Proceedings in Manufacturing Systems, Volume 20, Issue 3, 2025, pp. 117-128 ISSN 2067-9238

MATLAB SIMULATION OF ESP32 INTEGRATION WITH A RASPBERRY PI SOFT-PLC VIA MQTT

Adrian MAROȘAN[1,*], Claudia Emilia GÎRJOB[2], Anca Lucia CHICEA[3], Cristina Maria BIRIȘ[4], Alexandru BÂRSAN[5], Paula DRAȘOVEAN[6], Timotei MORARIU[7]

[1)] PhD, Lecturer., Machines and Industrial Equipment, University "Lucian Blaga" of Sibiu, Faculty of Engineering, Romania
[2)] PhD, Conf., Machines and Industrial Equipment, University "Lucian Blaga" of Sibiu, Faculty of Engineering, Romania
[3)] PhD, Conf., Machines and Industrial Equipment, University "Lucian Blaga" of Sibiu, Faculty of Engineering, Romania
[4)] PhD, Conf., Machines and Industrial Equipment, University "Lucian Blaga" of Sibiu, Faculty of Engineering, Romania
[5)] PhD, Lecturer., Machines and Industrial Equipment, University "Lucian Blaga" of Sibiu, Faculty of Engineering, Romania
[6)] Assist. Prof., PhD Student, Machines and Industrial Equipment, University "Lucian Blaga" of Sibiu, Faculty of Engineering, Romania
[7)] Assist. Prof., PhD Student, Machines and Industrial Equipment, University "Lucian Blaga" of Sibiu, Faculty of Engineering, Romania

Abstract: *This paper presents a MATLAB-based simulation framework for evaluating the temporal behaviour of an ESP32–MQTT–Soft-PLC integration intended for future deployment on a Raspberry Pi Soft-PLC platform. The proposed setup emulates a typical industrial IoT chain in which a virtual ESP32 node acquires synthetic ADC-like sensor data with Gaussian noise, performs preprocessing and normalisation, injects software jitter at task level, and publishes timestamped JSON messages via a Python MQTT client over a TLS-secured connection to a cloud-based MQTT broker. On the control side, a MATLAB Soft-PLC subscribes to the same MQTT topic, updates a process image at the beginning of each scan cycle and executes an ON-delay (TON) timing logic, while logging all relevant timestamps for performance analysis. End-to-end latency is defined as the time difference between publication at the ESP32 node and processing in the Soft-PLC, whereas jitter is evaluated as the deviation of the actual scan time from a nominal scan period. For a 30-second experiment with a 50 ms nominal scan time, the measured end-to-end latency exhibits a mean of 568.06 ms, with values ranging from 64.11 ms to 1144.76 ms, while the Soft-PLC scan time shows an average of 62.64 ms and a maximum of 172.03 ms. These results characterise the architecture as a soft real-time solution suitable for monitoring, data acquisition and supervisory control, but not for hard real-time applications with strict millisecond-level deadlines. The framework thus provides a practical basis for analysing latency and scan jitter before migrating the ESP32–Soft-PLC integration to real Raspberry Pi-based controllers.*

Key words: *ESP32, Raspberry Pi, Soft-PLC, MQTT, MATLAB, latency, real-time simulation, digital twin.*

1. INTRODUCTION

The rapid digitalization of industrial systems has accelerated the transition from traditional hardware PLCs towards open, software-centric control architectures. In these architectures, the control logic defined according to IEC 61131-3 is executed as software tasks on embedded boards or industrial PCs, rather than on dedicated PLC hardware [1, 2]. Open-source initiatives such as OpenPLC demonstrate that PLC functionality can be ported to low-cost platforms like Raspberry Pi while preserving a cyclic scan model, deterministic behavior under nominal conditions and compatibility with existing engineering workflows [3]. Soft-PLC solutions have been deployed in a variety of industrial and energy-related scenarios, ranging from mobile monitoring systems to large-scale renewable energy applications. For instance, mobile devices equipped with Soft-PLC functionality have been proposed for industrial environmental security monitoring, enabling local processing and decision-making close to the physical process [4]. Similarly, Soft-PLC-based controllers have been used to implement yaw control strategies for megawatt-scale wind power generation sets, emphasizing that software-based control can satisfy demanding stability and reliability requirements when properly engineered [5]. At the same time, several works highlight practical engineering challenges related to configuration, robustness and maintenance of PLC and Soft-PLC systems in real plants [6]. The correctness and safety of Soft-PLC applications depend not only on their functional logic but also on their temporal behavior. Research on scheduling methods for embedded operating systems supporting Soft-PLC control emphasizes that the choice of scheduling strategy and the configuration of tasks have a direct impact on scan-cycle time, jitter and deadline satisfaction [6]. These aspects become critical in applications where the control algorithm must react within tight time bounds, as is the case in wind-turbine yaw control or other fast dynamic processes[5]. In addition, practical experiences with PLC systems show that non-deterministic execution, resource contention and inadequate timing analysis can lead to subtle yet significant engineering problems [6]. In parallel with

[*] Corresponding author: Emil Cioran 4, 550025, Sibiu, Romania, Faculty of Engineering
E-mail addresses: *marosan.adrian@gmail.com* (A. Maroșan), *claudia.girjob@ulbsibiu.ro* (C.-E Gârjob)

these developments on the control side, the Internet of Things (IoT) has emerged as a key paradigm for connecting sensors, embedded devices and supervisory systems via lightweight communication protocols. MQTT has become one of the most widely adopted publish–subscribe protocols in IoT due to its minimal overhead, topic-based routing and flexible Quality-of-Service levels [7]. Comparative evaluations of IoT communication protocols show that MQTT provides competitive latency and energy-efficiency characteristics, particularly in scenarios with constrained devices and unreliable networks [7]. The introduction of MQTT 5.0 further extends the protocol with advanced messaging features and improved error handling, but also raises new questions regarding the impact of these mechanisms on end-to-end latency and communication efficiency, especially when combined with TLS-based security [8]. Implementation aspects of MQTT infrastructures play an important role in determining the actual performance of industrial IoT systems. Hardware-accelerated MQTT brokers and precise message-routing mechanisms have been proposed to reduce processing overhead, improve scalability and ensure predictable timing under high message loads [9]. At system level, IoT-integrated SCADA solutions for biomass power plant monitoring and digital power-grid supervision demonstrate that MQTT and related technologies can be successfully used to collect, aggregate and analyses large volumes of real-time data from heterogeneous field devices [10, 11]. These architectures highlight the potential of combining IoT communication layers with advanced analytics and decision-support tools in modern industrial environments. A recent trend concerns the deployment of intelligent IoT nodes based on microcontrollers such as the ESP32, which can act as proxy devices between sensors and cloud platforms. By integrating local processing, filtering and even AI-based inference, such nodes can reduce network traffic, improve responsiveness and facilitate edge intelligence in IoT systems [12]. Within the context of Soft-PLC and SCADA integration, these ESP32-based nodes can implement pre-processing pipelines that normalize sensor data, apply basic decision logic and encapsulate measurements into suitable message formats before transmission to higher-level controllers [12, 10]. The programmability and flexibility of Soft-PLCs are also supported by advances in engineering tools and development environments. For example, specialized ladder diagram editors for Soft-PLC systems provide user-friendly interfaces for designing control logic, automatically generating code and managing configuration data in a structured manner [13]. Combined with frameworks that describe embedded Soft-PLC architectures and CODESYS-based communication designs, these tools facilitate the deployment of Soft-PLC solutions in heterogeneous industrial networks [1, 2]. Open-source alternatives such as OpenPLC further enhance flexibility by allowing researchers and practitioners to experiment with different hardware platforms and integration patterns without being locked into proprietary ecosystems [3]. Against this background, the present work proposes a simulation-based methodology for analyzing the temporal behavior of an ESP32–MQTT–Soft-PLC chain using a combination of MATLAB and Python tools. A virtual ESP32 node generates a synthetic ADC-like signal with Gaussian noise, performs preprocessing and normalization, injects software jitter at the task level and sends timestamped JSON messages through a Python MQTT client using TLS to a cloud-hosted broker. On the receiving side, a MATLAB Soft-PLC implementation executes a cyclic scan, subscribes to the MQTT topic, decodes the JSON payload, updates a process image and evaluates a control program that includes an ON-delay timer (TON). By logging timestamps at publication, reception and scan-cycle boundaries, the framework quantifies both the end-to-end latency of the ESP32–MQTT–Soft-PLC chain and the jitter of the Soft-PLC scan time, thereby extending existing work on Soft-PLC implementation and MQTT-based industrial communication [1, 7–9]. The resulting analysis is relevant for future deployments that combine virtual modelling, open-source Soft-PLC platforms and IoT-integrated SCADA architectures in modern industrial settings [3, 10].

2. RELATED WORK

2.1. Embedded and open-source Soft-PLC architectures

Soft-PLC research has shown that software implementations of PLC functionality can successfully run on embedded platforms while preserving IEC 61131-3 semantics and the cyclic scan model used in industrial automation. Embedded Soft-PLC designs typically map the PLC runtime and communication stacks onto low-cost hardware, demonstrating that deterministic control behavior can be achieved without dedicated PLC racks when the underlying platform and configuration are carefully selected [1, 2].

Open-source initiatives provide additional flexibility by decoupling PLC functionality from proprietary ecosystems. OpenPLC, for example, offers a complete open-source PLC runtime capable of executing IEC 61131-3 control programs on commodity hardware such as Raspberry Pi, thereby enabling rapid prototyping, education and research on alternative automation architectures. Together, these contributions establish a technological foundation on which simulation-based studies, such as the Soft-PLC model proposed in this work, can be built and compared to potential hardware realizations [1, 3].

2.2. Real-time behavior, scheduling and engineering issues in Soft-PLC systems

The temporal behavior of Soft-PLC systems is strongly influenced by the characteristics of the underlying operating system and by the task-scheduling strategies that it supports. Research on real-time scheduling methods for embedded operating systems used in Soft-PLC control analyses various scheduling policies and shows how they affect scan-cycle determinism, deadline satisfaction and CPU utilization [6]. These studies highlight that, without an appropriate real-time configuration, the scan period may experience significant jitter, which can degrade control performance, especially in fast dynamic processes [5, 6].

Application-oriented work provides concrete evidence of Soft-PLC capabilities and limitations in industrial environments. The design of a yaw-control system for megawatt-scale wind turbines demonstrates that Soft-PLC controllers can be employed in safety-critical energy applications, provided that timing constraints are satisfied and the control logic is thoroughly validated. Complementary analyses of engineering problems in PLC control systems document typical challenges such as non-deterministic execution, communication delays, configuration errors and maintenance issues, underlining the need for systematic verification and performance assessment before deployment [5].

2.3. Engineering tools and programming environments for Soft-PLCs

Beyond runtime architectures, the effectiveness of Soft-PLC solutions also depends on the quality of the development tools available to engineers. A dedicated ladder-diagram editing environment implemented using table-based technology has been proposed to simplify the creation and management of Soft-PLC control programs, ensuring a structured representation of rungs, contacts and coils while enabling automatic code generation. Such tools help bridge the gap between traditional PLC engineering practices and the more flexible, software-oriented workflows associated with Soft-PLCs [2, 13].

In combination with architectural overviews of CODESYS-based Soft-PLC systems and embedded implementations, these programming environments support the systematic design of control logic, communication interfaces and monitoring functions, thereby facilitating the transition from simulation to deployable industrial solutions. The work presented in this paper adopts a similar philosophy by modelling a Soft-PLC execution cycle in MATLAB while following conventional PLC scan-cycle concepts, which aligns the simulation environment with established development practices [1, 13].

2.4. MQTT-based communication for low-latency industrial IoT

On the communication side, industrial IoT systems increasingly rely on protocols specifically designed for low-latency and resource-constrained environments. Comparative studies evaluate MQTT against other IoT protocols such as CoAP and HTTP, showing that MQTT achieves a favorable compromise between implementation complexity, bandwidth usage and latency in many application scenarios [7]. These analyses also highlight the role of transport configuration, message size and Quality-of-Service (QoS) parameters in shaping the overall communication performance.

The evolution towards MQTT 5.0 introduces additional features such as user properties, more detailed reason codes and enhanced error reporting, which have a measurable impact on communication efficiency and end-to-end delay. Evaluations of MQTT 5.0 security architectures show how TLS encryption, authentication mechanisms and advanced protocol options modify latency distributions and throughput, especially in scenarios with high message rates or constrained devices. Hardware-assisted MQTT brokers with precise message-routing logic further demonstrate that careful implementation at broker level can significantly reduce processing overhead and jitter, improving determinism for time-sensitive applications [8, 14].

2.5. IoT-integrated SCADA and digital-grid monitoring systems

MQTT and related IoT technologies have been integrated into SCADA architectures to enable open, scalable and interoperable industrial monitoring solutions. One example is an open-source IoT-integrated SCADA system for biomass power plants, where MQTT is used to collect measurements from distributed field devices and to feed visualization and alarm modules, demonstrating how IoT communication can enhance traditional SCADA capabilities [10]. Another contribution proposes a digital power-grid monitoring and intelligent analysis platform based on an IoT infrastructure, in which data from smart meters and substations are aggregated for advanced analytics and decision support [11].

These works show that IoT-enabled SCADA systems can handle heterogeneous data sources and support near real-time supervision in complex energy networks, while leveraging open protocols and scalable architectures [10]. The simulation framework proposed in this paper follows a similar philosophy at a smaller scale by combining a virtual IoT node, an MQTT broker and a Soft-PLC-like controller, thereby providing a controlled environment for exploring latency and jitter characteristics relevant to such IoT-SCADA integrations [10].

2.6. ESP32-based intelligent IoT nodes and Soft-PLC integration

At device level, the ESP32 microcontroller has emerged as a widely used platform for IoT and edge-computing applications. Recent research proposes an ESP32-based proxy device that integrates OpenAI services to enhance IoT-system efficiency, illustrating how local intelligence at the network edge can perform filtering, aggregation and transformation of sensor data before forwarding it to cloud services [12]. This approach reduces network load and can improve response times by offloading certain decision-making steps from remote servers to the edge device.

Complementing these ideas, mobile environmental-monitoring systems based on Soft-PLC demonstrate that software controllers can be deployed in combination with mobile devices for flexible data acquisition and alarm handling in industrial environments. Together, these works suggest that combining intelligent IoT nodes (such as ESP32-based devices) with Soft-PLC-style control and MQTT communication can yield powerful and adaptable architectures for modern industrial applications. This combination directly motivates the architecture investigated in the present work, where a virtual ESP32 node, an MQTT broker and a Soft-PLC-like MATLAB controller are jointly evaluated from a timing perspective [4, 7, 8, 12].

2.7. Synthesis and positioning of the present work

The reviewed literature points to three core themes that underpin this study: the architectural foundations of embedded Soft-PLC systems, the real-time behavior of Soft-PLC execution, and the timing performance of low-latency MQTT-based IoT communication. Existing embedded Soft-PLC implementations demonstrate how IEC 61131-3 controllers can be implemented as software components on general-purpose or embedded platforms, which directly supports the approach of modelling a Soft-PLC in MATLAB and reproducing a realistic scan-cycle execution. Complementary research on real-time scheduling in embedded operating systems emphasizes that scan-time jitter and deadline compliance are strongly shaped by task management strategies and system configuration, justifying the explicit measurement and evaluation of Soft-PLC scan duration in the present work. [1, 6].

From the communication perspective, comparative analyses of IoT protocols identify MQTT as a strong option for resource-constrained environments where low latency is required, while also showing that parameter tuning and network conditions can significantly affect end-to-end delay. Studies focused on MQTT 5.0 security architectures add further nuance by quantifying the overhead introduced by TLS and advanced protocol features, offering a useful basis for interpreting latency results obtained in secure, cloud-hosted deployments like the one used here. In parallel, research on ESP32-based proxy devices illustrates how edge-enabled IoT nodes can perform local preprocessing and timestamping prior to transmission, aligning closely with the concept of a virtual ESP32 node implemented in MATLAB to generate, normalize, and publish synthetic sensor data [7, 12].

Against this backdrop, the present work is situated at the intersection of embedded Soft-PLC design, real-time scheduling evaluation, and MQTT-based IoT communication, aiming to bridge conceptual architectures with measurable performance indicators in a controlled experimental setting. Specifically, it integrates a MATLAB-based Soft-PLC model that reflects established embedded implementation principles and preserves the logic of the classic PLC scan-cycle, so that execution behaviour can be analysed under realistic assumptions rather than idealised timing. Building on real-time scheduling insights, the study emphasises quantifying scan duration variability and jitter, treating these as practical indicators of determinism and deadline robustness in software-based control. In parallel, the work evaluates a secure MQTT communication path, with the latency results interpreted in the context of protocol-level comparisons and security overhead studies, acknowledging that cloud deployment, TLS, and configuration choices can meaningfully shape end-to-end delay. To connect the control and communication sides coherently, a virtual ESP32 node is designed as a controlled data-generation, preprocessing, and publishing element inspired by edge-proxy architectures, allowing consistent signal normalisation and timestamping logic to be replicated in simulation. This overall architecture enables a structured yet realistic investigation of how sensing, edge-style processing, secure MQTT transport, and Soft-PLC execution interact in an ESP32–MQTT–Soft-PLC chain, ultimately supporting a clearer understanding of the trade-offs between low latency and real-time control stability in software-defined industrial IoT scenarios [1, 6, 7, 8, 12].

Table 2.1

Summary of referenced articles and their relevance

Article	Domain	Key contribution	Relevance to this work
„Research and Implementation of Embedded Soft PLC System”	Embedded Soft-PLC	Describes the architecture and implementation of an embedded Soft-PLC system.	Provides the conceptual basis for modelling the MATLAB Soft-PLC and its execution cycle.
„Research on Real Time Scheduling Method of Embedded Operating System Based on Soft PLC Control”	Real-time scheduling for Soft-PLC	Studies real-time scheduling methods for embedded operating systems executing Soft-PLC tasks, with emphasis on scan-time determinism and deadline satisfaction.	Underpins the discussion and evaluation of Soft-PLC scan-time jitter and motivates the explicit logging and analysis of cycle-duration variations in this work.
„Developing and Comparing Protocols for Low-Latency IoT Networking ”	IoT communication protocols	Compares MQTT with alternative low-latency IoT protocols in terms of latency, efficiency and implementation complexity.	Justifies the selection of MQTT as the core communication protocol in the ESP32–MQTT–Soft-PLC chain and provides context for interpreting latency measurements.

3. SYSTEM ARCHITECTURE

This chapter presents the architecture of the ESP32–MQTT–Soft-PLC system used in the experimental framework. The system is organized as an industrial IoT chain in which a sensor-oriented edge node publishes data to an MQTT broker, while a PLC-like controller subscribes to this data and executes a cyclic control program.

Although the current implementation is fully virtual and based on MATLAB and Python, the structure is compatible with real deployments on embedded Soft-PLC platforms and physical ESP32 devices.

3.1. Overall ESP32–MQTT–Soft-PLC architecture

The overall structure of the system is illustrated in Fig. 1. The architecture is composed of three conceptual layers. At the edge, an ESP32-type node acquires sensor values and publishes them as timestamped messages. In the middle, an MQTT broker receives these messages

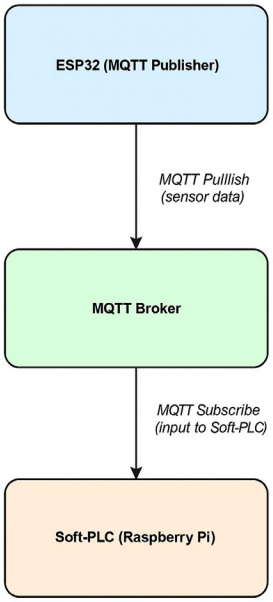

Fig. 1. Overall ESP32–MQTT–Soft-PLC Architecture.

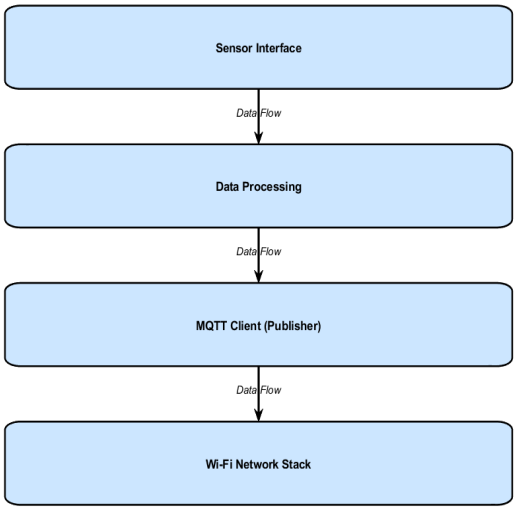

Fig. 2. Internal Structure of ESP32 MQTT Publisher.

and forwards them to interested subscribers according to the publish–subscribe paradigm. At the control level, a Soft-PLC executes a cyclic program that consumes the latest data provided by the broker and updates its internal process image and outputs.

Data generated at the edge node flows upwards towards the broker, where it is temporarily buffered and routed according to the configured topics. The Soft-PLC subscribes to the relevant topic and, at each scan cycle, retrieves the most recent message and integrates it into the control logic. Supervisory and analysis functions can be attached to the architecture by subscribing to additional topics or by processing the log files generated by the Soft-PLC. In the experimental setup, all these roles are implemented on a single host PC, but the relationships between components remain identical to those depicted in Fig. 1.

3.2. Internal structure of the ESP32 MQTT publisher

The internal organisation of the data-publishing node is detailed in Fig. 2. The figure represents the ESP32 node as a sequence of functional blocks that transform raw sensor readings into MQTT messages ready to be transmitted. The node begins with a sensor interface that acquires an analogue or digital measurement, or, in the virtual case, samples a synthetic signal generated in MATLAB.

The acquired value passes through a preprocessing and normalisation stage, where noise is filtered, offsets are corrected and the signal is scaled into a suitable range for the downstream control logic. Local decision logic can be inserted at this stage when needed, for example to implement thresholding or to derive additional features from the raw data. The result is then encapsulated by the MQTT client component, which constructs the payload, attaches timestamps and other metadata, and publishes the message to a configured topic on the broker.

The lower part of Fig. 2 emphasizes the role of the network and security stack. The ESP32 node maintains TCP/IP connectivity, manages Wi-Fi parameters and, when required, establishes a secure connection using TLS. In the virtual implementation used in this work, the same logical structure is reproduced in MATLAB and Python: MATLAB implements the sensor model, preprocessing and normalization, while Python, through the MQTT client, handles message publication and secure transport to the broker.

3.3. Functional structure of the MQTT broker

The MQTT broker is the central communication element of the architecture and its internal structure is depicted in Fig. 3. The broker accepts connections from clients, maintains their sessions, stores subscription information and routes messages between publishers and subscribers. At a functional level, the broker includes a connection and session manager that authenticates clients and tracks their state. A topic and subscription manager maintains the mapping between topics and subscribed clients, ensuring that each publish message is delivered to all interested receivers. A message queue and routing engine handle the storage and forwarding of messages, taking into account the configured Quality-of-Service level and any retained-message semantics. Security and TLS processing are also integrated into the broker, providing encrypted channels and certificate-based authentication when required. In the experimental setup, a cloud-hosted broker is used, but Fig. 3 remains valid for both cloud and on-premise implementations. This separation between publishers, broker and subscribers allows the ESP32 node and the Soft-PLC to operate independently while exchanging data via well-defined topics.

3.4. Soft-PLC scan-cycle model

The behaviour of the Soft-PLC controller is governed by a cyclic execution model, which is shown in Fig. 4. Each scan cycle starts by reading the current inputs, continues with the execution of the control program and ends with the update of the outputs and optional diagnostic information. At the beginning of the cycle, the Soft-PLC retrieves the most recent data received from the MQTT broker and updates its internal process image.

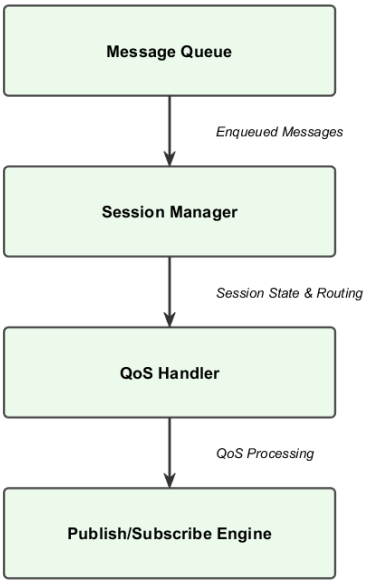

Fig. 3. Functional Structure of the MQTT Broker.

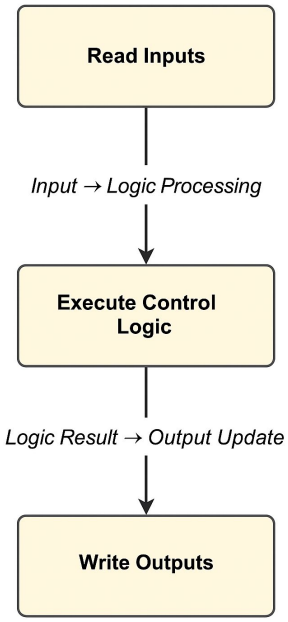

Fig. 4. Soft-PLC Scan Cycle.

The control logic then operates exclusively on this process image, following the classical PLC principle that all decisions within a cycle are based on a consistent snapshot of the inputs. The program may include timers, comparators, logic blocks and arithmetic operations. In the specific case studied in this work, an ON-delay timer is used to model a time-based decision that depends on how long a certain condition remains true.

After the program finishes, the Soft-PLC writes the computed values to its outputs. Depending on the application, these outputs could drive actuators, generate alarms, update status variables or be published again over MQTT towards other consumers. Figure 4 also suggests that housekeeping tasks, such as logging and diagnostics, can be performed near the end of each scan cycle. In the MATLAB-based implementation, the scan-cycle duration is measured at every iteration, allowing the analysis of scan-time variability and jitter.

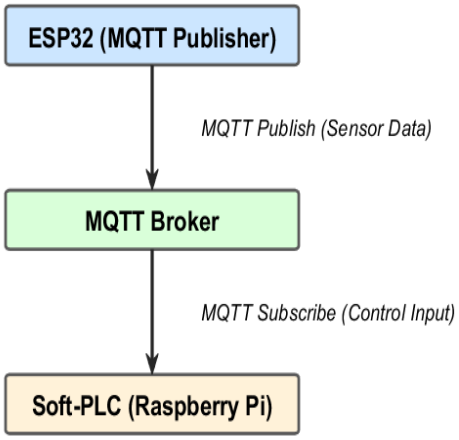

Fig. 5. ESP32 → MQTT Broker → Soft-PLC.

3.5. End-to-end communication chain

The interaction between the ESP32 node, the MQTT broker and the Soft-PLC is synthesized in Fig. 5. The diagram highlights the complete path followed by a data sample from its origin at the ESP32 to its use in the Soft-PLC control program. When a new sensor value is produced, the ESP32 node preprocesses and normalises it, assigns a publication timestamp and sends the resulting MQTT message to the broker.

The broker receives the message, places it in the appropriate internal queue and dispatches it to all clients that have subscribed to the corresponding topic. The Soft-PLC, acting as one of these subscribers, stores the most recently received message so that it can be read at the next scan cycle.

At the beginning of each cycle, the Soft-PLC reads this message, decodes the payload and updates its process image, thus closing the communication chain described in Fig. 5. From this point, the value participates in the execution of the control logic and may influence the timer or other program elements. By recording both the publication timestamp at the ESP32 node and the time of incorporation into the Soft-PLC cycle, the system makes it possible to compute the end-to-end latency for each message and to relate these values to the behaviour of the scan cycle.

This chapter therefore provides the structural and functional context for the methodology and experimental procedures described in the following sections, where the virtual implementation of each component and the detailed measurement of latency and jitter are presented.

4. METHODOLOGY

This chapter describes the methodology used to evaluate the ESP32–MQTT–Soft-PLC chain in a fully virtual environment. The goal is to obtain reproducible measurements of end-to-end latency and scan-time jitter, using a setup where the ESP32 node, the MQTT client, the Soft-PLC and the data-logging components are all implemented on the same host PC. The methodology follows the logical stages of defining objectives, designing the experimental architecture, modelling each subsystem, defining the benchmark procedure and finally computing the latency and jitter metrics from timestamped data.

4.1. Methodological objectives

The primary objective of the methodology is to construct an experimental framework that allows an integrated evaluation of a virtual ESP32 node, a secure MQTT broker and a cyclic Soft-PLC modelled in MATLAB, with explicit emphasis on the timing characteristics of the full chain from data publication at the node to processing in the controller. The approach is simulation-based and relies on widely available tools, so that the same scenario can be reproduced or extended with minimal effort.

Within this framework, the methodology aims to model a virtual ESP32 node capable of generating sensor-like data, preprocessing and normalising it, and publishing JSON messages via MQTT; to configure and use a public MQTT broker with TLS support so that the impact of security mechanisms on communication performance can be observed; to implement a cyclic Soft-PLC in MATLAB that includes an ON-delay timer and logging mechanisms; and to define and compute latency and jitter indicators based on the logs collected during the simulation.

An additional objective is that all components should be generic enough to allow later extension towards real hardware, with a physical ESP32 device and an industrial Soft-PLC running on platforms such as CODESYS or OpenPLC. In this sense, the virtual experiment serves both as a measurement tool and as a reference model for subsequent migration to an IoT-integrated SCADA architecture.

4.2. Experimental architecture

The experimental architecture is centred around three main blocks that interact through MQTT: the ESP32 node model implemented in MATLAB, the cloud-based MQTT broker and the MATLAB Soft-PLC. All components execute in a coordinated way within a single simulation environment. The ESP32 node periodically generates sensor data, preprocesses it and publishes it on a dedicated topic, while the Soft-PLC subscribes to the same topic and processes the received messages within a PLC-style scan cycle.

The global view of this setup is presented in Fig. 6. The figure shows the MATLAB host PC running both the ESP32 node model and the Soft-PLC logic, with a Python MQTT client acting as the bridge between MATLAB and the cloud MQTT broker over a TLS-secured connection. The broker sits outside the host PC and mediates all communications between publisher and subscriber.

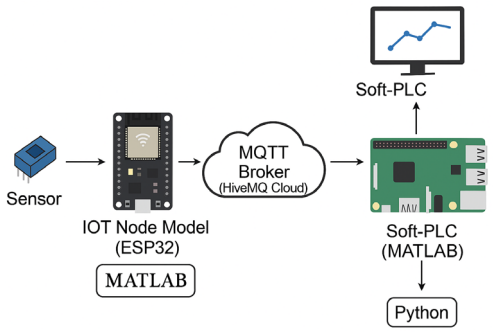

Fig. 6. Experimental Architecture with MATLAB ESP32 Node, Python MQTT Client and Soft-PLC via HiveMQ Cloud.

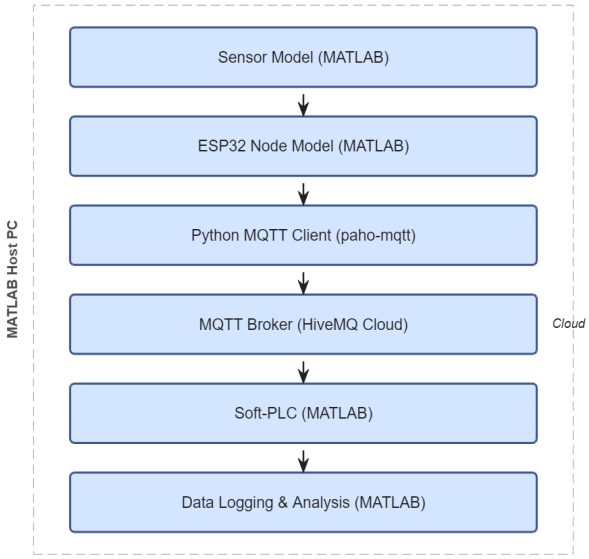

Fig. 7. Experimental Architecture.

A more abstract representation of the same architecture is given in Fig. 7. Here, the system is depicted as a vertical flow that starts with the sensor model, continues with the ESP32 node model and the Python MQTT client, passes through the MQTT broker and ends with the Soft-PLC and the data logging and analysis modules. This diagram emphasizes the logical data flow and the location of the measurement points, rather than implementation details.

From a data-flow perspective, the architecture follows the pattern of a SCADA or industrial IoT system. The virtual ESP32 node plays the role of an edge device that publishes both raw and processed data. The MQTT broker performs multiplexing and distribution of the messages. The MATLAB Soft-PLC acts as a control node that implements the automation logic, including the timer, and collects logs for later analysis. This separation of responsibilities mirrors modern industrial practice, where the controller and the communication infrastructure are designed as interconnected but clearly delimited subsystems.

4.3. Modelling the ESP32 node in MATLAB

The virtual ESP32 node is realised as a data-generation and processing pipeline within MATLAB. Its internal structure is illustrated in Fig. 8, which shows the progression from a synthetic sensor model to the final message publication. The pipeline begins with the generation of a synthetic analogue signal that evolves over time and is perturbed by Gaussian noise, in order to reproduce typical measurement variability.

The noisy signal is then subjected to preprocessing operations such as filtering and offset correction, after which it is normalized to a predefined numerical range suitable for the subsequent control logic. To emulate the non-deterministic execution that characterizes non-real-time embedded systems, a software-based jitter is introduced into the scheduling of the publishing task, so that the intervals between successive messages are not perfectly constant. At each iteration, the current sensor value, together with the derived normalized value and a publication timestamp, is packaged into a JSON

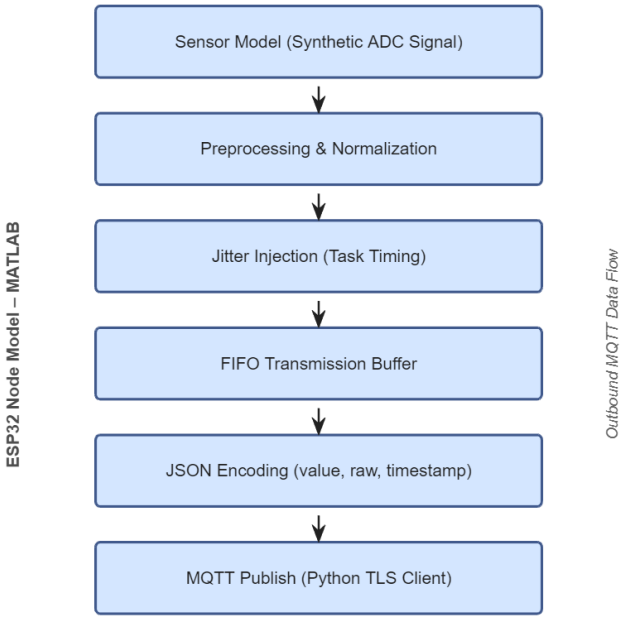

Fig. 8. ESP32 Data Generation Pipeline from Synthetic Sensor Model to MQTT Publish via Python TLS Client.

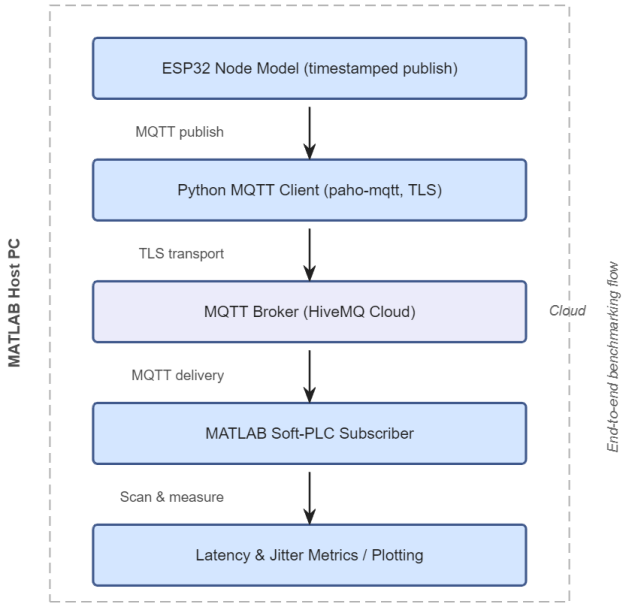

Fig. 9. Benchmarking procedure.

structure. As indicated in Fig. 8, this JSON message is passed from MATLAB to the Python MQTT client, which publishes it over a TLS-secured connection to the broker.

4.4. MQTT broker configuration

The MQTT broker is an essential part of the communication path and its configuration is therefore included explicitly in the methodology. The broker is hosted in the cloud and is accessed from the host PC using a secure URL, a dedicated TLS port and authentication credentials. The Python MQTT client is configured with the server address, the username and password, the root certificate required to validate the broker and a keep-alive interval that maintains the connection over the entire duration of the experiment.

On the application side, a single MQTT topic is used to carry the messages from the ESP32 node model to the Soft-PLC subscriber. This choice simplifies the analysis and ensures that all published messages are relevant for the benchmark (Fig. 9). The Quality-of-Service level is selected so that each message is delivered at least once, providing a balance between reliability and overhead. Although these details are not explicitly depicted in the figures, they are consistent with the architectural relationships shown in Figs. 6 and 7 and form the basis of the end-to-end latency that is later measured.

4.5. Modelling the MATLAB Soft-PLC

The MATLAB Soft-PLC is implemented as a cyclic process that mirrors the behavior of a conventional PLC. Its execution flow is depicted in Fig. 10, where the main phases of each cycle are shown: reading inputs, executing the control program, updating outputs and logging relevant data. At the beginning of every cycle, the Soft-PLC checks whether a new MQTT message has been received by the Python subscriber since the previous iteration. If a new message is available, it is decoded from JSON format and the values of interest are written into the process image representing the current inputs.

Once the process image has been updated, the control program is executed. In the present configuration, the program includes an ON-delay timer whose input is driven by the normalized sensor value provided by the ESP32 node model. The timer accumulates the time for which the input condition remains true and activates its output when the accumulated duration exceeds a specified threshold. After the logic has been executed, the Soft-PLC updates its outputs and records a timestamp marking the end of the cycle. By subtracting the timestamp at the beginning of the cycle from the timestamp at the end, the system obtains the actual scan time for that iteration. Figure 10 thus provides a compact representation of how input handling, logic execution, output update and logging are organized within each cycle.

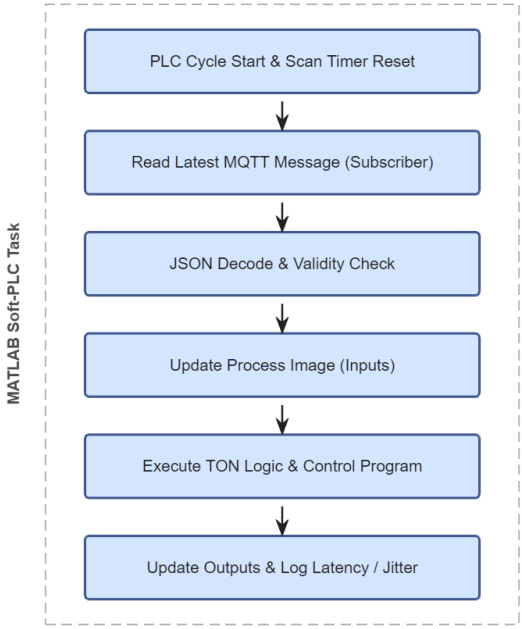

Fig. 10. MATLAB Soft-PLC Execution Cycle.

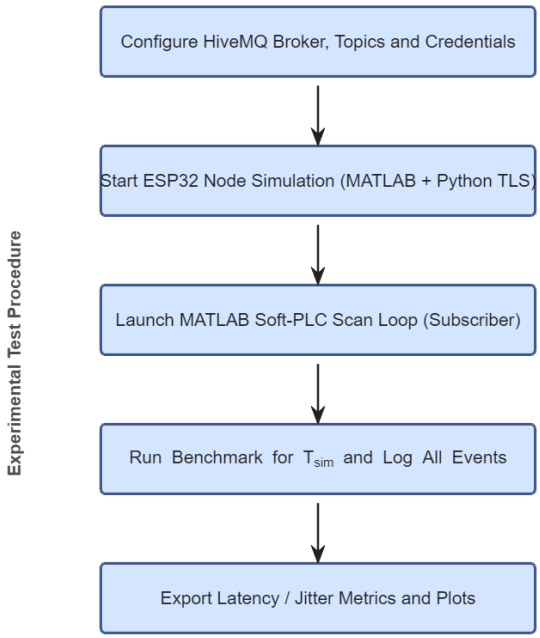

Fig. 11. Test Procedure for the Experiment.

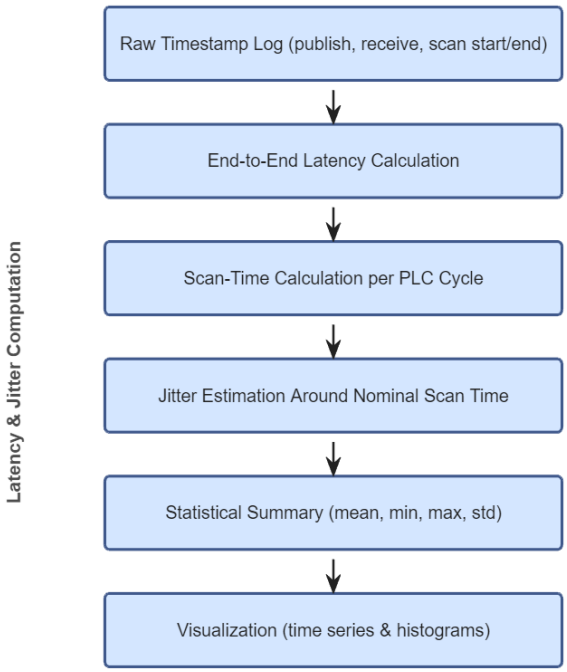

Fig. 12. Latency and Jitter Computation Pipeline.

4.6. Test procedure

The test procedure defines how the experimental framework is used to generate the data needed for latency and jitter analysis. The logical structure of the benchmark is presented in Fig. 11, where the flow from timestamped message publication at the ESP32 node through the broker to the Soft-PLC and the final logging of timing information is summarized as a sequence of operations. The ESP32 node model publishes messages containing both values and timestamps, the broker delivers them to the subscriber, and the Soft-PLC compares the publication times with its own reception times while also measuring the duration of each scan cycle.

4.7. Latency and jitter calculation

The final part of the methodology concerns the derivation of latency and jitter metrics from the raw log data produced during the experiment. The computation steps are summarized in Fig. 12, which starts from the timestamped records and ends with the statistical indicators used in the results chapter. For each received message, the end-to-end latency is calculated as the difference between the time at which the Soft-PLC processes the message and the publication timestamp inserted by the ESP32 node model. This difference is expressed in ms and stored in a vector that forms the basis for time series plots and histograms.

In parallel, the duration of every Soft-PLC scan cycle is determined by subtracting the start timestamp of the cycle from its end timestamp. The resulting values form a second vector that captures the variability of the scan time. The jitter is then interpreted as the deviation of these measured scan times from the nominal scan period chosen for the experiment. Figure 12 illustrates how these calculations are organized: raw timestamps are first transformed into per-message latencies and per-cycle durations, then aggregated into descriptive statistics and finally prepared for graphical representation. The methodology described in this chapter thus provides a complete path from the configuration of the virtual ESP32–MQTT–Soft-PLC chain to the numerical characterization of its temporal performance.

5. EXPERIMENTAL RESULTS AND DISCUSSION

This chapter presents the results obtained by running the unified ESP32–MQTT–Soft-PLC scenario in MATLAB and provides a critical analysis of the system behaviour in terms of signal quality, end-to-end latency and Soft-PLC scan-time jitter. The discussion relates the numerical indicators and plots to the expected performance of Soft-PLC-based industrial systems and to the typical requirements of IoT-enabled automation architectures.

5.1. Experimental configuration

The experiments were carried out using a single MATLAB script that integrates the virtual ESP32 node model, the secure MQTT connection to the cloud broker and the cyclic Soft-PLC. The total simulation time was set to 30 seconds, while the nominal scan period of the Soft-PLC was set to 50 ms, a value representative for medium-speed automation tasks where temporal resolution must be balanced against computational and communication overhead. The virtual ESP32 node generated at each iteration a raw sensor value modelled as a Gaussian random variable around a mean of approximately 100 ADC units and, by applying a scaling operation, a corresponding normalised value in a fixed numerical range. Each MQTT message transmitted to the broker contained both the raw and the normalised values together with the publication timestamp, all encapsulated in a JSON structure. The MATLAB Soft-PLC, subscribed to the topic used by the node, read in every scan cycle the latest available message, decoded the JSON payload, updated the process image, executed the ON-delay timer logic and recorded all time stamps necessary for the computation of latency and jitter.

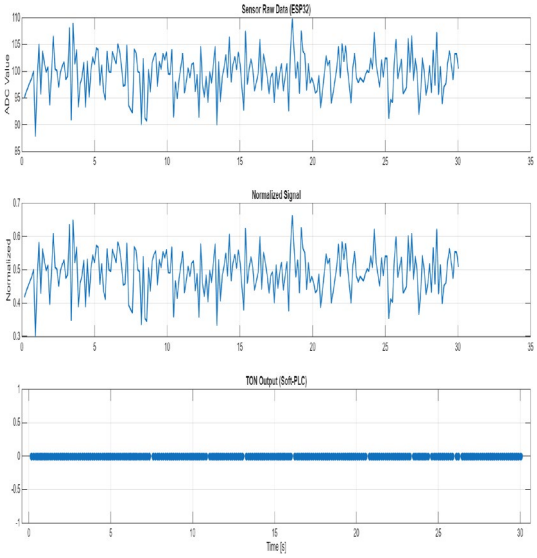

Fig. 13. Data Overview (raw sensor, normalized signal, TON output).

5.2. Sensor signal and TON timer analysis

The time evolution of the sensor signal and of the ON-delay timer output is summarized in Fig. 13, which contains three aligned subplots: the upper one shows the raw ADC values produced by the virtual sensor, the middle one shows the corresponding normalized signal and the lower one shows the binary output of the TON block. The raw signal oscillates between roughly 88 and 108 ADC units, which confirms that the sensor model reproduces a realistic combination of a stationary mean and measurement noise with a normal distribution.

The normalized signal lies in an interval of approximately 0.3 to 0.65, demonstrating that the preprocessing pipeline implemented at the ESP32 node delivers to the Soft-PLC a dimensionless quantity that can be directly used in threshold-based control logic. In the test scenario the activation threshold of the TON block was set to a normalized value of 0.5 and the delay time was set to two seconds. By inspecting the lower subplot of Fig. 13 it can be seen that the TON output remains essentially zero during the entire simulation window. This indicates that the time intervals during which the normalized signal exceeds the 0.5 threshold are too short for the timer to accumulate the required delay. The behavior is consistent with the standard definition of an ON-delay block in Soft-PLC environments, where the output switches only if the input condition remains continuously true for at least the preset time. From the point of view of implementation, this confirms that the TON logic has been correctly reproduced in MATLAB, while from an application point of view the tested scenario corresponds to a process in which no alarm or delayed switching condition is reached within the observed time window.

5.3. End-to-end latency analysis

The end-to-end latency of the ESP32–MQTT–Soft-PLC chain was computed for each message as the difference between the publication timestamp inserted by the virtual ESP32 node and the local time recorded when the Soft-PLC processed the message. The evolution of this latency over the 30-second simulation is shown in Fig. 14. The plot reveals a repetitive pattern of rising and falling values, with latency varying between approximately 100 ms and slightly above one second. This behavior reflects the combined influence of the Soft-PLC scan cycle, which determines when new data can be incorporated into the process image, and the variability introduced by MQTT transmission through the cloud broker and by the underlying operating system.

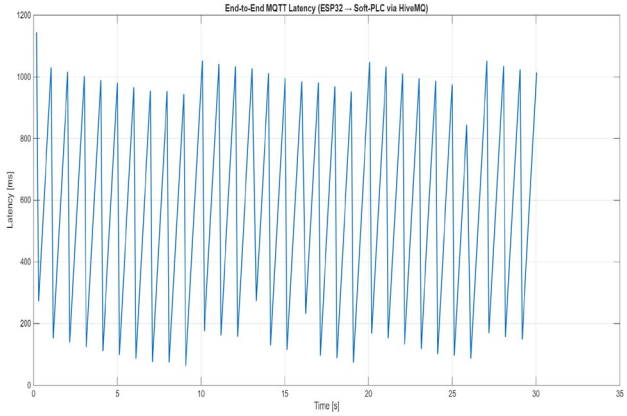

Fig. 14. End-to-End MQTT Latency over Time.

A more compact view of the same data is provided in Fig. 15, which displays the histogram of all latency samples collected during the experiment. Most values are concentrated between about 200 ms and 900 ms, while occasional peaks appear close to 1.1 s. The distribution exhibits a moderate tail towards higher values, indicating that sporadic congestion or scheduling delays can significantly increase the response time even though the average performance remains within the sub-second range. Numerically, the analysis of the latency vector yields a mean end-to-end latency of 568.06 ms, a minimum latency of 64.11 ms, a maximum latency of 1144.76 ms and a standard deviation of 289.70 ms. These results position the proposed architecture clearly in the domain of soft real-time systems: response times are adequate for monitoring and supervisory control, but they are not suitable for hard real-time tasks with strict millisecond-level deadlines.

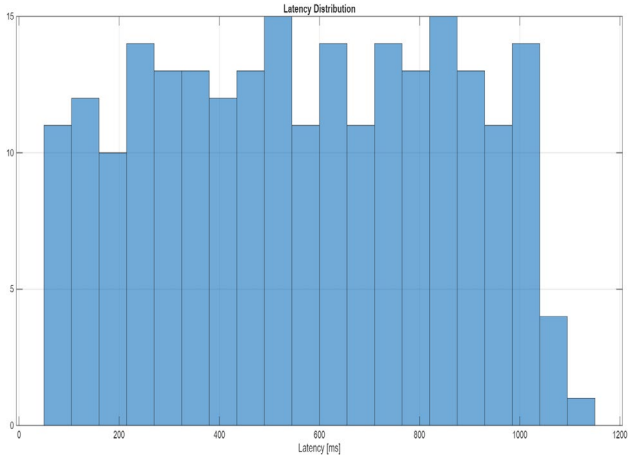

Fig. 15. Latency Distribution.

5.4. Soft-PLC scan jitter analysis

The variability of the Soft-PLC scan cycle was evaluated by measuring the effective duration of each iteration using the timing facilities available in MATLAB and storing the results in a dedicated vector. Figure 16, shows the resulting time series of cycle durations over the full 30-second experiment. Although the nominal scan period was configured to 50 ms, most measured values fall between roughly 60 ms and 70 ms, with several episodes where the duration increases up to around 170 ms, particularly at the beginning of the simulation and at a few isolated instants thereafter. The statistical indicators derived from these measurements show an average scan time of 62.64 ms and a maximum of 172.03 ms. On average, therefore, the Soft-PLC cycle is slightly longer than the nominal value, and in rare situations the scan time can exceed it by more than a factor of three. These deviations are caused by the interaction between MQTT communication (message reception and decoding), the execution of the TON control logic and the non-deterministic scheduling behavior of the operating system on which MATLAB runs. This kind of jitter is typical for software-implemented PLCs operating on general-purpose platforms without hard real-time guarantees. From the standpoint of temporal robustness, the observed variability is acceptable for industrial applications where the process evolves slowly or where the controller is placed at a supervisory level, but it would be problematic for high-dynamic closed-loop control, which demands much tighter bounds on the scan period.

5.5. Comparative discussion and industrial implications

Taken together, Figs. 13 to 16 show that the integrated ESP32–MQTT–Soft-PLC framework behaves consistently with the expectations for a soft real-time industrial IoT system. The sensor model and the preprocessing pipeline deliver a realistic input signal and a correctly functioning ON-delay timer, while the end-to-end latency remains in the order of a few hundred ms with occasional peaks around one second, and the scan-time jitter stays within a range that is compatible with applications of monitoring and supervisory control. The architecture is therefore well suited for tasks such as remote data acquisition, alarm supervision, non-critical actuation and preliminary prototyping of IoT-enabled automation solutions. At the same time, the results highlight clear limitations when the same architecture is considered for hard real-time control of fast processes. Both the latency statistics and the jitter characteristics indicate that the reaction time of the control loop cannot be guaranteed with the determinism required by motion control or high-performance drive systems. In such cases, the present framework should be viewed as a valuable analysis and prototyping tool rather than as a final deployment solution.

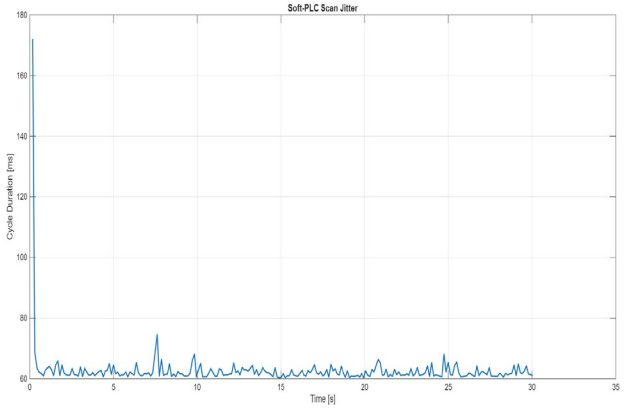

Fig. 16. Soft-PLC Scan Jitter over Time.

6. CONCLUSIONS AND FUTURE WORK

This work has proposed and analysed a unified simulation framework that links a virtual ESP32 node, a TLS-secured MQTT broker and a MATLAB-based Soft-PLC in order to study end-to-end latency and scan-time jitter in an industrial IoT context. The ESP32 node was modelled as a synthetic sensor that generates an ADC-like signal with Gaussian noise, applies preprocessing and normalisation, injects software jitter at task level and publishes timestamped JSON messages via a Python MQTT client. On the control side, a cyclic MATLAB Soft-PLC subscribed to the same topic, updated a process image at the beginning of each scan and executed an ON-delay timer, while logging all timestamps needed to compute latency and jitter. The experimental results obtained for a 30-second run with a nominal 50 ms scan period showed an average end-to-end latency of about 568 ms, with values between roughly 64 ms and 1.15 s, and an average scan time around 63 ms, with peaks close to 172 ms. These values clearly characterise the architecture as a soft real-time solution: it is adequate for monitoring, data acquisition and supervisory control, where response times of a few hundred ms are acceptable, but it is not suitable for hard real-time control tasks that require tight millisecond-level deadlines and strongly bounded jitter. The main contribution of the work lies in providing a coherent ESP32–MQTT–Soft-PLC chain entirely implemented in MATLAB, with an explicit link between communication timing and PLC-style TON logic, together with a reproducible procedure for extracting latency and jitter metrics from raw logs. At the same time, the study is constrained by the use of a general-purpose operating system without hard real-time guarantees, by a simple single-node, single-topic network configuration and by the exclusive focus on MQTT over TLS. Future work should therefore extend the framework to real Soft-PLC hardware (for example CODESYS or OpenPLC on embedded platforms), explore scenarios with multiple ESP32 nodes and higher traffic levels, and compare MQTT with alternative communication protocols. A further direction is to couple the controller with more complex process models and advanced control strategies, such as predictive or latency-compensated schemes, in order to better understand and mitigate the impact of communication-induced delays and jitter on closed-loop industrial performance.

Acknowledgments: Project was financed by “Lucian Blaga“ University of Sibiu through the research grant LBUS-IRG-2024.

REFERENCES

[1] Y. Ye, J. Wang, and L. Wang, "Research and implementation of embedded soft PLC system," *Proc. - 5th Int. Conf. Intell. Networks Intell. Syst. ICINIS 2012*, pp. 166–169, 2012, doi: 10.1109/ICINIS.2012.57.

[2] H. Song, Y. Fu, and M. Liu, "Overview of Embedded Soft PLC Based on Codesys and Communication System Design," *Proc. - 2023 8th Int. Conf. Multimed. Commun. Technol. ICMCT 2023*, pp. 27–31, 2023, doi: 10.1109/ICMCT60483.2023.00012.

[3] T. R. Alves, M. Buratto, F. M. De Souza, and T. V. Rodrigues, "OpenPLC: An open source alternative to automation," *Proc. 4th IEEE Glob. Humanit. Technol. Conf. GHTC 2014*, pp. 585–589, 2014, doi: 10.1109/GHTC.2014.6970342.

[4] F. Yan, H. Zhou, Z. Peng, M. Chen, Y. Xiong, and Z. Ren, "An industrial environmental security monitoring system in mobile device based on soft plc," *Proc. - 2020 Int. Conf. Intell. Transp. Big Data Smart City, ICITBS 2020*, pp. 711–715, 2020, doi: 10.1109/ICITBS49701.2020.00157.

[5] A. Li and S. Xie, "The design of the yaw control system for the MW generation set of wind power based on soft PLC," *2011 2nd Int. Conf. Mech. Autom. Control Eng. MACE 2011 - Proc.*, pp. 6966–6969, 2011, doi: 10.1109/MACE.2011.5988651.

[6] Y. Liu, S. Du, Y. Yang, Z. Tang, W. Quan, and S. Li, "Research on Real time Scheduling Method of Embedded Operating System Based on Soft PLC Control," *2025 10th Int. Symp. Adv. Electr. Electron. Comput. Eng.*, pp. 353–358, 2025, doi: 10.1109/isaeece66033.2025.11160147.

[7] S. Ebadinezhad, A. Y. Abdillahi, G. S. Fareed, Y. A. Tuama, J. Maombe, and I. H. Igwe, "Developing and Comparing Protocols for Low-Latency IoT Networking," *2nd Int. Conf. Mach. Learn. Auton. Syst. ICMLAS 2025 - Proc.*, pp. 1815–1821, 2025, doi: 10.1109/ICMLAS64557.2025.10967590.

[8] A.-T. Shih, H.-Y. Chien și Y.-M. Huang, *"Evaluation of the Communication Efficiency in MQTT 5.0 End-to-End Security Architecture,"* în 2024 IEEE Conference on Communications and Network Security (CNS), 2024, pp. 1–6, doi: 10.1109/CNS62487.2024.10735636.

[9] M. J. A. Baig, M. T. Iqbal, M. Jamil, and J. Khan, "Design and implementation of an open-Source IoT and blockchain-based peer-to-peer energy trading platform using ESP32-S2, Node-Red and, MQTT protocol," *Energy Reports*, vol. 7, pp. 5733–5746, 2021, doi: 10.1016/j.egyr.2021.08.190.

[10] P. Yuratsmeechan and C. Manop, "Open-Source IoT-Integrated SCADA System for Biomass Power Plant Monitoring," *2024 21st Int. Conf. Electr. Eng. Comput. Telecommun. Inf. Technol. ECTI-CON 2024*, pp. 1–7, 2024, doi: 10.1109/ECTI-CON60892.2024.10595023.

[11] M. Chen, M. Huang, J. Lai, W. Cai și H. Ling, *"Design and Implementation of Digital Power Grid Monitoring and Intelligent Analysis System Based on Internet of Things Platform,"* în 2024 International Conference on Power, Electrical Engineering, Electronics and Control (PEEEC), 2024, pp. 447–451, doi: 10.1109/PEEEC63877.2024.00088.

[12] P. Charoensawat, T. Inthasuth, A. Almahjoub și M. Saadi, *"Enhancing IoT System Efficiency through Integration with OpenAI: An ESP32-based Proxy Device Approach,"* în 2024 International Technical Conference on Circuits/Systems, Computers, and Communications (ITC-CSCC), 2024, pp. 1–5, doi: 10.1109/ITC-CSCC62988.2024.10628148.

[13] F. Hu, L. Fu, L. Liu, and G. Zhang, "A soft PLC ladder diagram edit software implemented based on table technology," *Proc. - 2008 Pacific-Asia Work. Comput. Intell. Ind. Appl. PACIIA 2008*, vol. 1, pp. 817–821, 2008, doi: 10.1109/PACIIA.2008.32.

[14] K. Yamamoto, A. Fukuhara și H. Nishi, *"Hardware Implementation of MQTT Broker and Precise Time Synchronization Using IoT Devices,"* IEEJ Transactions on Electrical and Electronic Engineering, vol. 17, pp. 209–217, 2022, doi: 10.1002/tee.23511.