Co-Simulation and Runtime Verification of Micro-ROS Communication Mechanisms

Xue Rui , Zijian Wang , Li Jiang , Jian Guo *

Posted Date: 28 April 2026

doi: 10.20944/preprints202604.1883.v1

Keywords: Micro-ROS; formal method; simulation; constrained signal temporal logic; runtime verification

 doi:10.20944/preprints202604.1883.v1

Article

Co-Simulation and Runtime Verification of Micro-ROS Communication Mechanisms

Xue Rui [1], Zijian Wang [2] , Li Jiang [1] and Jian Guo [2,*]

[1] School of AI and Big Data, Xinjiang Teacher's College, Xinjiang, China
[2] MoE Engineering Research Center for Software/Hardware Co-design Technology and Application, ECNU, Shanghai, China
* Correspondence: jguo@sei.ecnu.edu.cn

Abstract

Micro-ROS is a lightweight robot operating system for embedded devices with resource-constrained real-time features. Its communication mechanism, based on the XRCE-DDS protocol, facilitates data exchange in constrained environments. To ensure the safety and reliability of message transmission in Micro-ROS, this paper presents an integrated approach that combines simulation and runtime verification. A formal model of XRCE-DDS is constructed using timed automata, while key properties derived from the protocol are expressed in constrained Signal Temporal Logic ($\mathrm{STL}_{\mathrm{lb}=0}$). The model is implemented and simulated via Simulink/Stateflow. Furthermore, the key properties are verified with the runtime verification method using the Logical and Temporal Assessments tool in the Test Manager of Simulink Test. Through integrating simulation with runtime verification, this work effectively improves the safety assurance of the Micro-ROS communication mechanisms.

Keywords: Micro-ROS; formal method; simulation; constrained signal temporal logic; runtime verification

1. Introduction

The Robot Operating System(ROS) [1] has become the most widely used middleware for the development of robotic software systems. Its safety and reliability are critical to ensuring the robustness of robotic systems. in ROS-based robotic applications, Micro-ROS serves to bridge the performance gap between resource-constrained microcontrollers and powerful computing devices [2,3]. ROS encompasses communication mechanisms, software toolkits, high-level robotic capabilities, and a comprehensive ecosystem. It is extensively applied in the research and development of autonomous robots, unmanned aerial vehicles [4], and intelligent vehicles [5,6], establishing it as a meta-operating system in robotic system development.

Micro-ROS employs XRCE-DDS (eXtremely Resource Constrained Environments DDS) as its core communication middleware in resource-constrained embedded devices. This design aims to adapt the powerful data distribution capabilities of DDS to low-power, limited-memory environments such as microcontrollers (MCUs). This design enables small devices such as microcontrollers to seamlessly integrate into the extensive DDS ecosystem without needing to run a full DDS stack, facilitating real-time and reliable data exchange with more powerful computing nodes (e.g. servers, robot main control computers, etc.).

XRCE-DDS is a client-agent-based standard that enables small, resource-constrained embedded devices to connect to the DDS global data space. The modeling, implementation and analysis of XRCE-DDS applications in distributed multiprocessor real-time embedded systems are key research focuses. Among these, XRCE-DDS utilizes an XML-based data description language to define the format and content of data exchange. Through modeling and implementation, researchers investigate approaches to guaranty real-time data transmission and processing in distributed environments.

The safety of XRCE-DDS is crucial to the Micro-ROS communication mechanism. To construct highly trustworthy Micro-ROS communication based on XRCE-DDS, formal methods are applied to model and verify data exchange in Micro-ROS.

 doi:10.20944/preprints202604.1883.v1

Tools such as temporal logic and process algebra can be used to establish precise mathematical models for core interactions in XRCE-DDS, including client-agent handshakes, message sequences, and state machines. This approach rigorously proves that the protocol is free from deadlocks and livelocks under ideal conditions, satisfies critical safety properties, and eliminates logical flaws at the design stage. Safety policies, such as agent access control lists and topic permission mappings, can be defined using formal languages to enable automated verification of policy consistency and completeness.

The reliability of the XRCE-DDS architecture design has been significantly improved through the application of formal methods. When applied to high-safety fields such as autonomous driving and medical devices, this approach will effectively enhance the safety and operational reliability of these devices.

This paper focuses on the Micro-ROS communication mechanism, employing a combined approach of simulation and formal methods to explore the safety and reliability of XRCE-DDS. Timed automata are introduced for formal modeling, while constrained signal temporal logic is used to characterize the key properties of XRCE-DDS. The model is implemented in Simulink/Stateflow, and its effectiveness is evaluated through simulations. Furthermore, runtime verification is performed using the Logical and Temporal Assessments tool in Simulink Test's Test Manager to validate critical properties.

The contributions of this paper are mainly reflected in three key aspects.

1. This study treats the Micro-ROS communication middleware, specifically Micro XRCE-DDS, as a time-critical system and builds abstract modeling. Based on the open-source code of XRCE-DDS, timed automata models are built, including parallel of the client_publisher, client_subscriber, Agent, proxy_publisher, and proxy_subscriber modules.
2. A constrained signal temporal logic, denoted as $\mathrm{STL}_{\mathrm{lb}=0}$, is proposed to describe the properties of the Micro-ROS communication mechanism. Key properties are formally specified by the constrained signal temporal logic. Based on an analysis of the XRCE-DDS protocol, critical properties are extracted from natural language descriptions and subsequently converted into logical formulas.
3. This paper proposes a co-simulation and runtime verification framework based on models for Micro-ROS. The framework is implemented in Simulink/Stateflow, where simulations are conducted and runtime verification is performed to validate critical system properties.

The remainder of this paper is organized as follows. Section 2 provides background and related work; Section 3 presents the overall design architecture; Section 4 introduces the formal modeling of Micro-ROS; Section 5 extracts key properties from the XRCE-DDS protocol and describes them using the constrained signal temporal logic $\mathrm{STL}_{\mathrm{lb}=0}$; Section 6 details the model implementation and simulation in Simulink/Stateflow, as well as runtime verification. Finally, the conclusion and future work are summarized.

2. Related Work

Micro-ROS substantially bridges the performance gap between resource-constrained microcontrollers and powerful computing devices. As a lightweight and efficient solution designed for resource-constrained systems, Micro-ROS offers broad application prospects across various domains. To address real-time performance challenges in Micro-ROS, a priority-driven chain-aware scheduling system (PoDS) [2] based on the existing Micro-ROS architecture has been proposed, along with a novel executor, TIDE [7]. Both approaches aim to enhance the temporal predictability and runtime efficiency of Micro-ROS. Furthermore, by leveraging the functionalities of both ROS 2 and Micro-ROS, an experimental platform for developing attitude control algorithms [8] has been implemented. This platform enables the evaluation of response times, message loss rates, and message periodicity of ROS 2 entities under diverse network usage scenarios.

 doi:10.20944/preprints202604.1883.v1

[9] presents a simplified integration approach for combining PXROS-HR with Micro-ROS for real-world applications. The implementation streamlines the integration process by leveraging UART communication and eliminating external dependencies. The transition from a packet-oriented to a stream-oriented custom transport enhances system consistency.

Based on assumptions regarding the ROS API and ecosystem [10], behavioral models of ROS-based systems are inferred through static analysis. This approach relies on behavioral patterns where developers utilize the ROS framework API to implement reactive, periodic, and state-dependent behaviors.

The open-source ROS platform is widely used for rapidly reconfigurable robot units, enabling the development of high-level task programming. This development method is independent of hardware and software, employing virtualized machine control interfaces [11].

[12] proposes a bare-metal implementation of the XRCE-DDS standard on the CompSOC platform based on a multi-processor system-on-Chip (MPSoC) for the Micro-ROS communication mechanism. The framework includes a hosted XRCE-DDS client and a hosted XRCE-DDS agent. To capture the dynamics of system behavior, a Scenario-Aware Data Flow (SADF) model is introduced. This model characterizes the system's behavior under various execution scenarios.

To accurately describe the latency of reliable DDS data transmission in ROS 2, [13] derives a closed-form analytical model. This model takes into account key parameters including the packet delivery ratio, data period, and heartbeat period in DDS. Guidelines are offered to optimize the data period and the heartbeat period to latency.

In the analysis of ROS 2 security policies, [14] designed and implemented a tool called LiSA4ROS 2. This tool automatically extracts the ROS 2 computation graph through static analysis. Developers use the graph to derive minimal correct configurations for ROS 2 security policies, thereby reducing the likelihood of errors.

Research on verification techniques for ROS systems has become increasingly active. Runtime verification has been applied to ROS systems to ensure that they satisfy predefined properties and specifications during execution. Runtime verification is commonly used to validate hierarchical properties in robotic clusters. With the release and adoption of ROS 2 and Micro-ROS, researchers are increasingly focusing on verifying their safety aspects, such as communication mechanisms and data distribution services.

[15] introduces the TeSSLa-ROS-Bridge (TRB), which enables interaction with ROS-based robotic systems via the temporal stream-based specification language TeSSLa. Using TRB, the resulting monitor runs in parallel with the actual robot code and ensures the given safety conditions. [16] proposes a runtime verification method called Robot Monitor on Multilayer (RMoM). RMoM integrates resources, communications, robots, and cluster properties into a systematic hierarchical monitoring framework to detect violations of specified temporal properties during robotic cluster operations.

[17] proposes a state machine-based notation called RoboChart. RoboTool, designed specifically for RoboChart, supports mathematical modeling of controllers and can automatically generate executable code. A robotic case study involving an exploration task demonstrates the application of RoboChart and its associated tools.

To ensure the safety and reliability of ROS, many researchers have employed formal methods in their studies. [18] focuses on the abstraction, modeling, and automated verification of DDS in ROS 2. By constructing a distributed model in PRISM and configuring interface parameters, properties such as safety and liveness are verified for DDS in ROS 2. [19] proposes an automatic static verification technique for system-level safety properties in ROS-based applications. [20,21] utilize model-driven engineering to simplify the formal verification of ROS 2 applications. Their approach generates traces from ROS 2 execution logs, models ROS 2 using UPPAAL, and employs model checking to verify system properties. While these studies all focus on verifying ROS, they approach it from different perspectives. [18] primarily addresses DDS verification; [19] introduces a static automated verification technique; and [20,21] propose a model-driven engineering verification approach.

 doi:10.20944/preprints202604.1883.v1

Based on the preceding analysis, formal methods for the XRCE-DDS communication mechanism in Micro-ROS have rarely been studied. This paper focuses on the XRCE-DDS communication mechanism by investigating its formal modeling, simulation, and verification.

3. Framework of Micro-ROS Modeling and Analysis

3.1. Micro XRCE-DDS Communication Mechanism

Micro-ROS is a lightweight robot operating system optimized based on ROS 2. It integrates resource-constrained embedded device nodes into ROS 2 DDS and enables communication with these devices through the tailored Micro XRCE-DDS middleware [22] . The architecture of the Micro XRCE-DDS communication mechanism is illustrated in Figure 1 [23].

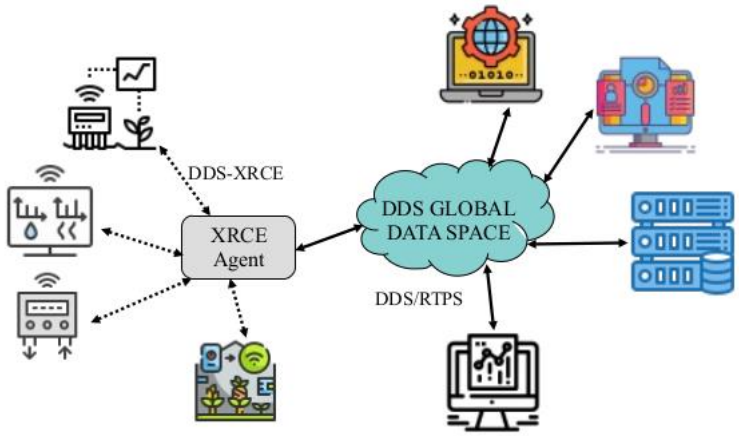

Figure 1. The architecture of Micro XRCE-DDS communication middleware.

Micro XRCE-DDS is a protocol based on a client-server architecture that enables communication between resource-constrained embedded devices and DDS. It consists of servers (XRCE Agents) and clients (XRCE Clients). The XRCE Agent acts as an intermediary between the XRCE Client and the DDS global data space, serving as a bridge between embedded devices and DDS. Each XRCE Client on the embedded device side connects to the DDS domain through the XRCE Agent. The XRCE Agent represents the XRCE Client and interacts as a peer with other entities in the DDS domain.

Micro XRCE-DDS is designed with transport protocol independence. It ensures reliable communication while remaining agnostic to the underlying transport protocol, thereby decoupling its functionality from specific transport layer implementations.

3.2. Time Automata

Real-time performance is a critical characteristic of embedded devices. We employ timed automata to model real-time behavior, . Timed automata are effective for modeling systems with timed characteristics. A timed automaton comprises a finite set of time variables called clocks. Clocks differ from ordinary variables due to restricted access, which implicitly increments as time progresses. Clocks have only two operation: read and reset. All clocks advance at a uniform rate. After the d time units have elapsed, each clock's value increases by d. Thus, a clock's value represents the time elapsed since its last reset.

Definition 1 *Clock Constraint* Clock constraints are primarily used to restrict possible time expenditures. They are defined as follows:

$$tc ::= x < c \mid x > c \mid x \leq c \mid x \geq c \mid tc \wedge tc'$$

where, c is a natural number.

 doi:10.20944/preprints202604.1883.v1

C denotes a finite set of clock variables, where any variable x in C is a clock variable. Additionally, $CC(C)$ represents the set of clock constraints defined over the clock set C.

Definition 2 *timed automata* A timed automata TA is a tuple:

$$\mathrm{TA} = (\mathrm{Loc}, \mathrm{Loc}_0, \mathrm{Act}, C, \mathrm{Inv}, \mapsto)$$

Where

- Loc: is the set of locations.
- $Loc_0 \subseteq Loc$ is the initial location set.
- Act is a finite set of actions or events. Here, actions are divided into two types: general actions and communication actions. Specifically: $c!msg$ denotes sending a message msg through channel c; $c?msg$ denotes receiving a message via channel c and storing it in the variable msg.
- C is the finite set of clocks.
- inv: $Loc \to CC(C)$ is the Invariant Function.
- $\mapsto \subseteq Loc \times Cond(C) \times Act \times Loc$ is the set of transition relation.

The transition relation is denoted by a quadruple (l, g, a, l'), where $g \in Cond(C)$ is a clock constraint with $Cond(C) \subseteq CC(C)$, and a is an action. $(l, g, a, l') \in \mapsto$ indicates that when g is satisfied, a transition occurs from location l to l', and the action a is executed during this transition. The function $inv()$ assigns an invariant to each location,, representing the time constraint for remaining there. The clock invariant $inv()$ must be satisfied while the system is at that location. If this invariant is violated, the system must leave the current location.

3.3. Framework of the Micro-ROS

The framework for formal modeling and analysis of micro-ROS communication mechanism, XRCE-DDS, is illustrated in Figure 2. For XRCE-DDS, timed automata are employed to abstract and construct models of the XRCE-DDS Agent and Client from open-source code. Key properties of the XRCE-DDS specification are extracted and specified using Constrained Signal Temporal Logic. The model is implemented and simulated in Simulink. Subsequently, runtime verification of key properties is performed using Simulink's verification toolkit. Through simulation and verification, a model of XRCE-DDS extracted from the code is satisfied with the specification.

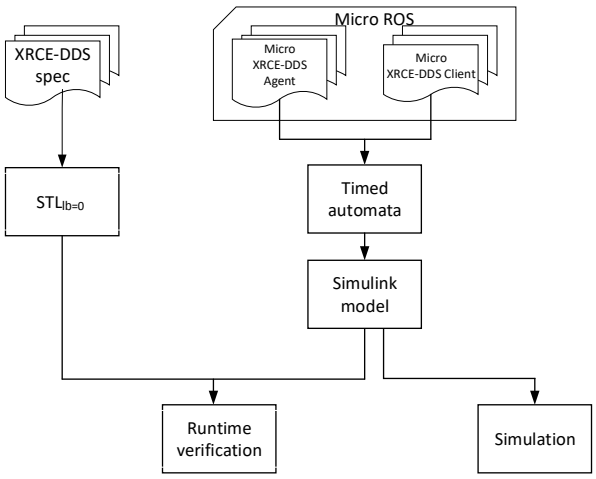

Figure 2. The framework of XRCE-DDS formal modeling, simulation, and runtime verification.

4. Formal Modeling the Micro-ROS

Micro XRCE-DDS of the micro-ROS communication mechanism consists of Agents and Clients. Based on the functionalities of these components, the system is modeled as five modules:

 doi:10.20944/preprints202604.1883.v1

client_publisher, client_subscriber, Agent, proxy_publisher, and proxy_subscriber modules. The modular architecture is illustrated in Figure 3.

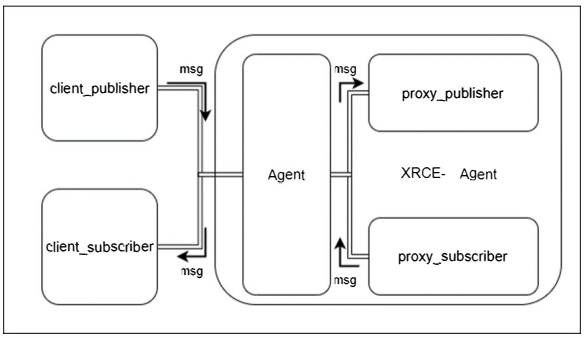

Figure 3. Architecture diagram of the model.

The five modules in the architecture correspond to three types of devices: resource-constrained devices, Agents, and resource-unconstrained devices.

The client_publisher module and client_subscriber module represent two embedded devices operating in resource-constrained environments, such as cameras and alarm systems. These modules correspond to two functional roles of the Micro XRCE-DDS Client. The first acts as a publisher in the Micro XRCE-DDS domain, sending messages to a specified topic within the ROS 2 network. The other acts as a subscriber, receiving messages from topics of interest in the DDS domain.

The proxy_publisher and proxy_subscriber modules represent two non resource unconstrained devices. The proxy_publisher receives messages from the client_publisher, while the client_subscriber receives messages from the proxy_subscriber. In this process, the client_publisher and proxy_publisher form a corresponding client-proxy pair. The client_subscriber and proxy_subscriber form another client-proxy pair. The Agent module is responsible for receiving transmitted messages and performing selective distribution.

4.1. Client_Publisher Model

The client_publisher module comprises four functional components: a reliable input stream, a reliable output stream, a heartbeat reply message stream, and message transmission. These components are modeled using four concurrent timed automata, denoted as TA_{CP_RIS}, TA_{CP_ROS}, TA_{CP_HAS}, and TA_{CP_FS}.

The overall timed automaton model of the client_publisher is defined as:

$$TA_{CP} = TA_{CP\_RIS} \parallel TA_{CP\_ROS} \parallel TA_{CP\_HAS} \parallel TA_{CP\_FS}$$

Only the TA_{CP_ROS} model is provided here. TA_{CP_ROS} models the reliable output stream, and its timed automaton model is as follows:

Definition 3 TA_{CP_ROS} is a tuple:

$$TA_{CP\_ROS} = (Loc_{CP\_ROS}, Loc_{0CP\_ROS}, Act_{CP\_ROS}, C_{CP\_ROS}, Inv_{CP\_ROS}, \mapsto_{CP\_ROS})$$

Where,

- $Loc_{CP\_ROS} = \{presending, sending\}$.
- $Loc_{0CP\_ROS} = \{presending\}$.
- $Act_{CP\_ROS} = \{a\_queue\_msg, c\_balance\}$.
- $C_{CP\_ROS} = \{t\}$ represents the clock set.
- $inv_{CP\_ROS} = \{(presending, t \leq 1), (sending, t \leq 3)\}$. It is the invariant function in the reliable output stream model.

 doi:10.20944/preprints202604.1883.v1

- $\mapsto$ is the set of transition relation in TA_{CP_ROS}, shown as Figure 4.

In the action set, *c_balance* denotes a reliable channel used to receive responses to messages. *a_queue_msg* is an unreliable message channel with a capacity of 1. Messages transmitted through *a_queue_msg* have a structured format with two parts: DATA and HB. DATA contains the message content, while HB contains heartbeat information.

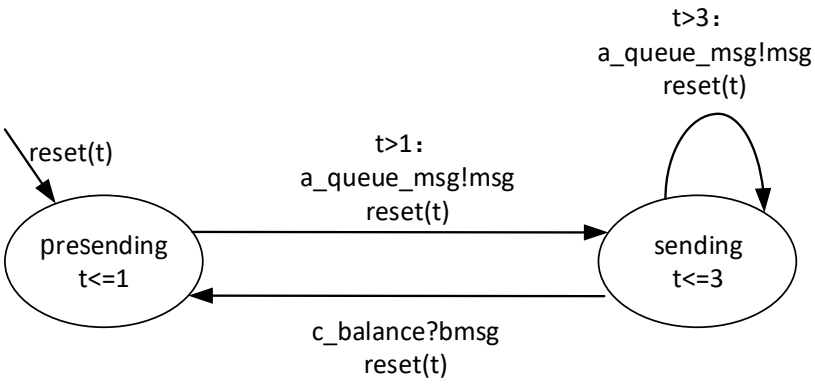

Figure 4. Time automata model TA_{CP_ROS}.

In this timed automaton model, the location *presending* indicates preparing to send a message, with the location invariant being $t \leq 1$. When the time guard $t > 1$ is satisfied, a transition occurs: the message *msg* is sent via the channel *a_queue_msg*, and the clock t is reset. The location transitions from *presending* to *sending*.

The location *sending* represents the message sending process. When the time guard $t > 3$ is met, the previous message sent successfully and another message is transmitted through the channel *a_queue_msg*, and the clock t is reset.

Since *a_queue_msg* is an unreliable channel, message loss may occur during transmission. If a message is received *bmsg* from the reliable channel *c_balance* during 3 time units, a previously sent message was lost and needs to be retransmitted. The location then turns from *sending* back to *presending*, ready to resend the message.

4.2. Client_Subscriber Model

Similar to the client_publisher, the client_subscriber also consists of four functional components: a reliable input stream, a reliable output stream, a heartbeat reply message stream, and message transmission. However, in terms of message transmission, unlike the client_publisher, it includes not only a sending model but also a receiving model. Therefore, this module consists of five timed automata, labeled TA_{CS_RIS}, TA_{CS_ROS}, TA_{CS_HAS}, TA_{CS_FS}, and TA_{CP_RM}.

The overall timed automaton model of the *client_subscriber* can be expressed as:

$$TA_{CS} = TA_{CS\_RIS} \parallel TA_{CS\_ROS} \parallel TA_{CS\_HAS} \parallel TA_{CS\_FS} \parallel TA_{CS\_MS}$$

Here we provide the definition of TA_{CS_ROS}. Since the client_subscriber must account for the device's sleep characteristics, the TA_{CS_ROS} automaton incorporates this feature by adding a *sleep* location. The formal model TA_{CS_ROS} is defined as follows.

Definition 4 TA_{CS_ROS} is a five tuple:

$$TA_{CS\_ROS}=(Loc_{CS\_ROS}, Loc_{0CS\_ROS}, Act_{CS\_ROS}, C_{CS\_ROS}, Inv_{CS\_ROS}, \mapsto_{CS\_ROS})$$

Where,

- $Loc_{Cs\_ROS} = \{init, waitRet, timeout, sleep\}$.
- $Loc_{0CS\_ROS} = \{init\}$.

 doi:10.20944/preprints202604.1883.v1

- $Act_{CS\_ROS} = \{a\_set\_push\_sub\_msg, c\_subs\_success, a\_storeInfo, a\_restoreInfo, f\_sub\_finished, leftwaketime\}$.
  The communication actions are $a\_set\_push\_sub\_msg$, $c\_subs\_success$ and $f\_sub\_finished$. $a\_set\_push\_sub\_msg$ is a channel with a certain capacity, capable of facilitating asynchronous communication.
- $C_{CS\_ROS} = \{t_1, t_2\}$ represents the clock set. t_1 is the clock used to record awake and sleep durations. t_2 tracks whether the sending time has exceeded the timeout threshold.
- $inv_{CS\_ROS} = \{(init, t_1 \leq 100), (waitRet, (t_1 \leq 100) and (t_2 \leq 9)), (timeout, (t_1 \leq 100) and (t_2 \leq 10)), (sleep, t_1 \leq 100)\}$.
- $\mapsto$ is the set of transition relation in TA_{CS_ROS}, shown as Figure 5.

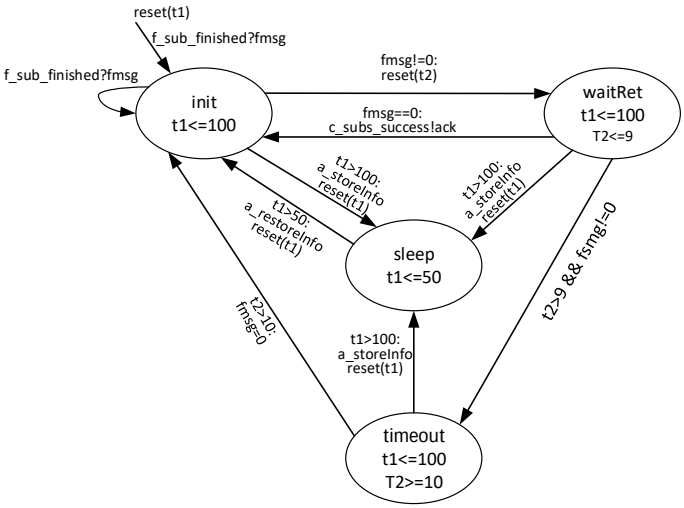

Figure 5. Time automata model TA_{CS_ROS}.

In the location set, *init* represents the initial location where the reliable output stream begins. *waitRet* denotes the location where the reliable output stream awaits the return result of the subscription message after sending it. *timeout* indicates the location reached when the reliable output stream times out while waiting. *sleep* represents the location where the client_subscriber model enters sleep mode, causing the reliable output stream to cease operation.

In the initial state *init*, it is assumed that the embedded device is awake and the clock t_1 is reset. Once the channel $f\_sub\_finished$ receives a message from the proxy subscribe model, which is stored in the variable $fmsg$, and the message is correct, the location transitions to *waitRet*. Additionally, the clock t_2 is reset. If the clock $t_2 > 9$ and $fmsg \neq 0$ at the location *waitRet*, the received message is incorrect and the system changes to the location *timeout*. After remaining in the location *timeout* for one time unit, the system returns to *init* to attempt receiving the message again. The proxy subscribe model re-sends this message after the timeout.

An response message is sent through the channel *c_subs_success* to indicate that the message has been received correctly, and the system returns to the initial location *init*.

Assuming that the embedded device has an awake duration of 100 time units and a sleep duration of 50 time units, if the device remains awake in the locations *init*, *waitRet*, or *timeout* for more than 100 time units, it immediately enters the location *sleep*. The device remains dormant for 50 time units in the location *sleep*. When the device enters the location *sleep*, it stores the current context via the action $a\_storeInfo$. Upon awakening, the context is restored via the action $a\_restoreInfo$.

4.3. Agent Model

In the Micro XRCE-DDS Agent, there is a module responsible for retrieving various types of messages from the network and placing them into a scheduler, which utilizes a multi-level priority queue. Another module reads messages from the queue, distributes them to different ProxyClient

 doi:10.20944/preprints202604.1883.v1

objects based on their IDs, and processes them further. Based on these two steps, the Agent model abstracts two functionalities: message reception and message distribution. Message reception acquires incoming messages from the network, while message distribution determines the target agent of a message based on its content and forwards it accordingly. Both message reception and distribution can be modeled using two timed automata: TA_{REV} and TA_{DIS}. The overall Agent module model can be represented as following.

$$TA_{AG} = TA_{REV} \parallel TA_{DIS}$$

In the agent module, the TA_{REV} model is defined below.

Definition 5 TA_{REV} is a tuple:

$$TA_{AG} = (Loc_{REV}, Loc_{0REV}, Act_{REV}, C_{REV}, Inv_{REV}, \mapsto_{REV})$$

Where,

- $Loc_{REV} = Loc_{0REV} = \{idle\}$.
- $Act_{REV} = \{a\_queue\_msg\}$.
- $C_{REV} = \{t\}$.
- $inv_{REV} = \{(idle, t \leq 1)\}$.
- The transition relation is shown as Figure 6.

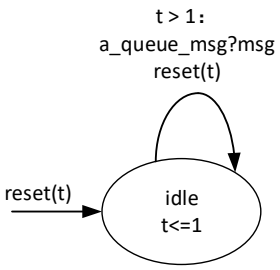

Figure 6. Time automata model TA_{REV}.

In the timed automaton TA_{REV} model, when in the *idle* location and the time guard condition $t > 1$ is satisfied, the automaton receives a message from the channel *a_pop_push_msg*, stores it in the variable *msg*, and resets the clock t to wait for the next message reception. The channel *a_pop_push_msg* has a certain capacity, enabling asynchronous communication. When the Agent module receives a message, it forwards the message to either the proxy_publisher module or the client_subscriber module via TA_{DIV}, depending on the type of message.

4.4. Proxy_Publisher Model

The proxy_publisher module represents a ProxyClient entity in a Micro XRCE-DDS. The ProxyClient is responsible for receiving data from the client_publisher through the Agent module and forwarding it to the DDS domain.

According to the description of XRCE-DDS in the XRCE-DDS protocol, when the XRCE-DDS Agent acts as a publisher proxy, it receives messages from the client_publisher. The proxy_publisher needs to abstract a timed automaton for the reliable input stream to process data received from the Agent module.

Therefore, the ProxyClient entity is abstracted into timed automata models responsible for the reliable input stream, heartbeat messages, and message transmission. The three timed automata constructed for the proxy_publisher module are TA_{PP_RIS}, TA_{PP_HAS}, and TA_{PP_FS}. The overall timed automaton model for the proxy_publisher module can be expressed as:

 doi:10.20944/preprints202604.1883.v1

$$TA_{PP} = TA_{PP\_RIS} \parallel TA_{PP\_HAS} \parallel TA_{PP\_FS}$$

In the proxy_publisher module, TA_{PP_HAS} is the heartbeat response module, and its model is as follows.

Definition 6 TA_{PP_HAS} is a tuple:

$$TA_{PP\_HAS}=(Loc_{PP\_HAS},Loc_{0PP\_HAS},Act_{PP\_HAS},C_{PP\_HAS},Inv_{PP\_HAS},\mapsto_{PP\_HAS})$$

Where,

- $Loc_{PP\_HAS} = \{Rev, MissMsg, CheckSeq\}$.
- $Loc_{0PP\_HAS} = \{Rev\}$.
- $Act_{PP\_HAS} = \{ack\_msg, nack\_msg, HB\}$.
- $C_{PP\_HAS} = \varnothing$.
- $inv_{PP\_HAS} = \varnothing$.
- $\mapsto$ is the set of transition relation in TA_{PP_HAS}, shown as Figure 7.

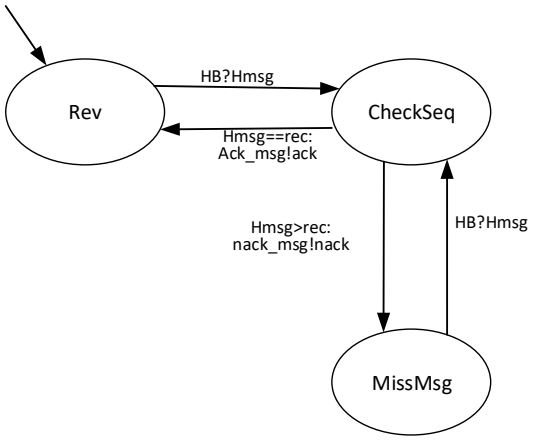

Figure 7. Time automata model TA_{PP_HAS}.

In the Act_{PP_HAS} set, three channels have been established. One is for heartbeat messages, one for response messages, and another for reporting packet loss. Heartbeat messages are received through the HB channel and packet loss is checked. If no packet loss is detected, a response message is sent through the ack_msg channel. If packet loss occurs, the relevant information is transmitted via the $nack\_msg$ channel.

TA_{PP_HAS} receives messages through the HB channel, unpacks them, and checks for packet loss. If the sequence number in the heartbeat message is greater than the recorded message number, it indicates that the packet was lost earlier. In this case, a message is sent through the $nack\_msg$ channel to request retransmission. Otherwise, if the message is received, a response is sent through the ack_msg channel to confirm receipt.

4.5. Proxy_Subscriber Model

The proxy_subscriber module serves as the entity model corresponding to the client_subscriber for the Micro XRCE-DDS Agent. It responds to client_subscriber subscriptions and sends data to the client_subscriber.

As described in the XRCE-DDS protocol, when the XRCE-DDS Agent acts as a proxy for subscribers, it must receive messages from the proxy_subscriber and retransmit them to the client_subscriber. To ensure consistency between sent and received messages, it is necessary to periodically send heartbeat messages and receive response messages. Furthermore, since the XRCE-DDS Client may periodically enter sleep mode, the proxy_subscriber must not send messages once the Client has entered the sleep state.

 doi:10.20944/preprints202604.1883.v1

The proxy_subscriber module utilizes the reliable input stream module TA_{PS_RIS} and the reliable output stream module TA_{PS_ROS} to process incoming and outgoing messages. It also handles heartbeat messages to be sent and processes received response messages, which are modeled using the timed automaton TA_{PS_HAS}. The proxy_subscriber sends two types of messages: basic messages requiring reliable transmission and heartbeat messages ensuring reliable delivery. This sub-function also checks whether the client_subscriber module is in a sleep state. The proxy_subscriber can only send messages when the client_subscriber is awake. A timed automaton, TA_{PS_FSWT}, is constructed for Periodic Sending. The proxy_subscriber module consists of four timed automata models: TA_{PS_RIS}, TA_{PS_ROS}, TA_{PS_HAS}, and TA_{PS_FSWT}. The proxy_subscriber module can be represented as:

$$TA_{PS} = TA_{PS\_RIS} \parallel TA_{PS\_ROS} \parallel TA_{PS\_HAS} \parallel TA_{PS\_FSWT}$$

For the TA_{PS_FSWT} timed automaton, according to the description of the XRCE-DDS Agent in the XRCE-DDS protocol, it is necessary to receive the subscriber messages and respond accordingly. According to the XRCE-DDS protocol, the timed automaton TA_{PS_FSWT} must receive sleep or awake messages from the client_subscriber and respond accordingly.

The XRCE-DDS Agent cannot send messages when the client_subscriber is in sleep state. This timed automaton is defined as follows.

Definition 7 TA_{PS_SFWT} is a tuple:

$$TA_{PS\_SFWT}=(Loc_{PS\_SFWT},Loc_{0PS\_SFWT},Act_{PS\_SFWT},C_{PS\_SFWT},Inv_{PS\_SFWT},\mapsto_{PS\_SFWT})$$

Where,

- $Loc_{PS\_SFWT} = \{clientSleep, clientWake, flushMsg\}$.
- $Loc_{0PS\_SFWT} = \{clientSleep\}$.
- $Act_{PS\_SFWT} = \{C\_stat, C\_sleep, sleep\_flag, wake\_flag\}$.
- $C_{PS\_SFWT} = \{t\}$.
- $inv_{PS\_SFWT} = \{(clientWake, t <= 100), (flushMsg, t <= 100)\}$.
- $\mapsto$ is the set of transition relation in TA_{PS_SFWT}, shown as Figure 8.

In the location set LOC_{PS_FSWT}, *clientSleep* is the initial location, representing that the client_subscriber, is asleep. Upon receiving a wake-up signal from the client_subscriber, a transition occurs from the *clientSleep* to the *clientWake* location, indicating that the client_subscriber has awaked. The $flushMsg$ location denotes the location in which the proxy_subscriber can transmit messages. Once a sleep message is received from client_subscriber, the system enters the location *ReadySleep* and then transitions to the location *clientSleep*, and stops transmitting messages.

In the action set Act_{PS_SFWT}, there are four actions. When the channel C_start receives the wake-up signal, the embedded device is awakened. The channel C_sleep receives the sleep signal, and the embedded device is going to enter a sleep state. The channel $sleep\_flag$ sends a message instructing the proxy_subscriber to stop transmitting messages to the embedded device. The channel $wake\_flag$ transmits a message to the proxy_subscriber model, which then sends data to the client_subscriber module.

 doi:10.20944/preprints202604.1883.v1

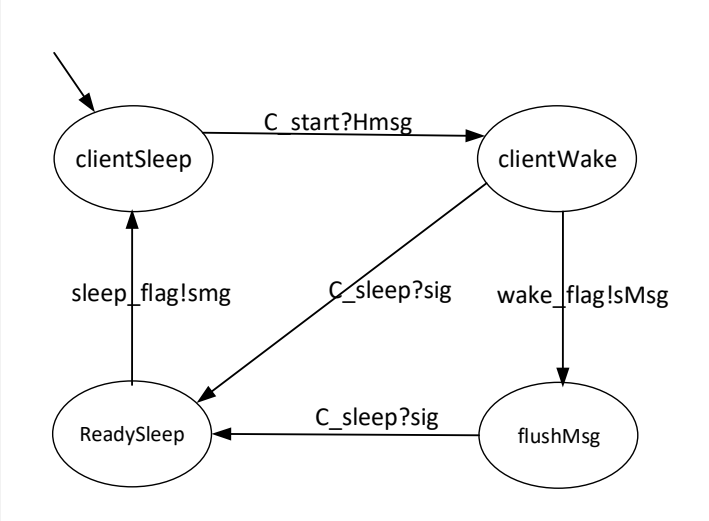

Figure 8. Time automata model TA_{PS_SFWT}.

The embedded device is sleep in the initial location *clientSleep*. Once it wakes up, the embedded device sends a signal to the proxy_subscriber module via the *C_start* channel, indicating that the embedded device has entered the awakened state. The TA_{PS_FSWT} then sends a signal to TA_{PS_ROS} notifying it that the embedded device is awake, which allows TA_{PS_ROS} to send messages. If the embedded device needs to enter the sleep state again, TA_{PS_FSWT} sends a signal to TA_{PS_ROS} indicating that the embedded device is going to sleep and no further messages should be sent.

5. Properties of Micro-ROS Model

To perform runtime verification of the model, it is to extract the key properties from the Micro-ROS communication specifications and represent them using constrained signal temporal logic formulas.

5.1. Constrained Signal Temporal Logic

Signal Temporal Logic (STL) is a temporal logic used to describe the behavior of continuous-time signals. By incorporating real-valued time variables and temporal operators, STL formally characterizes the dynamic properties of systems over continuous time intervals. Its semantics are based on measurements over time intervals, which provides a logical language for verifying and monitoring of real-time systems.

The syntax of an STL formula is defined as follows:

$$\phi ::= \texttt{true} \mid p \mid \neg\phi \mid \phi_1 \wedge \phi_2 \mid \phi_1 \mathcal{U}_{[a,b]} \phi_2$$

For the time interval $[a, b]$, $a, b \in \mathbb{R}_{\geq 0}$ and $a \leq b$. In the Micro-ROS communication model, the focus is generally on the upper bound of the response time when describing the model's properties, while the lower bound corresponds to the current moment. Therefore, to better represent the Micro-ROS requirements, we propose a constrained signal temporal logic, denoted as $\mathtt{STL}_{\mathrm{lb}=0}$. The syntax and semantics of $\mathtt{STL}_{\mathrm{lb}=0}$ are defined below.

Definition 8 An $\mathtt{STL}_{\mathrm{lb}=0}$ formula ϕ is recursively defined:

$$\phi ::= p \mid \neg\phi \mid \phi_1 \vee \phi_2 \mid \phi_1 \, \mathsf{U}_{[0,a]} \, \phi_2$$

Where p is an atomic proposition, a in the time interval $[0, a]$ denotes the upper bound of the time interval, and $a > 0$. ϕ_1 and ϕ_2 are $\mathtt{STL}_{\mathrm{lb}=0}$ formulas.

Other operators, such as the implication operator $\rightarrow$, the future operator $\mathtt{F}$, and the always operator $\mathtt{G}$, can be expressed as follows:

 doi:10.20944/preprints202604.1883.v1

$$\phi_1 \rightarrow \phi_2 = \neg\phi_1 \vee \phi_2$$
$$\mathrm{F}_{[0,a]}\,\phi = \mathrm{True}\,\mathrm{U}_{[0,a]}\,\phi$$
$$\mathrm{G}_{[0,a]}\,\phi = \neg\mathrm{F}_{[0,a]}\,\neg\phi$$

The satisfaction relation $(s,t) \models \varphi$ denotes if a signal s satisfies an $\mathtt{STL}_{\mathrm{lb}=0}$ formula ϕ at time t.

Definition 9 The $\mathtt{STL}_{\mathrm{lb}=0}$ semantics for a signal s is given recursively below.

$$\begin{aligned}
&(s,t) \models p \iff p(s(t)) = T\\
&(s,t) \models \neg\phi \iff (s,t) \not\models \phi\\
&(s,t) \models \phi_1 \vee \phi_2 \iff (s,t) \models \phi_1 \vee (s,t) \models \phi_2\\
&(s,t) \models \phi_1\, U_{[0,a]}\, \phi_2 \iff \exists t' \in [t, t+a],\ (s,t') \models \phi_2 \quad \text{and} \quad \forall t'' \in [t,t'],\\
&\qquad (s,t'') \models \phi_1
\end{aligned}$$

Where $p(s(t))$ indicates that the signal s satisfies property p at time t. For example, consider the property: "Within 200 seconds, if property p is satisfied, then property q will be satisfied within 3 seconds." This property can be expressed using the $\mathtt{STL}_{\mathrm{lb}=0}$ formula as:

$$\mathrm{G}_{[0,200]}(p \rightarrow \mathrm{F}_{[0,3]}q)$$

the outermost $\mathrm{G}_{[0,200]}$ represents the duration of the system's monitoring. During this monitoring period, if property p is satisfied, then property q will also be satisfied within 3 seconds.

5.2. Key Properties Abstraction

To ensure the reliability of communication, seven key properties are extracted based on the Micro-ROS communication specifications. These properties primarily address message transmission and reception, ensuring consistent ordering of messages and the ability to retransmit them in case of loss.

- **property ECFS**: All messages sent by the client_publisher can be delivered.
  The property is formulated as follows:

  $$\mathrm{F}_{[0,180]}client\_pub\_finished = 1$$

  In runtime verification, when the proxy_publisher receives all messages sent by the client_publisher, the signal $client\_pub\_finished$ eventually becomes 1, indicating that all messages have been delivered.
- **property ECRA**: The client_subscriber can receive all messages to which it has subscribed.
  This property is formulated as follows:

  $$\mathrm{F}_{[0,180]}client\_received\_all = 1$$

  In runtime verification, when the client_subscriber has received all subscribed messages, the signal $client\_received\_all$ eventually becomes 1.
- **property NIWS**: Once the client_subscriber enters sleep mode, no additional messages will arrive to wake it.
  This property is formulated as follows:

  $$\mathrm{G}_{[0,180]}(subClient\_status = 1 \rightarrow \mathrm{G}_{[0,50]}subClient\_receiveSig = false)$$

 doi:10.20944/preprints202604.1883.v1

When the client_subscriber enters sleep mode (indicated by the signal *subClient_status* changing to 1), it will no longer passively receive messages from the proxy_subscriber. At this point, the value of the signal *subClient_receiveSig* for message arrivals becomes false and remains so throughout the sleep duration (50 seconds).

- **property RMIO**: The proxy_publisher always receives messages in the same order as they are sent by the client_publisher.
  This property is formulated as follows:

$$G_{[0,180]}(publisher\_last\_received \geq 10 \rightarrow VideoInfo = TDDS)$$

  When the proxy_publisher receives the 10th message (i.e., the final message), the $VideoInfo$ array containing the messages to be sent must be identical to the TDDS array of received messages. Since the array indices correspond to the sequential order of message transmission and reception, identical arrays indicate that the order of message sending and receiving order is consistent.
- **property RMIO2**: The client_subscriber always receives messages in the same order as they are sent by the proxy_subscriber.
  This property is formulated as follows:

$$G_{[0,180]}(subscriber\_last\_received \geq 10 \rightarrow GetInfo = FDDS)$$

  When the client_subscriber receives the 10th message (i.e., the final message), the $GetInfo$ array containing the messages to be sent must be identical to the FDDS array of received messages. Since the array indices correspond to the sequential order of message transmission and reception, identical arrays indicate that the order of messages sending and receiving is consistent.
- **property TMRK**: When a message is lost, the receiver can detect it in a timely manner.
  This property is formulated as follows:

$$G_{[0,180]}(missing\_fig = 1 \&\& sendMissSeq < last\_receive) \rightarrow$$

$$F_{[0,20]} findMissSeq = sendMissSeq$$

  If a message is not acknowledged by the receiver, the receiver will detect the loss within 20 seconds and notify the sender of the missing sequence number by setting the $findMissSeq$ signal.
- **property FARS**: All messages not received by the client_subscriber will be promptly resent by the proxy_subscriber.
  This property is formulated as follows:

$$G_{[0,180]}(last\_sent > last\_ack + 1 \rightarrow F_{[0,20]} last\_sent = last\_ack + 1)$$

  When the client_subscriber fails to receive a message, the *last_sent* value of the proxy_subscriber will be greater than the acknowledged value of *last_ack*. The proxy_subscriber attempts to resend the message; upon successful transmission, it adjusts the *last_sent* value to match *last_ack* within 20 seconds.

6. Implementation, Simulation and Verification

We utilized Simulink/Stateflow to implement and simulate each module, and analyzed the results. We also applied the Logical and Temporal Assessments tool from Simulink's Simulink Test package Test Manager to perform runtime verification of key properties extracted from the Micrio-ROS specification.

Here, we take a specific and commonly used application scenario as an example and instantiate the model using Simulink/Stateflow. In this case study, an embedded device-controlled camera acts as an XRCE Client that publishes messages in the DDS domain. Additionally, an embedded device-

 doi:10.20944/preprints202604.1883.v1

controlled alarm functions as an XRCE Client that subscribes to messages in the DDS domain and has a periodic sleep characteristic. Furthermore, a PC runs an XRCE Agent, which acts as a proxy for both the camera and the alarm. The top-level module of the Simulink/Stateflow model is shown in Figure 9.

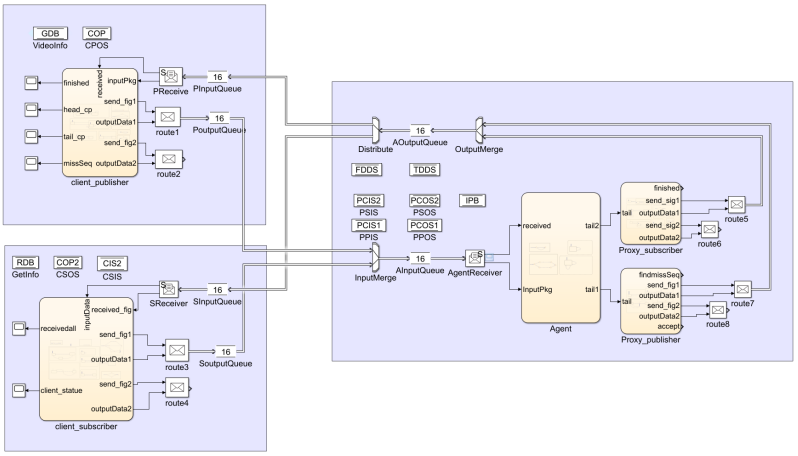

Figure 9. System model of Simulink/Stateflow.

6.1. Simulation and Result Analysis

In Simulink, the model's simulation duration is set to 160 seconds. The final simulation results are observed through the Scope, as shown in Figure 10.

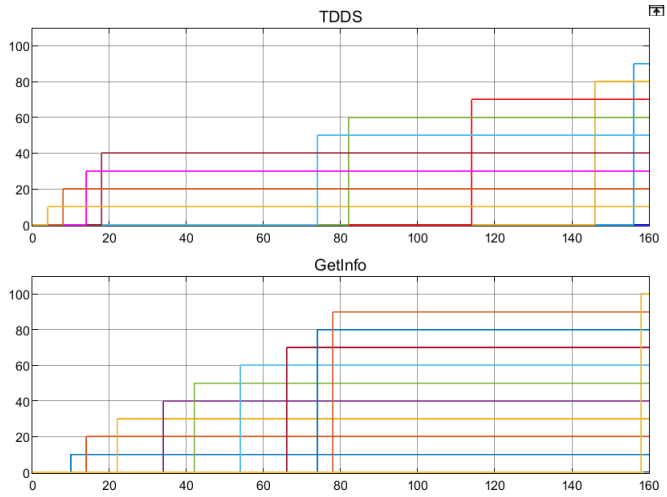

Figure 10. Simulated receiving messages in 160 seconds.

In Figure 10, the horizontal axis represents time, and the vertical axis indicates the number of received messages. TDDS denotes the proxy_publisher in the XRCE-DDS Agent, which receives information from the camera. GetInfo represents the alarm in the XRCE-DDS Client, which receives information from the DDS domain. The varying time intervals between consecutive message receptions demonstrate that the experiment successfully simulated delays in message reception caused by transmission loss in an unreliable network. However, TDDS ultimately failed to receive all messages, primarily due to an error in calculating the simulation time. The maximum detection time was set to only 160 seconds, which was insufficient for receiving and acknowledging all messages. This is because our simulation model includes the emulation of message loss, where some messages are retransmitted after being lost, thereby delaying their normal transmission.

We modified the simulation time in the environment to 180 seconds. After running the simulation again, the data reception status shown in Figure 11 was observed.

 doi:10.20944/preprints202604.1883.v1

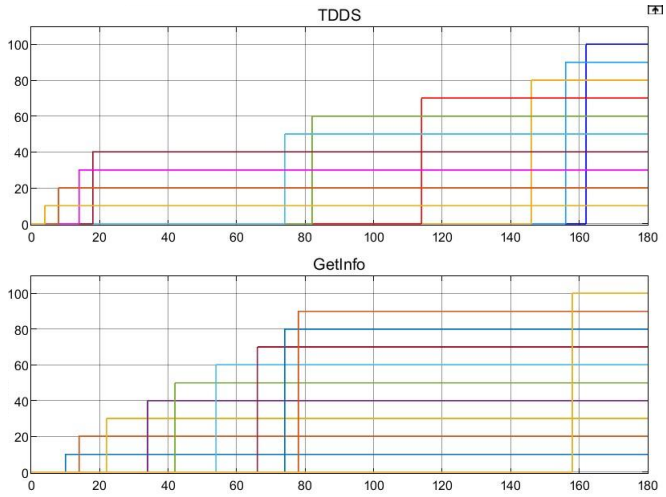

Figure 11. Simulated receiving messages in 180 seconds.

Figure 11 shows that after *GetInfo* receives the 9th message, approximately 80 seconds pass before the 10th message is received. This is due to the resource-constrained embedded device entering a sleep period. During this interval, the XRCE-DDS Client enters a sleep state and cannot receive messages from the XRCE-DDS Agent. When the XRCE-DDS Client wakes up and returns to an active state, it resumes receiving messages and successfully receives the 10th message from the XRCE-DDS Agent.

6.2. Runtime Verification and Result Analysis

To verify the properties of Micro-ROS, the Logical and Temporal Assessments tool from Simulink Test's Test Manager package is applied for runtime verification. This tool accepts properties described in natural language and converts them into signal temporal logic(STL). The STL formulas generate a syntax trees, which is then evaluated from the leaf nodes upward to construct monitors for runtime verification.

The tool allows properties to be described in a manner close to natural language. Figure 12 shows the interface provided by the tool for engineers to input these properties.

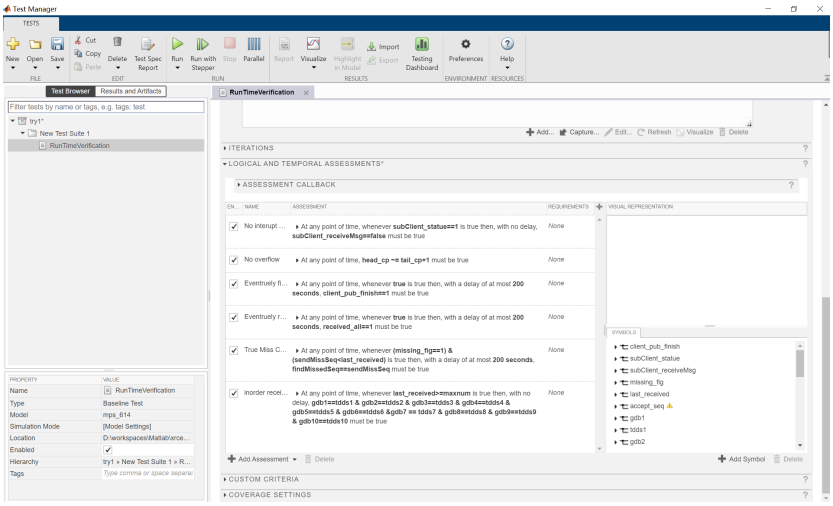

Figure 12. Interface of Test Manager.

When using this tool to describe properties, the engineer adds the signals involved in the property to the tool via the "Add Symbol" button at the bottom-right corner to configure the detector. Then, the specific property specifications are written in the "ASSESSMENT CALLBACK" section. The seven properties of Micro-ROS are defined in the tool as follows.

 doi:10.20944/preprints202604.1883.v1

- ECFS: At any point of time, whenever true is true, with a delay of at most 180 seconds, $client\_pub\_finished == 1$ must be true.
- ECRA: At any point of time, whenever true is true, with a delay of at most 180 seconds, $client_received_all == 1$ must be true
- NIWS: At any point of time, whenever $subClient_statue == 1$ is true, with a delay between 0 seconds and 50 seconds, $subClient\_receiveSig == false$ must be true
- RMIO: At any point of time, whenever $publisher\_last\_received \geq 10$ is true, with no delay, $VedioInfo == TDDS$ must be true
- RMIO2: At any point of time, whenever $subscriber\_last\_received \geq 10$ is true, with no delay, $GetInfo == FDDS$ must be true
- TMRK: At any point of time, whenever $missing\_fig == 1 \&\& sendMissSeq < last\_receive$ is true, with a delay between 0 seconds and 20 seconds, $findMissSeq == sendMissSeq$ must be true
- FARS: At any point of time, whenever $last\_sent > last\_ack + 1$ is true, with a delay between 0 seconds and 20 seconds, $last\_sent = last\_ack + 1$ must be true

The tool was used to perform runtime verification on the seven properties mentioned above, and the model passed successfully all seven verifications. The verification results are presented in Figure 13.

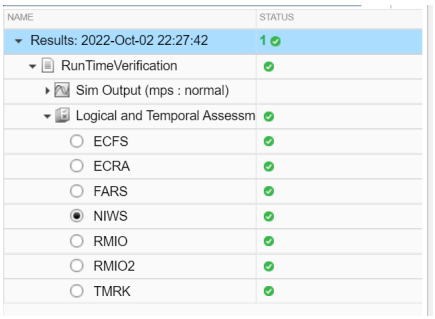

Figure 13. Property verification results.

As shown in Figure 13, the results indicate that all seven key properties have passed verification. The system model successfully transmits messages from resource-constrained embedded devices to other nodes. Additionally, resource-constrained embedded devices can receive messages from other nodes.

7. Conclusion

With the continuous development of the Robot Operating System(ROS), Micro-ROS designed for resource-constrained devices has garnered increasing attention from both academia and industry. To ensure the safety and reliability of Micro-ROS, we studied the Micro-ROS specifications and established a modeling, simulation, and verification framework for its communication mechanisms. This framework helps ensure the safety of communication mechanisms in simulated execution paths. Based on specific Micro-ROS application scenarios, we implemented models on the MATLAB Simulink/Stateflow platform to simulate the Micro-ROS communication process. Additionally, we performed runtime verification using the Simulink Test toolkit for the seven key properties extracted from the Micro-ROS specifications.

We have modeled and analyzed the communication behavior after the communication link was established. Before Micro-ROS performs data communication, the Micro XRCE-DDS Agent needs to authenticate and authorize the Micro XRCE-DDS Client in order to ensure its communication security. Future work will investigate the security of Micro XRCE-DDS Client permissions within the DDS domain from an information security perspective, aiming to prevent device impersonation and mitigate security risks.

 doi:10.20944/preprints202604.1883.v1

Author Contributions: Conceptualization, X.R. and J.G.; methodology, Z.W. L.J. and J.G.; software, Z.W.; validation, X.R., Z.W. and J.G.; formal analysis, Z.W. and X.R.; investigation, L.J.; resources, X.R.; data curation, Z.W.; writing—original draft preparation, Z.W. and X.R.; writing—review and editing, J.G.; visualization, X.R.; supervision, J.G.; project administration, J.G.; funding acquisition, X.R. All authors have read and agreed to the published version of the manuscript.

Funding: This paper was supported by the Shanghai Special Program for Promoting High-Quality Industrial Development (Project No. 250203).

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Data are contained within the article.

References

1. QUIGLEY M, CONLEY K, GERKEY B, et al. ROS: an open-source Robot Operating System[C]//ICRA workshop on open source software, Kobe, May 12 - 17, 2009. Piscataway: IEEE, 2009: 5.
2. Wang, Z.; Liu, S.; Ji, D.; Yi, W. Improving Real-Time Performance of Micro-ROS with Priority-Driven Chain-Aware Scheduling. Electronics 2024, 13, 1658. https://doi.org/10.3390/electronics13091658
3. Micro-ROS - Puts ROS 2 onto Microcontrollers [EB/OL]. (2024-02-13) [2024-05-06]. https://micro.ros.org.
4. LEE H, YOON J, JANG M-S, PARK K-J. A robot operating system framework for secure uav communications[J]. Sensors, 2021,21 (4): 1369.
5. REKE M, et al. A Self-Driving Car Architecture in ROS2[C]//2020 International SAUPEC/RobMech/PRASA Conference, Cape Town, Jan 29-31, 2020. Piscataway: IEEE, 2020:1-6.
6. YANG L G, CHI H F. SLAM Self-Cruise Vehicle Based on ROS Platform[C]// 2021 IEEE International Conference on Consumer Electronics and Computer Engineering (ICCECE), Guangzhou, Jan 15-17, 2021. Piscataway: IEEE, 2021: 6-11.
7. Z. Wang, S. Liu, X. Jiang, D. Ji and Y. Wang, "TIDE: a Timing-Deterministic and Efficient Executor for Micro-ROS,"2023 IEEE International Conferences on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData) and IEEE Congress on Cybermatics (Cybermatics), Danzhou, China, 2023, pp. 487-494, doi: 10.1109/iThings-GreenCom-CPSCom-SmartData-Cybermatics60724.2023.00096.
8. A. Mamani-Saico and P. R. Yanyachi, "Implementation and Performance Study of the Micro-ROS/ROS2 Framework to Algorithm Design for Attitude Determination and Control System," in IEEE Access, vol. 11, pp. 128451-128460, 2023, doi: 10.1109/ACCESS.2023.3330441.
9. J. Zahradník, M. Daňhel and H. Kubátová, "Integration of PXROS-HR with Micro-ROS in Robotic Systems," 2024 13th Mediterranean Conference on Embedded Computing (MECO), Budva, Montenegro, 2024, pp. 1-6, doi: 10.1109/MECO62516.2024.10577777.
10. Dürschmid T, Timperley C S, Garlan D, et al. Rosinfer: Statically inferring behavioral component models for ros-based robotics systems[C]//Proceedings of the IEEE/ACM 46th International Conference on Software Engineering. 2024: 1-13.
11. iklas Terei, Rolf Wiemann, Annika Raatz, ROS-Based Control of an Industrial Micro-Assembly Robot, Procedia CIRP, Volume 130, 2024, Pages 909-914.
12. Dehnavi S., Goswami D., Koedam M., et al. Modeling, implementation, and analysis of XRCE-DDS applications in distributed multi-processor real-time embedded systems. 2021 Design, Automation and Test in Europe Conference and Exhibition (DATE). IEEE, 2021: 1148-1151.
13. H. -S. Park, S. Lee, D. Um, H. Ryu and K. -J. Park, "An Analytical Latency Model of the Data Distribution Service in ROS 2," IEEE INFOCOM 2025 - IEEE Conference on Computer Communications, London, United Kingdom, 2025, pp. 1-10, doi: 10.1109/INFOCOM55648.2025.11044454.
14. G. Zanatta, G. Caiazza, P. Ferrara, L. Negrini and R. White, "Automating ROS2 Security Policies Extraction through Static Analysis," 2024 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Abu Dhabi, United Arab Emirates, 2024, pp. 3627-3634, doi: 10.1109/IROS58592.2024.10802507.
15. Begemann M.J., Kallwies H., Leucker M., Schmitz M. (2023). TeSSLa-ROS-Bridge – Runtime Verification of Robotic Systems. In ICTAC 2023. Lecture Notes in Computer Science, vol 14446. Springer, Cham. https://doi.org/10.1007/978-3-031-47963-2-23

 doi:10.20944/preprints202604.1883.v1

16. C. Hu, W. Dong, Y. Yang, H. Shi and G. Zhou, "Runtime Verification on Hierarchical Properties of ROS-Based Robot Swarms," in IEEE Transactions on Reliability, vol. 69, no. 2, pp. 674-689, June 2020, doi: 10.1109/TR.2019.2923681.
17. Li, W., Ribeiro, P., Miyazawa, A. et al. Formal design, verification and implementation of robotic controller software via RoboChart and RoboTool. Auton Robot 48, 14 (2024). https://doi.org/10.1007/s10514-024-10163-7.
18. Y. Liu, Y. Guan, X. Li, R. Wang and J. Zhang, "Formal Analysis and Verification of DDS in ROS2," 2018 16th ACM/IEEE International Conference on Formal Methods and Models for System Design (MEMOCODE), Beijing, China, 2018, pp. 1-5, doi: 10.1109/MEMCOD.2018.8556970.
19. Carvalho, A. Cunha, N. Macedo and A. Santos, "Verification of system-wide safety properties of ROS applications," 2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Las Vegas, NV, USA, 2020, pp. 7249-7254, doi: 10.1109/IROS45743.2020.9341085.
20. Dust L, Gu R, Mubeen S, Ekström M and Seceleanu C (2025) A model-based approach to automation of formal verification of ROS 2 systems. Front. Robot. AI 12:1592523. doi: 10.3389/frobt.2025.1592523.
21. Dust L., Gu R., Seceleanu, C. et al. Pattern-based verification of ROS 2 applications using UPPAAL. Int J Softw Tools Technol Transfer 27, 313–332 (2025). https://doi.org/10.1007/s10009-025-00802-4.
22. DDS for eXtremely Resource Constrained Environments, https://www.omg.org/spec/DDS-XRCE/1.0/PDF
23. Kucuk, Kerem and Solpan,Sevval. (2022). DDS-XRCE Standard Performance Evaluation of Different Communication Scenarios in IoT Technologies. EAI Endorsed Transactions on Internet of Things. e1. 10.4108/eetiot.v8i4.2691.