serialization formats, transport protocols, and middleware processing rather than the proposed message structure alone. The resource utilization measurements were examined by comparing the Native and SHICF configurations independently within each ROS environment.

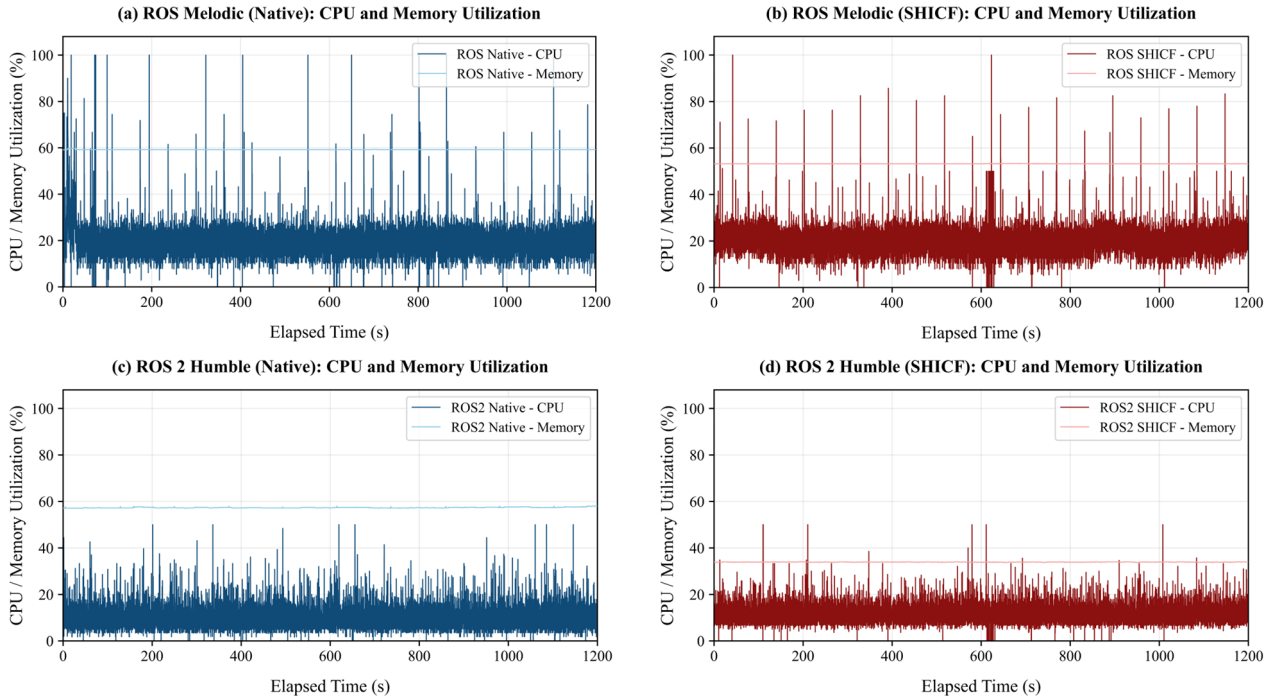


Figure 13. CPU and memory utilization for the Native and SHICF communication configurations: (**a**) ROS Melodic Native; (**b**) ROS Melodic SHICF; (**c**) ROS 2 Humble Native; (**d**) ROS 2 Humble SHICF.

5.3. Performance Metrics—Total Network Throughout

Figure 14 presents the network usage measurements obtained using the Native and SHICF communication configurations in ROS Melodic and ROS 2 Humble environments. Figures 14a and 14b present the network usage measurements for the Native and SHICF configurations in ROS Melodic, respectively. The Native configuration recorded an average network usage of 309.45 KB/s, whereas SHICF recorded 286.49 KB/s. Figure 14c,d show the corresponding results for ROS 2 Humble. In ROS 2 Humble, average network usage decreased from 62.64 KB/s for the Native configuration with Best Effort QoS to 41.20 KB/s for SHICF. These results indicate that SHICF exhibited lower network usage than the corresponding Native configuration in both ROS environments under the evaluated conditions. However, the observed differences may reflect the combined effects of binary serialization, transport protocols, and middleware processing rather than the proposed message structure alone. Furthermore, network usage was evaluated by comparing the Native and SHICF configurations independently within each ROS environment, rather than directly comparing the performance of ROS Melodic and ROS 2 Humble, which were implemented on different hardware platforms.

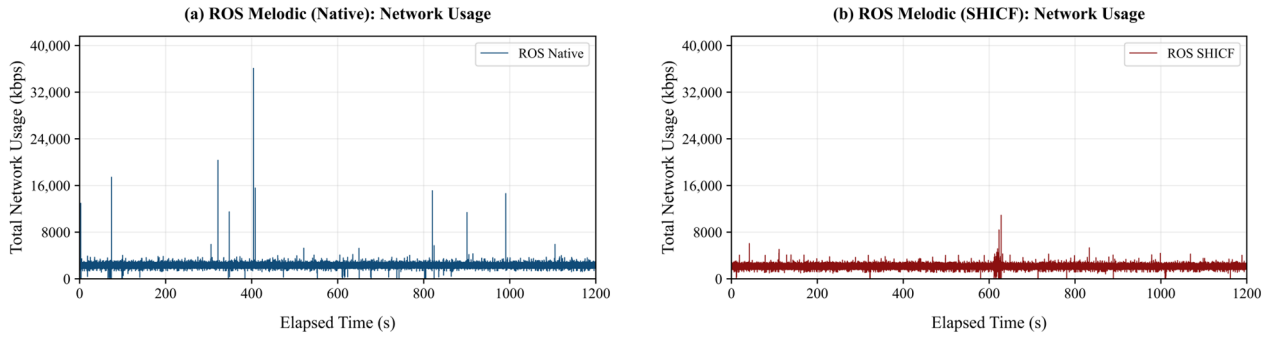


Figure 14. *Cont.*

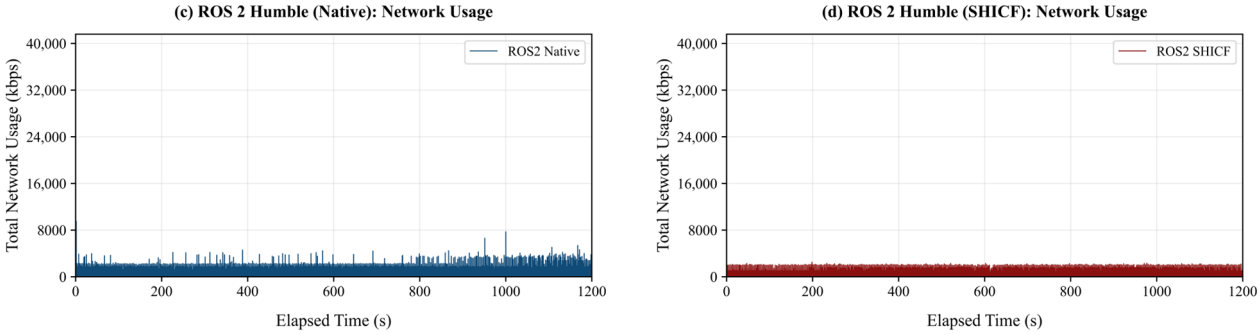


Figure 14. Network usage for the Native and SHICF communication configurations: (**a**) ROS Melodic Native; (**b**) ROS Melodic SHICF; (**c**) ROS 2 Humble Native; (**d**) ROS 2 Humble SHICF.

5.4. Performance Metrics—Total Application Throughput

Figure 15 presents the application-level data rate measurements obtained using the Native and SHICF communication configurations in ROS Melodic and ROS 2 Humble environments. Figures 15a and 15b present the application-level data rate measurements for the Native and SHICF configurations in ROS Melodic, respectively. The Native configuration recorded an average application-level data rate of 2.09 KB/s, whereas SHICF recorded 0.75 KB/s. Figure 15c,d show the corresponding results for ROS 2 Humble. In ROS 2 Humble, the average application-level data rate decreased from 2.45 KB/s for the Native configuration with Best Effort QoS to 0.83 KB/s for SHICF. These results indicate that SHICF exhibited lower application-level data rates than the corresponding Native configurations in both ROS environments under the evaluated conditions.

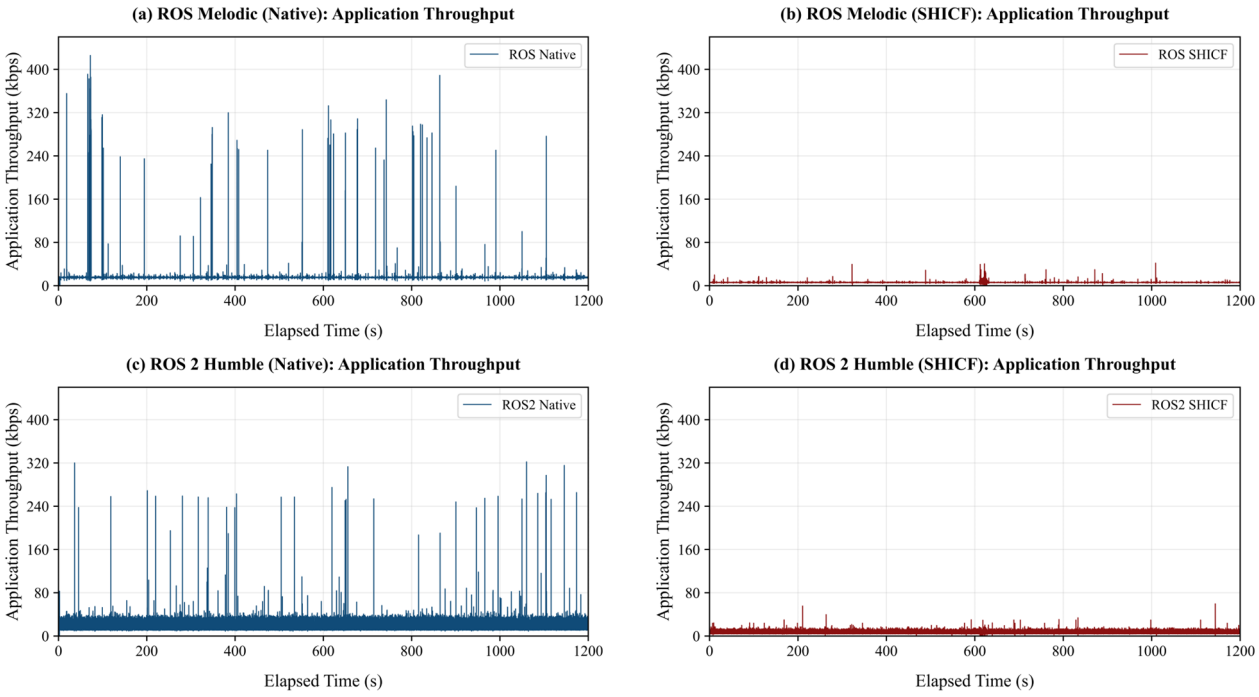


Figure 15. Application-level data rates for the Native and SHICF communication configurations: (**a**) ROS Melodic Native; (**b**) ROS Melodic SHICF; (**c**) ROS 2 Humble Native; (**d**) ROS 2 Humble SHICF.

Furthermore, the application-level data rates were evaluated by comparing the Native and SHICF configurations independently within each ROS environment, rather than directly comparing the performance of ROS Melodic and ROS 2 Humble.

5.5. Performance Metrics—Cumulative Packet Loss Rate

Figure 16 presents the application-level message loss rates measured for the Native and SHICF communication configurations in ROS Melodic and ROS 2 Humble environments. Figures 16a and 16b present the application-level message loss rates for the Native and SHICF configurations in ROS Melodic, respectively. The Native configuration recorded an average message loss rate of 0.45%, whereas SHICF recorded 0.29%. Figure 16c,d show the corresponding results for ROS 2 Humble. In ROS 2 Humble, the average message loss rate decreased from 1.42% for the Native configuration with Best Effort QoS to 0.20% for SHICF. These results were evaluated independently within each ROS environment, focusing on the message delivery characteristics of the Native and SHICF configurations rather than on direct performance comparisons between ROS Melodic and ROS 2 Humble. The evaluation was primarily intended to assess communication behavior and demonstrate the feasibility of real-time monitoring and integrated communication among heterogeneous UGVs through a single GCS. Overall, SHICF exhibited lower observed application-level message loss rates than the corresponding Native configurations in both ROS environments under the evaluated experimental conditions. The reported loss rates were calculated based on application-level acknowledgment (ACK) timeouts.

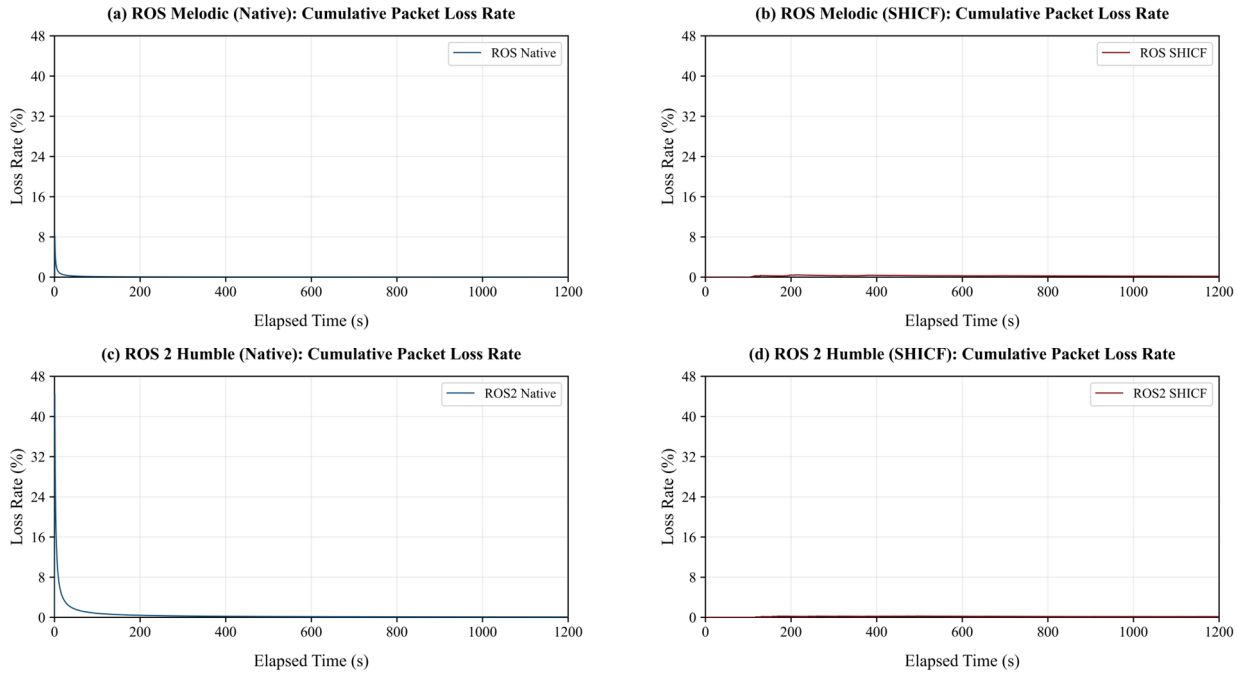


Figure 16. Application-level ACK timeout rates for the Native and SHICF communication configurations: (**a**) ROS Melodic Native; (**b**) ROS Melodic SHICF; (**c**) ROS 2 Humble Native; (**d**) ROS 2 Humble SHICF.

6. Conclusions

This study proposed the Scalable Heterogeneous Interoperable Communication Framework (SHICF) for the integrated monitoring and control of heterogeneous unmanned ground vehicles (UGVs) through a single Ground Control Station (GCS). SHICF adapts the messaging and interoperability concepts of STANAG 4586 to UGV systems by introducing a common binary message structure for exchanging vehicle status information and control commands. The performance of SHICF was evaluated against the corresponding Native communication configurations in ROS Melodic and ROS 2 Humble. The experimental results showed that SHICF achieved lower communication latency and reduced data transmission overhead under the evaluated conditions, demonstrating its feasibility

for heterogeneous UGV integration. However, these performance differences may reflect the combined effects of serialization formats, transport protocols, and middleware processing rather than the proposed binary message structure alone. Furthermore, because the experiments were conducted on different hardware platforms, performance comparisons were performed within each ROS environment rather than directly between the two platforms. Therefore, the findings demonstrate the performance characteristics of the evaluated end-to-end communication configurations while confirming the applicability of SHICF to heterogeneous robotic systems.

The proposed framework is designed to accommodate additional robotic platforms and communication protocols through platform-specific interfaces while maintaining a common message structure. However, the current evaluation was limited to two UGV platforms and the tested network conditions.

Future work will extend the experimental environment to approximately 5–10 or more heterogeneous UGVs to systematically evaluate communication performance, network traffic, resource utilization, and message delivery characteristics as the number of connected platforms increases. Additional experiments will investigate communication reliability under network congestion, packet loss, and communication failure conditions, as well as the reliability of safety-critical control commands, including emergency stop and autonomous navigation commands. Furthermore, comparative experiments involving MAVLink-based communication will be conducted to assess the performance and interoperability characteristics of SHICF in heterogeneous unmanned systems, including UGVs and UAVs. The separate sensor streaming path associated with Message #4100 will also be evaluated in terms of bandwidth consumption, packet loss characteristics, and congestion behavior during high-bandwidth camera and LiDAR data transmission.

Author Contributions: Conceptualization, S.K. and O.-k.H.; methodology, S.K. and O.-k.H.; software, S.K. and S.L.; validation, S.K. and S.L.; formal analysis, S.K.; investigation, S.K. and S.L.; resources, O.-k.H.; data curation, S.K.; writing—original draft preparation, S.K.; writing—review and editing, S.K. and O.-k.H.; visualization, S.K. and S.L.; supervision, O.-k.H.; project administration, O.-k.H.; funding acquisition, O.-k.H. All authors have read and agreed to the published version of the manuscript.

Funding: This research was supported by the Ministry of Trade, Industry and Energy (MOTIE) and the Korea Institute for Advancement of Technology (KIAT) through the "Support for Mid-sized Enterprises and Regional Innovation Alliances (R&D, RS-2024-00421022)" program.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: The raw data supporting the conclusions of this article will be made available by the authors on request.

Acknowledgments: We thank the reviewers for their helpful remarks that strengthened this work.

Conflicts of Interest: The authors declare no conflicts of interest.

References

1. Cuadros Zegarra, E.; Barrios Aranibar, D.; Cardinale, Y. IoRT-Based Middleware for Heterogeneous Multi-Robot Systems. *J. Sens. Actuator Netw.* **2024**, *13*, 87. [CrossRef]
2. Quigley, M.; Gerkey, B.; Conley, K.; Faust, J.; Foote, T.; Leibs, J.; Berger, E.; Wheeler, R.; Ng, A.Y. ROS: An Open-Source Robot Operating System. In Proceedings of the ICRA Workshop on Open-Source Software, Kobe, Japan, 12–17 May 2009.
3. Bonci, A.; Gaudeni, F.; Giannini, M.C.; Longhi, S. Robot Operating System 2 (ROS 2)-Based Frameworks for Increasing Robot Autonomy: A Survey. *Appl. Sci.* **2023**, *13*, 12796. [CrossRef]
4. Maruyama, Y.; Kato, S.; Azumi, T. Exploring the Performance of ROS2. In Proceedings of the 13th ACM SIGBED International Conference on Embedded Software (EMSOFT '16), Pittsburgh, PA, USA, 1–7 October 2016; pp. 1–10.

5. Wei, Z.; You, H.; Xu, H.; Deng, Z. The Evolution of the Robot Operating System Communication Ecosystem: An Overview of the DDS Architecture and Emerging Communication Protocols. *Electronics* **2026**, *15*, 2632. [CrossRef]
6. NATO Standardization Office. *STANAG 4586: Standard Interfaces of UA Control System (UCS) for NATO UA Interoperability*, 4th ed.; NATO Standardization Office: Brussels, Belgium, 2017.
7. NATO Standardization Office. *NATO UA Control System (UCS) Architecture and Interface Definition*; AEDP-17 (Implementation Guidance for STANAG 4586), Ed. A Ver. 1; NATO Standardization Office: Brussels, Belgium, 2017.
8. Kayayurt, B.; Yayla, I. Application of STANAG 4586 Standard for Turkish Aerospace Industries UAV Systems. In Proceedings of the 2013 IEEE/AIAA 32nd Digital Avionics Systems Conference (DASC), East Syracuse, NY, USA, 5–10 October 2013.
9. Müller, W.; Segor, F.; Mühlenberg, D.; Driessen, B.; De Geus, V.; Papp, Z.; Svenmarck, P.; Kalis, A.; Amditis, A. Interoperability of Armed Forces Unmanned Systems: The INTERACT Project. In Proceedings of the Open Architecture/Open Business Model Net-Centric Systems and Defense Transformation 2023, Orlando, FL, USA, 30 April–4 May 2023; Volume 12544.
10. Macenski, S.; Foote, T.; Gerkey, B.; Lalancette, C.; Woodall, W. Robot Operating System 2: Design, Architecture, and Uses in the Wild. *Sci. Robot.* **2022**, *7*, eabm6074. [CrossRef] [PubMed]
11. Portugal, D.; Rocha, R.P.; Castilho, J.P. Inquiring the Robot Operating System Community on the State of Adoption of the ROS 2 Robotics Middleware. *Int. J. Intell. Robot. Appl.* **2025**, *9*, 454–479. [CrossRef]
12. Ginting, M.F.; Otsu, K.; Edlund, J.A.; Gao, J.; Agha-Mohammadi, A.-A. CHORD: Distributed Data-Sharing via Hybrid ROS 1 and 2 for Multi-Robot Exploration of Large-Scale Complex Environments. *IEEE Robot. Autom. Lett.* **2021**, *6*, 5064–5071. [CrossRef]
13. Hammoudeh García, N.; Chen, Y.; Lieb, D.; Wortmann, A. Evaluation of a Model-Driven Approach for the Integration of Robot Operating System-Based Complex Robot Systems. *Int. J. Adv. Robot. Syst.* **2025**, *22*, 17298806251363648. [CrossRef]
14. Koubaa, A.; Allouch, A.; Alajlan, M.; Javed, Y.; Belghith, B.; Khalgui, M. Micro Air Vehicle Link (MAVLink) in a Nutshell: A Survey. *IEEE Access* **2019**, *7*, 87658–87680. [CrossRef]
15. Nam, G.; Go, J.; Kwon, C.; Jeong, S. A Study on Comparative Analysis and Improvement of STANAG 4586 and MAVLink Protocol for UAS Interoperability Enhancement. *J. Korea Inst. Mil. Sci. Technol.* **2020**, *23*, 618–638. [CrossRef]
16. Marques, M.M. STANAG 4586—Standard Interfaces of UAV Control System (UCS) for NATO UAV Interoperability. In *NATO STO Educational Notes, STO-EN-SCI-271*; NATO Science and Technology Organization: Neuilly-sur-Seine, France, 2015.
17. Frazzetta, S.; Pacino, M. A STANAG 4586 Oriented Approach to UAS Navigation: The Case of Italian Sky-Y Flight Trials. *J. Intell. Robot. Syst.* **2013**, *69*, 21–31.
18. Rodrigues, A.V.; Carapau, R.S.; Marques, M.M.; Lobo, V.; Coito, F. Unmanned Systems Interoperability in Military Maritime Operations: MAVLink to STANAG 4586 Bridge. In Proceedings of the OCEANS 2017—Aberdeen, Aberdeen, UK, 19–22 June 2017; pp. 1–5.
19. Kim, S.; Ha, O.-K. Design of a Lightweight State Reporting Protocol Based on Robot Operating System for Unmanned Ground Systems. In Proceedings of the 2026 6th International Conference on Electrical, Computer and Energy Technologies (ICECET), Rome, Italy, 6–9 July 2026; pp. 1–6.
20. Pantow, P.P.; Wellem, T. Perancangan dan Implementasi Encoder dan Decoder CRC-16 Berbasis Tabel Lookup pada Arduino. *IT-Explore* **2024**, *3*, 49–62. [CrossRef]
21. Ardi, N.; Razali, S.M.; Thohari, A.H. Web-Based Ground Control Station for Real-Time Monitoring Quadcopter UAVs. In Proceedings of the 8th International Conference on Applied Engineering, Batam, Indonesia, 30–31 October 2025.
22. Kavas-Torris, O.; Gelbal, S.Y.; Cantas, M.R.; Aksun-Guvenc, B.; Guvenc, L. V2X Communication between Connected and Automated Vehicles (CAVs) and Unmanned Aerial Vehicles (UAVs). *Sensors* **2022**, *22*, 8941. [CrossRef] [PubMed]
23. Rajapaksha, U.U.S.; Jayawardena, C.; MacDonald, B.A. Design, Implementation, and Performance Evaluation of a Web-Based Multiple Robot Control System. *J. Robot.* **2022**, *2022*, 9289625. [CrossRef]
24. Song, B.-Y.; Choi, H. ROS Gateway: Enhancing ROS Availability across Multiple Network Environments. *Sensors* **2024**, *24*, 6297. [CrossRef] [PubMed]
25. Open Robotics. ros1_bridge: Bridge Communication between ROS 1 and ROS 2. ros1_bridge GitHub Repository. Available online: https://github.com/ros2/ros1_bridge (accessed on 9 September 2026).
26. Tardioli, D.; Parasuraman, R.; Ögren, P. Pound: A ROS node for Reducing Delay and Jitter in Wireless Multi-Robot Networks. *Robot. Auton. Syst.* **2019**, *111*, 73–87. [CrossRef]

Disclaimer/Publisher's Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.