International Journal of Engineering Trends and Technology
ISSN: 2231–5381 / https://doi.org/10.14445/22315381/IJETT-V73I1P110

Volume 73 Issue 1, 120-135, January 2025

Original Article

Latency Analysis of WebSocket and Industrial Protocols in Real-Time Digital Twin Integration

Mohammed Hlayel[1,2*], Hairulnizam Mahdin[2], Husaini Aza Mohd Adam[3]

[1] *Faculty of Computer Science and Information Technology, Universiti Tun Hussein Onn Malaysia, Johor, Malaysia.*
[2]*Fatima College of Health Sciences, Institute of Applied Technology, Abu Dhabi, UAE.*
[3]*Kolej Komuniti Seberang Jaya, Pulau Pinang, Permatang Pauh, Malaysia.*

[*]*Corresponding Author : mohammed.hlayel@fchs.ac.ae*

Received: 19 September 2024 Revised: 09 November 2024 Accepted: 23 December 2024 Published: 31 January 2025

Abstract - *Integrating digital twins with industrial automation, particularly in remote environments, demands efficient real-time communication to synchronize virtual models with physical devices. Latency and resource usage are critical for ensuring seamless operation, especially when considering IEC 61588 standards, which mandate response times under 100 milliseconds for real-time systems. This paper presents a comparative analysis of WebSocket as an intermediary protocol with Siemens S7, benchmarked against widely used industrial communication protocols, including MQTT, Modbus, and OPC UA. The study focuses on real-time performance and resource efficiency in remote setups, using Amazon Web Services (AWS) cloud-based Node-RED as the protocols' server. Although WebSocket is not an industrial automation standard, it demonstrated competitive latency and resource efficiency, particularly when integrated with Siemens S7 for remote operations. The results show that WebSocket with S7 outperforms other protocols like MQTT, while Modbus and OPC UA exhibit higher latency in cloud-based Node-RED environments. OPC UA's performance significantly improved after optimizing the currently used Node-RED's server sampling parameters. These findings underscore the need for further optimization of Node-RED nodes to enhance protocol handling in remote industrial settings, as performance still lags behind local configurations.*

Keywords - *Digital Twin, Modbus, MQTT, OPC UA, WebSocket, Real-Time communication.*

1. Introduction

Since the introduction of Industry 4.0, the Fourth Industrial Revolution, industrial automation has been modified and extended by integrating physical devices with counterpart physical models, which are computationally referred to as Digital Twins [1]. Digital twins are representations of real physical objects that can be monitored, analyzed, and optimized in real time in industrial processes [2]. They bridge the physical and digital worlds, facilitating predictive maintenance, performance optimization, and enhanced decision-making [3, 4]. Depending on the level and scope of the system being modeled, there are various types of digital twins, including process, product, and system twins. This study focuses on a Unity-based Digital Twin Prototype (DTP), a specific kind of system twin used in the initial automation design phases to simulate industrial processes such as automated sorting systems and PLC-controlled operations without the presence of the physical machine counterpart [5]. For smooth and effective process visualization, communication between physical devices, like Siemens S7 PLCs, and the digital twin must take place in real-time as industrial standards, ensuring the simulation mirrors real-world industrial conditions [6]. The reliance on the mature communication protocols is based on the integration between the digital twin and physical devices, e.g., Programmable Logic Controllers (PLCs), on which information transfer is based on real-time data transmission, particularly in remote applications where latency, reliability, and the management of resources play a crucial role [7-9]. Latency, which represents the phase delay between the input of a system and the output, plays a vital role in the performance of real-time communication. Additionally, consuming CPUs and memories is essential and significant concerning attaining the robustness that explains the steady-state operation of digital twins that scales without being resource-constrained [10-12]. Yet when the development of Industry 4.0 began, currently available industrial communication protocols (MQTT, Modbus, OPC UA) were mainly known to have the problem of real-time synchronization, particularly in remote environments. This gap is significant as it limits the potential applicability of digital twin systems for industrial automation, with latency and resource constraints playing a dominant role.Prevalent, standardized industrial protocols for the exchange of real-time messages between program logic PLCs and machines are MQTT (Message Queuing Telemetry Transport) [13], OPC

Unified Architecture (OPC UA) [14], S7comm (S7 Communication) [15], and Modbus [16]. Such protocols are mainly implemented in local network settings where the PLC and the associated machines are in one physical location. However, the digital twin approach expands the scope of communication beyond local environments, requiring solutions that overcome distance constraints while still maintaining real-time communication between the physical PLC and the virtual model of the digital twin [17, 18].

Although Industry 4.0 researchers have suggested several real-time communication approaches, most do not propose and experimentally validate the performance of their approach with practical latency measurements, typically using protocol specifications without taking into account the real-world nature of network environments or the conditions under which these protocols may be implemented. This deficiency highlights the critical function of the IEC 61588 standard (i.e., the Precision Time Protocol (PTP), which requires a response time of 100 ms (for real-time use) to a maximum of 200 ms for simulation use) [19]. Regarding the synchronization of time-critical systems, the IEC 61588 standard is of fundamental importance for the precision and effectiveness of industrial automation.

Timely data exchange within those constraints is fundamental to exploiting the capabilities of Industry 4.0 technologies and ensuring proper operational efficiency in distributed systems. This paper targets these problems by proposing the WebSocket with Siemens S7 as an implementable solution. Through empirical latency measurements in both on-site and cloud scenarios, this work confirms the protocol's performance according to the stringent bounds of IEC 61588, pointing to its ability to mitigate latency and minimize resource consumption even in distant installations. These results help to close the gap between abstract protocol descriptions and practical use in the real world of Industry 4.0 and provide a new solution for real-time communication in the industry of Industry 4.0.

WebSocket has become a potential solution for integrating and using industrial communication along with MQTT, allowing real-time remote data exchange [20, 21]. This study introduces the WebSocket with the Siemens S7 communication protocol (S7) approach. It assesses its suitability along with other well-known industrial protocols (e.g., MQTT, Modbus, and OPC UA), in this regard considering the efficiency that these protocols achieve for the real-time connectivity and resource management of the digital twin paradigm. Although WebSocket has promise as a lightweight transport protocol for remote control, its performance in handling high-quality, industrial standard precision real-time data exchange remains to be explored. Furthermore, the rapid rise in usage of Node-RED [22] as a scalable and flexible platform for the management of external communication protocols has been crucial in bypassing the immutable IP policy, VPN, port forwarding, or complex and risky network rework traditionally found in the use of real-time protocol. This also points to the necessity of assessing the adequacy of Node-RED for remote applications. Although the platform Node-RED provides good extensibility and scalability, the capability of its nodes to work in real-time communication, mainly when latency is a critical factor of the flow, needs careful evaluation. This study also evaluates Node-RED's performance and the resource implications of using different protocols to manage digital twin operations hosted on Amazon Web Services (AWS) [23]. In addition, this paper systematically studies latency and the resource consumption of WebSocket with S7, MQTT, Modbus, and OPC UA protocol. It mainly focuses on WebSocket's real-time performance and resource consumption in a cloud-based digital twin system. The experiment comprises a Unity-based digital twin prototype, a Siemens S7-1500 PLC emulator, and an AWS-based Node-RED, with performance metrics on latency, CPU, and memory usage across these protocols. It offers insights into the most suitable communication protocols for low-latency and resource-efficient real-time data exchange in remote industrial environments by evaluating protocol efficiency and resource management. These findings inform the choice of protocols for digital twin insertion and highlight the potential use of Node-RED nodes and WebSocket as alternative intermediate communication channels.

The following literature review provides in-depth insight into the characteristics and shortcomings of protocol-dependent features and constraints that apply to processing real-time information flows in remote digital twin systems on which this work is founded. Rocha et al. [24] conducted a computational comparison of OPC UA and MQTT between industrial plants and cloud servers. The authors determined that MQTT resulted in lower latency and scalability than OPC UA, particularly for remote applications. This study emphasized the critical role of protocol selection in achieving efficient data exchange in industrial IoT applications, especially in scenarios where low latency and scalability are paramount for remote interactions. [24]

Arda Kocamuftuoglu, Okan Akbay, and Serkan [25] reported a comparative study of some industrial communication protocols (Modbus and MQTT) in IoT environments. They also pointed to the need for protocol selection in relation to device hardware potential and communication principles, emphasizing the enduring aspect of IoT platforms. Nevertheless, their study did not focus on the particular problems of remote data transfer, which is the aim of the present work. In addition, Putpuek et al. [26] investigated the performance of OPC UA and MQTT in the ETAT Smart Lab Application, with the aim of data transfer from RS-485 Modbus sensors to the PLCnext platform. The findings showed that OPC UA outperformed MQTT regarding reduced latency when transmitting sensor data to the PLCnext platform. However, they also noted that MQTT's simpler

architecture made it more suitable for smaller-scale, resource-constrained IoT devices, underscoring the trade-offs between performance and simplicity depending on the application. Ventuneac and Gaitan [27] implemented an IIOT gateway with the Sitara AM335X processor and measured the behavior of the Modbus and the OPC UA protocols performed in an industrial context. Their results showed Modbus's better performance in local-scale applications with fast serial communication. Nevertheless, when incorporated with OPC UA, the system introduced latency, a known issue due to operating system limitations, thereby highlighting the limitations in the scalability of these protocols for remote operation. Walczak and Hetmanczyk [28] also investigated the characteristics of different IIoT protocols, such as OPC UA, MQTT, and Modbus TCP.

Their results demonstrated that although OPC UA offers a secure encryption capability and is well suited for sophisticated industrial applications, it increases network traffic more than MQTT, particularly for large data sets. On the other hand, MQTT was found to be more bandwidth-efficient and especially useful for transmitting short payloads and limited bandwidths. Fette and Melnikov [29] introduced WebSocket as a protocol for low-latency and real-time solutions for web-based applications and highlighted its potential to replace inefficient polling mechanisms in web applications. Although their study did not focus on industrial automation, we believe the protocol's design for bidirectional communication with minimal overhead makes it a strong candidate for adaptation in industrial environments as an intermediate transport mechanism, particularly for applications like remote digital twin integrations.

However, further exploration is required to determine WebSocket's performance in PLC-based and industrial settings. Despite the vast collection of industrial communication protocol research, there are still open questions about latency, particularly within the context of remote digital twin integration. Several works [24–28] have focused on the local network performance while disregarding the remote operation's scalability issues and the latency consideration. In addition, those kinds of research are often conducted by employing randomly generated data, which do not follow the inherent sequential property of the industrial real-time data flow, as the data procession is significant in PLCs' environments. On the contrary, this paper considers the simulation of a realistic DTP representing a real machine's functional behavior in the industrial process. It affords a more accurate simulation of industrial processes. Moreover, this work systematically compares the latency performance and resource consumption of MQTT, Modbus, and OPC UA in remote real-time applications against those shared by local network environments and those intermediate platforms offered by the Node-RED. The goal is to circumvent the need for a VPN, fixed IP address, port forwarding, complicated networking setup, etc. It also extends earlier research by examining the viability of using WebSocket as a middle protocol and an out-of-box replacement protocol to traditional industrial (e.g., Modbus, OPC UA, or MQTT) communication protocols, i.e., for real-time data transfer and system's effectiveness in remote digital twin applications.

2. Materials and Methods

2.1. Methodology

The experiments presented in this work are devoted to the performance evaluation of different communication protocols in real-time, integrating digital twins with Siemens PLCs, and considering environmental conditions, namely local and remote. The following process and procedure are explained in detail in the subsequent subsections. The experimental testbed consists of a digital twin application developed on Unity3D, representing the simulated version of an automated sorting system that communicates with a Siemens S7-1500 PLCSIM Advanced emulator using Node-RED as the deployed communication hub, both locally and on AWS, emulating operations remotely to enable protocol-specific routing and conversion. The hardware specifications of all these devices used in the experiment, namely Samsung Tab Active 3, central computer, AWS EC2 instance, and Siemens PLC simulator, are highlighted in detail in Table 1.

Automation commands from the digital twin are handled by WebSocket, MQTT, Modbus, and OPC UA protocols. Each of them had been implemented based on existing libraries and, where possible, tuned for the best performance under real-time conditions: WebSocket sends JSON-encoded Boolean values every constant interval of time (50 ms); MQTT adopts byte arrays with Quality of Service level 0; Modbus uses 16-bit registers to minimize useless overhead; whereas OPC UA leverages session-based read/write operations and optimized polling intervals. These protocols enabled the digital twin to have bidirectional communication with the PLC for simulating various operations: conveyor control, sensor on/off, and sorting of packages. The experiment procedure, starting from the delivery of the commands created in Unity Digital Twin, proceeds through Node-RED and is delivered to Siemens PLC, followed by the delivery back to the Digital Twin. Latency is estimated as a Round Trip Time (RTT) using high-precision timers available within Unity. Resource usage, such as CPU and memory, are measured on the system side using a Python script. Further, network performance shall be monitored using Wireshark. Additionally, performance metrics (latency, throughput, error rate) have been considered in the environment for load and network conditions). Amazon Web Services (AWS) with Node-RED Server established a direct interface between the nodes and the platform without static IP/VPN configuration. Furthermore, network throughput was analyzed using Wireshark to compare the effectiveness of each protocol. Wireshark logged network traffic during the experiment and provided insights into network load and throughput patterns.

Table 1. Hardware specifications

Device	Main Computer	Samsung T. Active3	AWS EC2 Instance	Siemens PLC
Model	Desktop	SM-T575	t3.micro	S7-1500
CPU	12th Gen Intel(R) i5, 2.5 GHz	Samsung Exynos 9810, 2.7 GHz Octa-Core	Intel(R) Platinum 8259CL, 2.50 GHz	CPU 1517TF-3 PN/DP
RAM	48 GB DDR4	4 GB	1 GB	3 MB
Storage	500 GB SSD	64 GB	8 GB	8 MB
System	Windows 11 Pro 64-bit	Android 13	Linux AMI x86_64	Firmware V2.9
WLAN/NIC	Realtek PCIe GbE, Data rate: 2.5 GB/sec	Wi-Fi, 802.11 a/b/g/n/ac, 600 Mbps	Elastic Network Adapter (ENA)	PROFINET interfaces: TCP/IP

2.2. Experimental Setup

The architecture and data flow of the local and remote setup are illustrated in Figures 1 and 2. The setup included the following components:

2.2.1. Digital Twin Model

Developed in Unity3D (version 2022.3.17f1) [30], the digital twin simulates an automated box-sorting plant, featuring processes such as piston activation, conveyor belt movement, and box detection. Unity3D was selected for its versatility in simulating complex industrial processes and its ability to support remote testing. The designed DTP communicates with the PLC using Boolean values (for binary states) as the primary data type for transmitting DTP component status. The interaction between the digital twin and the PLC, along with protocol-specific data transmission, is detailed in Figure 1.

2.2.2. Siemens S7-1500 PLCSIM

The Siemens PLCSIM Advanced [31] (PLC Simulator) was employed to simulate the PLC's behavior for controlling the processes replicated by the digital twin. Programmed via the TIA Portal, PLCSIM was chosen for its ability to emulate real-time operations of the S7-1500 accurately, allowing comprehensive testing of multiple communication protocols. PLCSIM's capability to simulate real-time industrial processes made it an effective tool for analyzing protocol performance in a controlled, repeatable environment without requiring physical hardware.

2.2.3. AWS Cloud Infrastructure

AWS provided the necessary cloud-based infrastructure for the experiment. This setup enabled testing under conditions that mimic real-world industrial scenarios where the digital twin and the PLC are not co-located, introducing latency and network reliability challenges.

2.2.4. Node-RED Server

Node-RED acted as the communication hub between the digital twin and Siemens PLC. Node-RED facilitated protocol-specific data flow, ensuring proper routing and conversion of data between the digital twin and the PLC.

2.3. System Architecture

The system architecture presented in Figure 1 aimed to replicate a typical remote industrial setup, where real-time data exchange between a digital twin and a physical PLC is required. The architecture was designed with scalability and real-world applicability, focusing on remote monitoring and control. The key architectural components included:

2.3.1. DTP on Unity Player

The digital twin prototype, running on Unity Player on a local PC, simulated an automated sorting plant. It communicated with the Siemens S7-1500 PLC through the Node-RED server hosted locally and on AWS. The digital twin visualized the process and issued commands to the PLC based on either user input or pre-programmed routines, as depicted in Figure 2.

2.3.2. Siemens S7-1500 PLC

The emulated PLC, located at a remote site, controlled physical processes based on data received from the digital twin. It communicated with the Node-RED server over the internet using the protocols under evaluation (WebSocket/S7, MQTT, Modbus, and OPC UA). The PLC processed incoming data in real-time and sent feedback to the digital twin for further actions and visualization.

2.3.3. Node-RED Servers on Local PC and AWS Cloud

Node-RED acted as the central communication hub for routing data between the DTP and the Siemens PLC, deployed on a local server (PC) and a cloud-based instance (AWS). The local Node-RED server managed low-latency communication between the digital twin and Siemens PLC, providing real-time interactions and performance benchmarking for WebSocket, MQTT, Modbus, and OPC UA in a controlled environment. This setup is suitable for direct protocol comparisons under optimal conditions before extending the tests to remote operations. Meanwhile, the AWS Node-RED server facilitated remote operations, simulating industrial

applications with variable latency, ensuring the system's ability to handle complex, long-distance data routing while maintaining scalability and reliability across different network conditions.

2.3.4. Data Flow

The data flow followed a cyclic process, where the DTP sent commands to the PLC via the Node-RED server. The PLC processed these commands and responded, and the responses were relayed back to the DTP for visualization or further action. Each protocol was tasked with managing specific data types, and the latency of these transmissions was measured and analyzed.

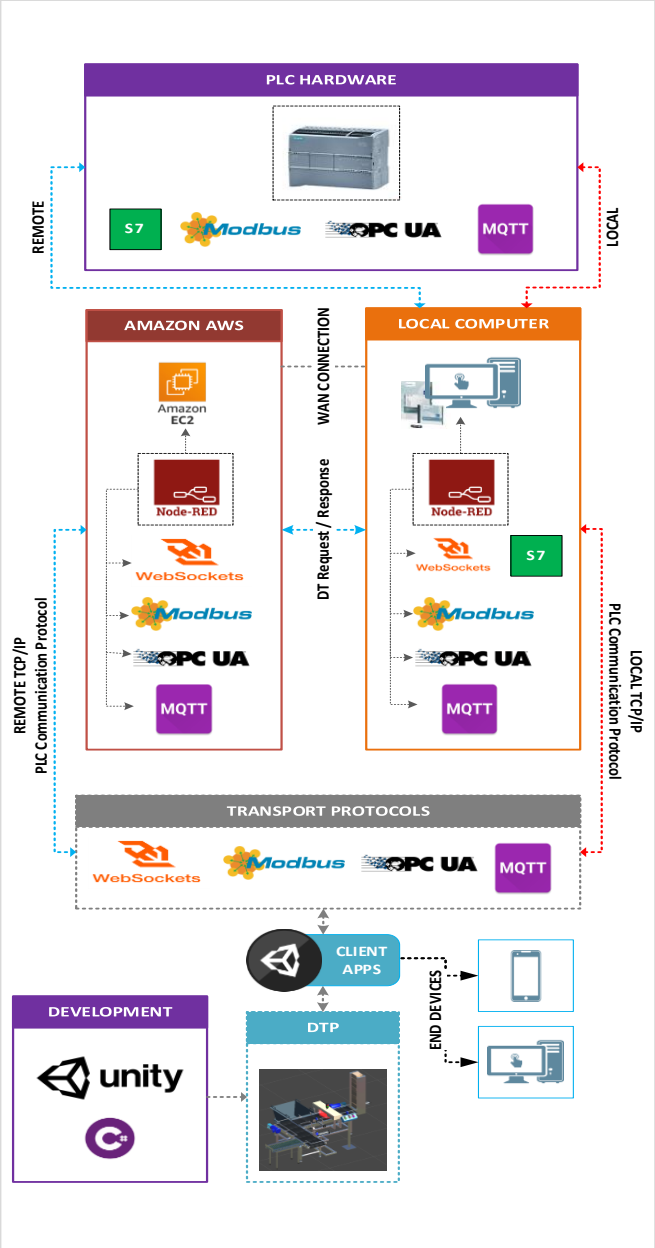

Fig. 1 Framework architecture

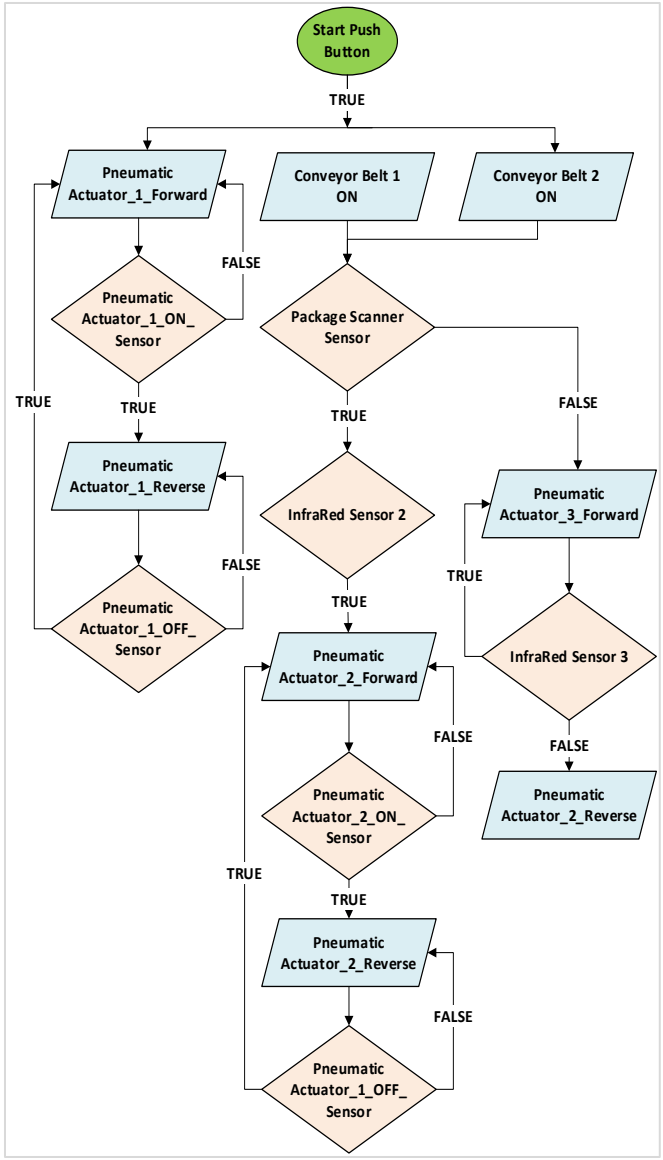

Fig. 2 PLC Logic Dataflow Chart

This experimental setup and architecture were designed to evaluate the performance of different communication protocols under realistic remote conditions, focusing on latency, resource efficiency, and protocol-specific challenges. By using scalable and flexible tools like Unity, Node-RED, and AWS, the experiment provides insights into the practical application of these protocols in industrial automation and digital twin integration.

2.4. Implementation of Communication Protocols

This study leveraged various libraries in C# and frameworks to implement and test communication protocols within a real-time, remote digital twin environment. Each I/O component communicated continuously via the respective protocol, facilitating seamless interaction between the virtual environment and the PLC controller, as detailed in Figure 2.

Using methods such as *Session.Read* and *Session.Write*, the Opc.Ua.Client library [32] was employed to manage the OPC UA connection for direct data handling. OPC UA handles each Boolean value as a separate operation, resulting in individual write actions for every data point. This approach, while precise, can increase the frequency of operations and, thus, the overall traffic, especially when dealing with multiple Boolean values. The data exchange involved Boolean values formatted according to OPC UA standards. A coroutine running at 50-millisecond intervals ensured timely updates.

Testing revealed that reducing polling intervals could burden the network and system, potentially compromising stability, while OPC subscriptions introduced higher latency. Thus, polling was confirmed as the more effective method for maintaining low latency in real-time applications. For Modbus communication, the Modbus_Client class in Unity, utilizing the EasyModbus library [33], facilitated real-time interactions with a Modbus server. Modbus efficiently packs multiple Boolean values into a single 16-bit word, minimizing overhead and optimizing network traffic. The class is connected to the PLC via a specified IP and port, executing key operations like *ReadHoldingRegisters* for data retrieval and *WriteSingleRegister* for sending commands. Data updates occurred every 50 milliseconds, balancing system performance with network traffic management. The MqttClient class of uPLibrary.Networking.M2Mqtt library [34], managed MQTT communication by initializing a MqttClient and setting up subscription and publication topics. MQTT also handles data efficiently by packing multiple Boolean values into byte arrays, which reduces the frequency of transmissions and lowers overhead. This format allows for the transmission of simple Boolean flags and more complex data structures. Periodic data publishing was driven by system state changes, using QoS 0 to ensure reliable delivery. The script also included an event handler to decode incoming MQTT messages into their original data types, updating system variables accordingly. The WebSocketSharp library [35] within the Unity framework provided a robust solution for real-time data communication, which is critical for maintaining synchronous operations between virtual models and physical systems. Unlike Modbus and MQTT settings, we configured WebSocket to send individual Boolean values encapsulated in JSON objects, which results in more frequent, smaller data packets being transmitted. While offering flexibility in handling various data types, this approach can lead to more varied traffic patterns due to the framing and sending of each message separately. WebSocket's ability to handle individual requests dynamically is useful in applications requiring frequent updates, but it may increase network traffic depending on the application's demands.

2.5. Latency and Resource Management Measurement

2.5.1. Latency Measurement

The latency measured was the RTT from when a Boolean write command is triggered from the DTP automation model and the receipt of the processed response from the Siemens PLC emulator, which includes the overall time for transmitting a command, processing by the PLC, and returning the processed response as response to the triggered input. In the interest of precision and consistency, this measurement process has been wholly automated with C# scripts integrated into the Unity3D environment. High-resolution timing was captured using the *system.Diagnostics.Stopwatch* class, starting the timer upon issuance of a command by the DTP and stopping upon receipt of the response. C# scripts enabled RTT logging for each protocol, minimizing manual intervention and errors. WebSocket commands were sent as JSON-encoded Boolean values, MQTT used byte arrays, Modbus relied on its concept of register-based interaction, and OPC UA leveraged session-based read/write operations. Results of RTT for each individual protocol for operating the DTP machine following the same procedure and PLC sequences were dynamically logged into CSV files for subsequent analysis, capturing latency under various network conditions, including local and AWS-hosted environments.

2.5.2. Resource Usage Measurement

To monitor resource utilization (CPU and memory) during protocol operations, a custom Python script was used to capture real-time performance data for critical processes: Unity (digital twin model), PLCSIM (Siemens PLC simulator), and Node-RED (communication hub). The script tracked resource usage through Process IDs (PIDs) and logged data into a CSV file for analysis. This approach enabled measurement of overall system performance and individual process load across local and cloud Node-RED instances. The data captured included:

- System-wide CPU and memory usage.
- Per-process resource usage for Unity, PLCSIM, and Node-RED.
- Each attempt's Start and end markers allow for correlation with experimental phases.

3. Results and Discussion

The experiments were conducted under controlled conditions, with each protocol tested 100 times to ensure the reliability of the results. The latency data were recorded and analyzed using statistical methods to determine each protocol's average, median, minimum, maximum, and standard deviation of round-trip time.

This analysis examines the performance of four industrial communication protocols—WebSocket/S7, Modbus, MQTT, and OPC UA—through their RTT efficiency and resource usage within a simulated environment using Unity, PLCSIM, and Node-RED applications. The results provide insights into the strengths and weaknesses of each protocol, particularly in scenarios where minimizing latency and optimizing resource consumption are essential.

3.1. Round Trip Time Analysis

The average RTT and distribution for each protocol were evaluated, as illustrated in Figure 3. WebSocket with S7 emerged as the top performer, with an average RTT of 43.8 milliseconds (ms) in local environments, making it ideal for real-time applications that require fast data exchange. Modbus, with an average RTT of 70.8 ms, is moderately efficient and suitable for scenarios where WebSocket may not be applicable.

When the setup was extended to cloud-based environments for simulating remote environments, as shown in Figure 4, the performance of these protocols shifted. WebSocket/S7 outperformed other protocols, with an average RTT of 87 milliseconds. This performance demonstrates WebSocket's ability to handle real-time communication, even in remote cloud environments. The increase in RTT from 43.8 ms in local environments to 87 ms in cloud setups is mainly attributable to the network latency time to the AWS servers, which averages around 21 milliseconds halfway.

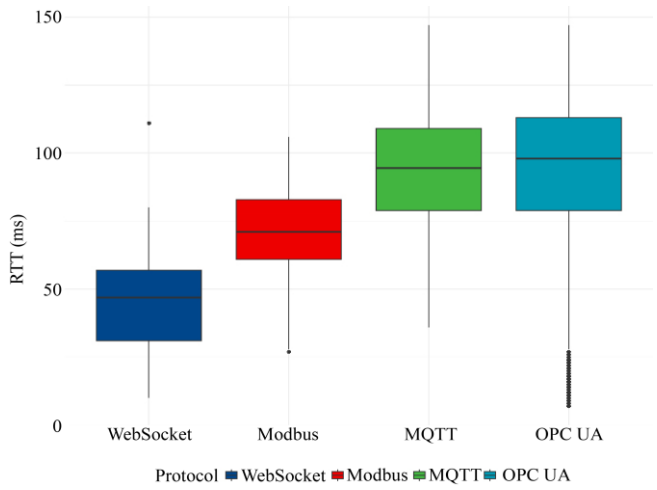

Fig. 3 Round Trip Time Distribution in Local Setup

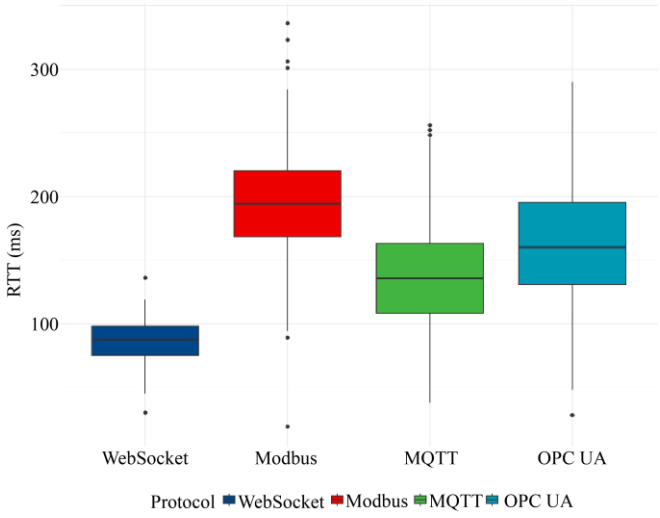

Fig. 4 Round Trip Time Distribution in Remote Setup

Given that the ping time from the PLCSIM to AWS was also measured at around 21 milliseconds, the RTT increase in the cloud setup mirrors the additional network latency, suggesting that WebSocket and S7 server nodes efficiently handle message transmission without introducing significant internal delays. The results indicate that the bulk of the RTT increase is due to the natural network latency associated with cloud-based communication. WebSocket with S7 still transmits data nearly as efficiently as the base ping time. This highlights WebSocket's ability to maintain its real-time communication properties even in remote setups, as it incurs minimal additional overhead beyond the inherent network latency.

On the other hand, Modbus experiences a significant RTT increase, averaging 194 milliseconds in cloud environments, nearly three times its local performance. This suggests Modbus is more sensitive to network latency and distance when communicating with a cloud server using Node-RED. One possible explanation for this significant RTT increase is the Node-RED Modbus server node, which may introduce additional processing overhead and delays. Unlike protocols like WebSocket optimized for lightweight, continuous data flow, Modbus was initially designed for local networks and might not perform as efficiently in cloud-based environments, mainly when relying on intermediary systems like Node-RED. Its older protocol design and reliance on a polling mechanism may not be optimized for high-latency networks, making it less suitable for real-time, cloud-based applications.

MQTT and OPC UA exhibited similar RTTs of 93-94 milliseconds locally, with MQTT's RTT increasing to around 133 milliseconds in cloud environments. This 40 ms increase is comparable to the increase observed with WebSocket, suggesting that the MQTT Node-RED server nodes manage network latency effectively. Despite the additional overhead introduced by cloud communication, MQTT's performance remains stable, indicating its suitability for remote real-time applications. However, OPC UA experiences a more considerable RTT increase, rising to 162 milliseconds in cloud environments. This higher RTT can be attributed to OPC UA's more complex data encapsulation processes, which likely increase its sensitivity to network latency.

Additionally, Node-RED's OPC UA server nodes may introduce further latency due to their handling of sampling intervals and data transmission, particularly in cloud environments where network conditions are less stable. These factors make OPC UA less efficient for real-time communication in Node-Red cloud-based setups than protocols like MQTT and WebSocket. The variability in RTTs, depicted in Figure 3 and further detailed in the cloud comparison (Figure 4), emphasizes WebSocket's consistency, as indicated by its narrower box plot. This suggests that WebSocket provides more predictable performance locally and over the cloud.

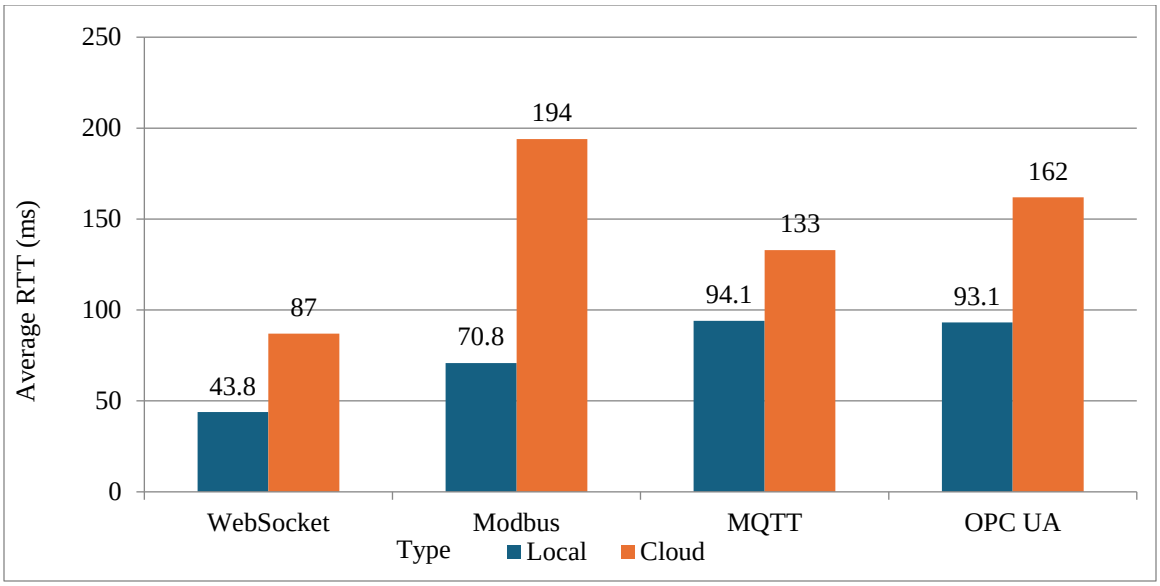

Fig. 5 Comparison of average RTT in local and remote setup

Table 2. Comparative Summary of Communication Protocols in Local and Cloud Setups

Protocol	Average RTT (ms)		Median RTT (ms)		Min RTT (ms)		Max RTT (ms)		Standard deviation (ms)		Significance (ANOVA)
	Local	Cloud	Local	Cloud	Local	Cloud	Local	Cloud	Local	Cloud	
WebSocket	43.8	87.0	47.0	88.0	10.0	30.0	111.0	136.0	16.2	16.0	***
MQTT	94.1	133.0	94.5	130.0	36.0	38.0	147.0	256.0	18.0	38.0	***
Modbus	70.8	194.0	71.0	194.0	27.0	19.0	106.0	336.0	15.4	42.0	***
OPC UA	93.1	162.0	98.0	159.0	7.0	28.0	147.0	290.0	28.6	35.0	***

In contrast, the more significant variability in RTTs observed with OPC UA, and to a lesser extent with MQTT, particularly in cloud environments, indicates that these protocols may experience more fluctuations in performance, likely due to their sensitivity to network conditions and data handling processes. While Figure 5 provides an overview of the communication performance of the various protocols in local and remote setups, Table 2 summarizes the vital statistical metrics, including the mean, median, standard deviation, minimum, and maximum round trip times for each protocol.

The table highlights the differences in RTTs between local and cloud environments for each protocol. An ANOVA test was conducted to assess the statistical significance of the differences between the protocols, with the results showing a highly significant difference ($p < 0.001$) across all protocols, denoted by three asterisks (***). This significance indicates that the differences in RTTs across the protocols and between local and cloud environments are not due to random variation but reflect meaningful distinctions in protocol performance.

3.2. CPU Usage Analysis

The CPU usage captured data across different protocols was analyzed, with the results divided by the 12 available CPU cores. This accurately assesses the CPU demands per protocol, as shown in Figures 6, 7, and 8. The accompanying Table 3 further details the statistical metrics, including mean, median, and standard deviation of CPU usage across protocols, providing a more granular understanding of the CPU demands associated with each protocol. The data highlights the significant differences in CPU requirements, with WebSocket being more efficient in some scenarios but still demanding in others. Modbus and OPC UA show similar trends, varying efficiency depending on the application, while MQTT consistently demands more processing power, particularly in Unity.

Figure 6 highlights the average CPU usage and the variability across different protocols. MQTT emerges as the most CPU-intensive protocol in Unity, with an average usage of 4.65%, closely followed by WebSocket at 4.36%. In contrast, OPC UA demonstrates the most efficient performance in Unity, with the lowest average CPU usage of 2.06%, making it a strong candidate for CPU-constrained environments, followed by Modbus with a moderate usage level. In the PLCSIM environment, although the difference in CPU consumption between protocols is slight, WebSocket/S7 (S7 only communicates with PLCSIM) consumes the most CPU resources, with an average usage of 2.71%, indicating higher resource demands in this specific setting. Modbus and OPC UA follow closely, with averages of 2.20% and 2.32%,

respectively. Interestingly, MQTT shows lower CPU usage in PLCSIM, averaging 1.75%, compared to its higher demands in Unity. For Node-RED, WebSocket/S7 is the most efficient protocol, requiring the least CPU, with an average usage of just 0.09%, making it ideal for lightweight applications. Modbus and OPC UA also perform relatively higher in this environment, with averages of 0.35% and 0.27%, respectively. Although MQTT is also relatively efficient in Node-RED, it shows higher variability, suggesting less consistent performance. Figure 7 illustrates the trends in CPU usage across multiple trials. WebSocket and OPC UA exhibit consistent CPU usage patterns, particularly in Unity and Node-RED applications, indicating stable performance over time. In contrast, Modbus and MQTT display more significant variability, with noticeable fluctuations in CPU usage, particularly in Unity. This suggests potential inefficiencies or inconsistent processing demands for these protocols. Figure 8 compares the average CPU usage across all protocols, reinforcing that WebSocket is highly efficient in Node-RED but demands slightly more processing power in Unity and PLCSIM.

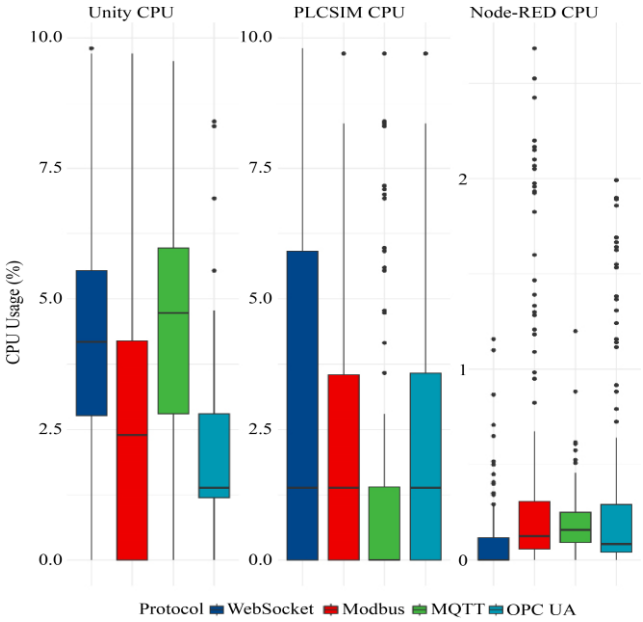

Fig. 6 CPU usage distribution

Table 2. Statistical metrics of CPU and memory usage across unity, PLCSIM, and Node-RED

Protocol	Unity_CPU ***					PLCSIM_CPU **					NodeRED_CPU ***				
	Mean	Median	SD	Min	Max	Mean	Median	SD	Min	Max	Mean	Median	SD	Min	Max
MQTT	4.65	4.73	2.14	1.18	9.56	1.75	1.38	2.68	1.18	9.70	0.18	0.16	0.15	0.03	1.20
Modbus	2.70	2.39	2.41	1.18	9.70	2.20	1.38	2.55	1.18	9.70	0.35	0.13	0.57	0.03	2.68
OPC UA	2.06	1.38	1.67	1.18	8.40	2.32	1.38	2.68	1.18	9.70	0.27	0.08	0.44	0.04	1.99
WebSocket	4.36	4.18	2.02	1.18	9.80	2.71	1.38	3.00	1.18	9.80	0.09	0.00	0.19	0.03	1.16
	Unity_Memory ***					PLCSIM_Memory ***					NodeRED_Memory ***				
MQTT	9.37	9.39	0.06	9.21	9.45	1.48	1.52	0.12	1.08	1.52	1.09	1.09	0.01	1.07	1.12
Modbus	8.51	8.22	0.49	7.91	9.95	0.60	0.11	0.67	0.10	1.51	1.13	1.11	0.05	1.04	1.30
OPC UA	9.37	9.41	0.15	8.87	9.59	1.46	1.53	0.54	0.23	2.87	1.16	1.16	0.02	1.10	1.18
WebSocket	8.48	8.51	0.16	8.10	8.86	0.12	0.09	0.19	0.09	1.51	1.05	1.05	0.01	1.04	1.06

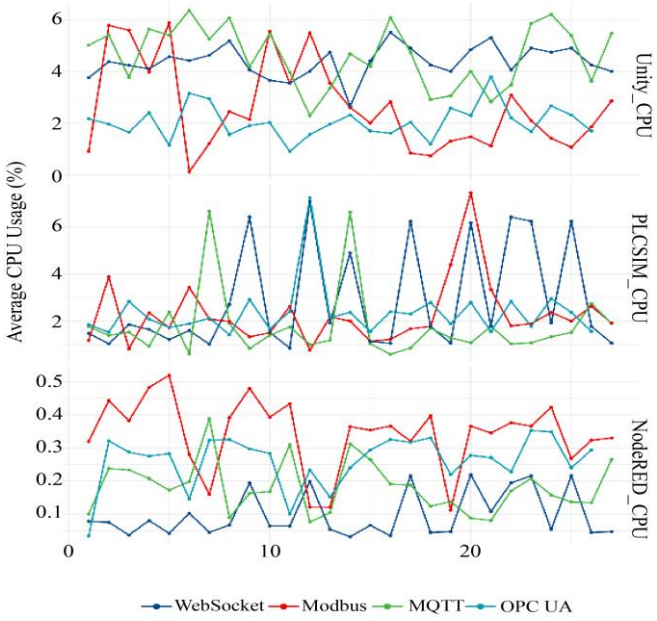

Fig. 7 Average CPU Usage Trend

MQTT demonstrates the highest CPU usage in Unity, eflecting its processing demands, while OPC UA maintains the lowest consumption across environments. Modbus, on the other hand, offers balanced performance across all applications but exhibits some variability. In summary, protocol selection should consider not just average CPU usage but also the consistency of performance across different environments. WebSocket excels in lightweight scenarios such as Node-RED, while OPC UA proves to be more suitable for CPU-constrained environments like Unity. Depending on the application, MQTT and Modbus show higher variability in specific environments, potentially affecting overall system stability.

3.3. Memory Usage Analysis

Regarding memory usage (Figure 9 and Table 3), WebSocket exhibits the lowest and most stable memory consumption, making it the most efficient protocol in this context. Modbus shows higher and more consistent memory usage around 9 MB, reflecting reliable yet greater consumption.

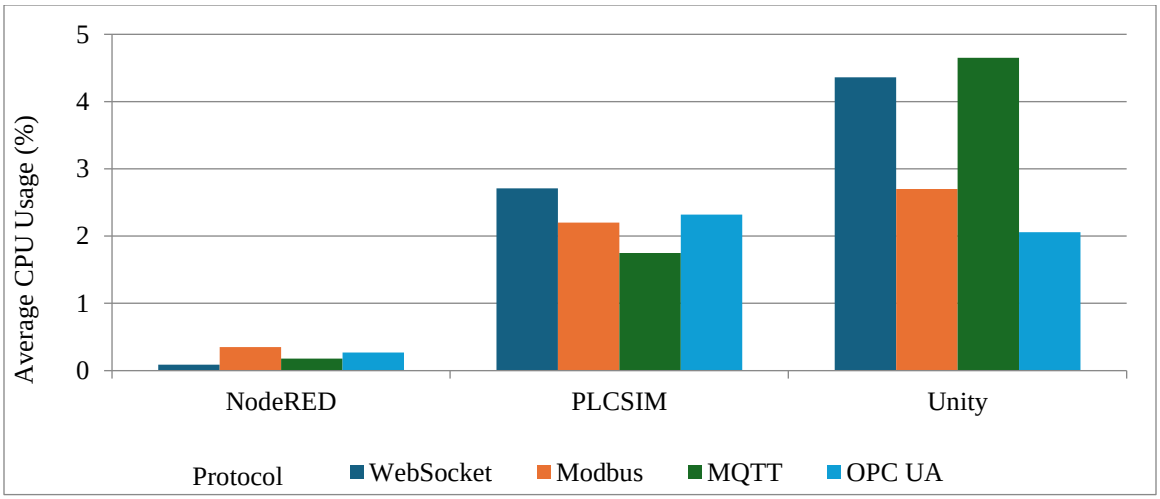

Fig. 8 Comparison of CPU usage average

MQTT and OPC UA demonstrate more variable memory usage, with OPC UA showing a higher average than MQTT, indicating that both protocols have less predictable performance.

In the PLCSIM environment, WebSocket continues to be the most memory-efficient protocol, with the lowest average memory usage. Modbus and MQTT show moderate and consistent memory consumption, making them reliable options for applications where predictable resource usage is critical. However, OPC UA exhibits higher variability while slightly lower in average memory consumption, which could lead to occasional memory spikes, making it potentially less stable in resource demands.

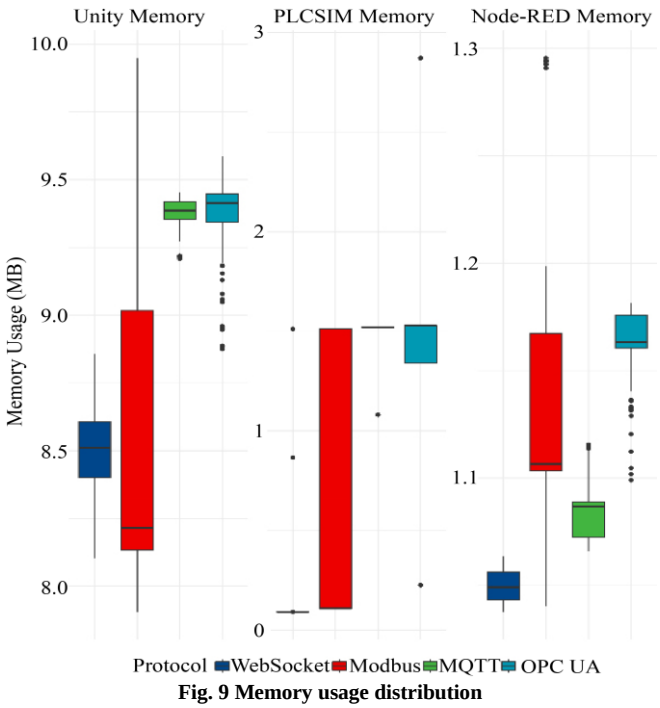

Fig. 9 Memory usage distribution

In Node-RED, WebSocket remains the most memory-efficient protocol, with the lowest and most consistent usage. This reinforces its suitability for lightweight and resource-constrained environments. Modbus and MQTT consume slightly more memory but maintain relative consistency, ensuring predictable resource usage. However, OPC UA exhibits the most variability in memory consumption again, suggesting it may introduce unpredictability in such applications, which could be problematic for environments requiring stable memory performance.

Figure 10 summarizes the memory consumption patterns for all protocols across different environments. WebSocket consistently demonstrates the lowest memory usage across all applications, highlighting its efficiency in these contexts. Modbus exhibits slightly higher memory usage but remains relatively consistent, followed by MQTT.

At the same time, OPC UA shows more significant variability, particularly in Node-RED, suggesting potential fluctuations in its memory demands. In Unity, all protocols consume significantly more memory, ranging from 8.5 to 9.4 MB, with more minor differences in their performance. Overall, the image reinforces WebSocket's efficiency, especially in resource-constrained environments, with Modbus and MQTT being reliable options for more predictable memory usage. At the same time, OPC UA may introduce some unpredictability in memory-heavy scenarios.

3.4. Correlation between RTT and Resources Usage

The correlation analysis in Table 4 reveals distinct relationships between RRT and resource usage across different protocols. PLCSIM shows a strong negative correlation (-0.8) with CPU usage, indicating that lower latency is related significantly to higher CPU utilization in this environment. In contrast, NodeRED exhibits a moderate positive correlation (0.5) with CPU usage, suggesting that higher CPU loads might increase latency due to overhead.

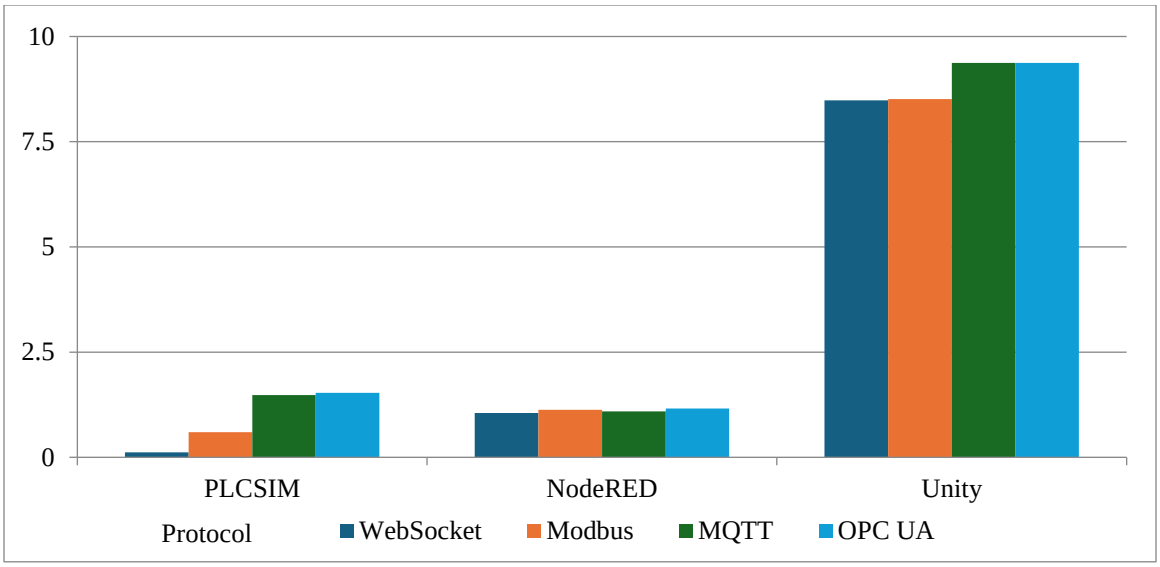

Fig. 10 Comparison of average memory consumption

Table 4. Correlation between Round Trip Time and Resource Usage

	TERMS	MQTT	Modbus	OPC UA	WebSocket	Correlation
CPU	Unity	55.8	32.3	24.7	52.3	-0.3
	PLCSIM	21	26.4	27.9	32.5	-0.8
	NodeRED	2.2	4.2	3.2	1.1	0.5
Memory	Unity	9.4	8.5	9.4	8.5	0.9
	PLCSIM	1.48	0.6	1.46	0.12	1
	NodeRED	1.09	1.13	1.16	1.05	0.7
RRT (ms)		94.1	70.8	93.1	43.8	

Memory usage consistently shows strong positive correlations with round-trip time, particularly in PLCSIM (1.0), implying that increased memory demands generally lead to longer round-trip times. Overall, protocols like WebSocket, which efficiently manage CPU and memory resources, appear better suited for low latency scenarios.

3.5. Bandwidth Analysis

Wireshark was used to capture and interpret network traffic to assess the bandwidth efficiency of WebSocket, MQTT, Modbus, and OPC UA protocols. The analysis focused on throughput, total data size, and packet distribution metrics. Results are shown in Table 5 and in Figures 11 and 12. WebSocket exhibited the lowest average throughput of 1159 bytes per second, with a total data transmission of 34 KB and a consistent packet size variability of 46%. As shown in Figure 11, this performance is contributed by the following advantages of WebSocket's lightweight JSON-based format and its friendliness to resource-constrained environments. The actual distribution of transmitted bytes (shown in Figure 12) also highlights the robustness of WebSocket in fluctuating environments. Modbus achieved moderate throughput of 4023 bytes per second, balancing a total size of 118 KB with minimal variability (28%) in packet sizes. Its small 16-bit word size resulted in a mean packet size of 3450 bytes and less variability than other protocols (see Table 5).

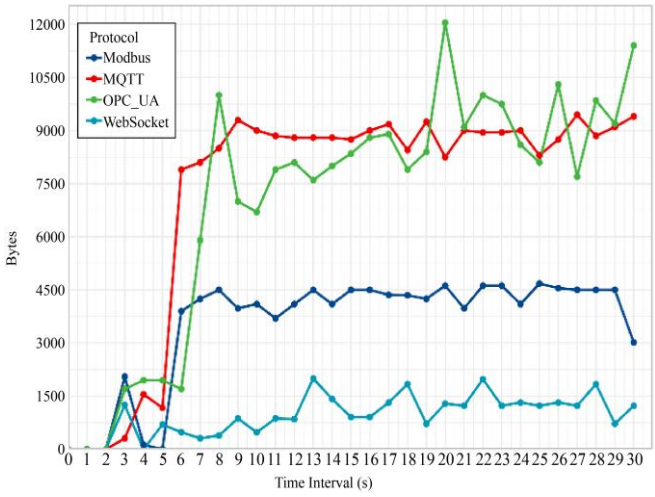

Fig. 11 Bandwidth usage trends across protocols over time

Table 5. Comparison of bandwidth usage, throughput, packet size, and variability for WebSocket, MQTT, Modbus, and OPC UA protocols

Protocol	Relative Bandwidth Usage	Total Size [KB]	Throughput [Bytes/s]	Average Packet Size [Bytes]	Min to Max Range [Bytes]	Variability [S.D. % of Mean]
WebSocket	Lowest (6%)	34	1159	975	300–1980	46%
Modbus	Moderate (20%)	118	4023	3450	125–4720	28%
OPC UA	High (33%)	215	7333	6780	560–12100	44%
MQTT	Highest (41%)	228	7782	7100	310–9450	38%

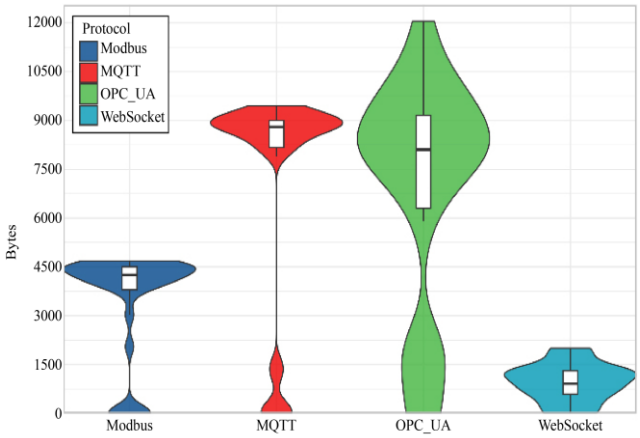

Fig. 12 Packet size distribution across protocols

This results in Modbus being appropriate for applications with small datasets and assured operation. OPC UA demonstrated higher throughput, at 7333 bytes per second, and bandwidth consumption (215 KB), with a wide range of packet sizes and variability at 44%, meaning it is prone to network fluctuations. All these capabilities visible in Figures 11 and 12 demonstrate the potential for demanding applications but highlight the need for an a-priori bandwidth-before optimization for bandwidth-constrained environments. MQTT recorded the highest throughput, at 7782 bytes per second, with a total bandwidth usage of 228 KB and moderate variability (38%). While its high throughput is advantageous for large data transfers, it necessitates careful traffic management in high-concurrency conditions to maintain steady-state performance. These findings highlight the cost of compromise between the efficiency of the protocol and scalability. The WebSocket is suitable for low-bandwidth continuous applications, while Modbus has accommodated efficiency and robustness with suitable data volume. MQTT and OPC UA have a higher throughput but are demanding good strategies for congestion management because of their volatility. The results highlight the necessity of adopting protocol selection according to individual application needs (e.g., network conditions, data volume, latency tolerance).

3.6. Discussion

A key observation was that OPC UA and Modbus performed exceptionally well in a direct local network setup without Node-RED implementation, where the PLC acted as both the server and Unity as the client. Both protocols achieved low latency in this configuration, with RTT averaging less than 5 milliseconds. This demonstrates their effectiveness in local environments with minimal overhead and direct communication, making them highly suitable for time-sensitive industrial applications. However, when Node-RED was introduced as a server to mediate communication between the DTP and the PLC, the latency for both protocols increased significantly. OPC UA server node in Node-RED, in particular, showed initial round-trip times exceeding 500 ms on several occasions during the experiments. This prompted further investigation, identifying the *minimumSamplingInterval* parameter in the OPC UA server node (104-opcuaserver.js) as the root cause. This parameter, set at 500 ms, determines how frequently the server samples variable values, directly affecting data transmission latency.

To address this, the library script was modified to adjust the minimumSamplingInterval to 50 ms, allowing for more frequent updates and significantly improving OPC UA communication's responsiveness. After the modification, the round-trip time was reduced considerably, bringing it within acceptable limits for real-time industrial applications. This finding underscores the importance of optimizing and tuning protocol parameters to meet the system's specific requirements. Flexibility in protocol implementations is critical to ensure that latency is minimized without sacrificing other performance characteristics, such as resource efficiency or data integrity.

Modbus, while not affected by sampling intervals in the same way as OPC UA, also experienced higher latencies when using Node-RED compared to the local setup. This increase in latency is likely due to the additional overhead introduced by Node-RED as a mediator, which processes and routes the communication. While Modbus remains reliable and robust, further optimization of Node-RED nodes could reduce this overhead and bring the performance closer to what was observed in the direct PLC-client communication scenario. MQTT, on the other hand, showed a different set of challenges. When attempting to achieve communication latencies below 100 ms, limitations in the native clock memory of the Siemens PLC became evident.

The native clock runs at a maximum frequency of 10 Hz, making it challenging to achieve higher publishing and subscribing frequencies using typical configurations. A TOD (Time of Day) timer and Flip-Flop logic were used to drive more frequent updates to overcome this limitation. However,

this approach introduced instability in the communication, as Siemens PLCs do not natively support MQTT compared to other protocol setups. This limitation highlights the inherent challenges of integrating MQTT with Siemens hardware, especially in real-time applications where stability and timing precision are crucial. In summary, the performance of OPC UA and Modbus in a direct local setup was exemplary, with latencies under 5 ms, making them ideal for applications where high-speed, low-latency communication is essential. However, significant latency increases were observed when Node-RED was used as an intermediary server, particularly for OPC UA, which required parameter adjustments to meet real-time performance requirements. The study shows the importance of optimizing Node-RED server nodes, especially for protocols like OPC UA and Modbus, to minimize round-trip times and achieve better performance in remote environments. Additionally, MQTT's limitations in Siemens PLCs due to the native clock constraints suggest that, while MQTT is a robust protocol for IoT applications, its integration with specific industrial hardware may require advanced configurations or alternative approaches to achieve low-latency, stable communication. Further research should explore ways to improve the native support of MQTT on Siemens PLCs or investigate more stable alternatives for high-frequency communication.

3.7. Optimization Strategies for Cloud-Based Protocols

Performance optimization in cloud environments is critical in communication protocols that can be applied effectively in real-time industrial systems. This work compared WebSocket, MQTT, Modbus, and OPC UA, introducing their respective features while discussing optimization strategies toward cloud-based scenarios with improved efficiency. These strategies drive practitioners in the field toward better implementations. With WebSocket, this is achieved by reducing the size of JSON payloads. Combining more variables into one transmission reduces overhead and increases throughput. Additionally, binary encoding reduces the transmission size and processing time compared to plain text. Thus, WebSocket is more suitable for applications that demand faster real-time updates.

MQTT provides a balance between reliability and efficiency. QoS can optimize its performance by tuning. Quality of Service 0 reduces latency and preserves resources for non-critical data, while Quality of Service 1 or 2 helps ensure the delivery of critical information. Retained messages during network interruptions enable data continuity, while lightweight topic hierarchies optimize bandwidth usage. While Modbus is considered a straightforward and resource-efficient industrial protocol, reducing duplicate searches through efficient register mapping can be further optimized. Modbus collects more data points into a single request by decreasing polling frequency, which lowers network use and, in turn, the server load. Although less scalable for extensive systems, it works well in contexts with limited resources.

Dynamic adjustment of the sampling interval according to current network conditions will improve performance in an OPC UA deployment. Subscriptions instead of polling reduce unnecessary data exchanges, thus optimizing resources. Instead of transmitting data in XML, binary encoding significantly increases speeds and reduces payload sizes, making OPC UA suitable for handling structured data across large-scale industrial installations. In cloud integration, with Node-RED servers hosted on AWS, mainly using region-specific EC2 instances, latency is reduced since the distance between devices and the cloud is minimized.

Another optimization mechanism is the consideration of elastic scaling for dynamic allocation of AWS resources based on network demand, mainly when high traffic is generated by the interaction of multiple users simultaneously with the Node-Red server. Meanwhile, it reaches system sustainability by automatically scaling up instances to sustain throughput in case of service degradation. Scaling down in low demand keeps resources conserved and reduces operational expenses.

AWS features such as Elastic Load Balancing (ELB) and auto-recovery options enhance fault tolerance and ensure continuous availability of services, even in the event of hardware or software failures. Additionally, leveraging AWS's edge services like CloudFront or local zones can bring processing closer to the point of data generation, reducing latency further and enhancing real-time responsiveness.

3.8. Scalability, Challenges, and Limitations

3.8.1. Scalability of Protocols

In industrial systems with several digital twins, communication protocols must be scalable. With an increasing number of digital twins, the protocols face more stringent demands from latency to resource usage.

While WebSocket is well suited for providing low-latency communication, it may experience performance bottlenecks as the number of connections increases. Solution strategies like payload aggregation and message batching may alleviate those issues; keeping many connections open will require system resources.

MQTT shows good scalability because of the broker-based architecture, which eases communication management even in multi-digital twin setups. However, high publishing rates and subscriptions demand more resources from the broker, affecting latency and reliability. Modbus's synchronous request-response model is less scalable and probably better suited for smaller-scale implementations. OPC UA is inherently scalable due to its structured way of handling data; however, a lot of overhead is introduced in large-scale setups. Satisfying performance in integrating many devices with high-frequency updates can only be achieved through dynamic optimization of the sampling intervals and server resources.

3.8.2. Challenges in Cloud-Based Deployments

These latency and scalability issues become prominent in a cloud environment while deploying these protocols. Latency increases because of the physical distance between devices and the cloud servers, network congestion, and routing overhead. Workloads running as region-specific EC2 instances can reduce latency, as it would be closer to the client populations; however, cross-region data transfers and cloud-to-on-premises communication can exacerbate delay. Elastic scaling ensures dynamic resource allocation to handle variable loads but incurs additional costs, especially in traffic surges. In industrial-scale cloud deployments, cost is a significant concern. While this study has used the free-tier EC2 instances of AWS for testing, scaling up industrial systems to handle greater loads and multiple digital twins would incur heavy expenses in terms of higher-performance cases, data transfers, and storage. The costs can be contained by using reserved instances for long deployments, planning for reduced data transfer volumes, and using edge computing services like AWS CloudFront. The use of AWS services, such as Elastic Load Balancer (ELB) and auto-recovery mechanisms, could improve reliability by working to ensure continuity during failures or peak usage. However, relying on these services requires critical planning to avoid extra costs while ensuring system performance in real-time applications.

3.8.3. Study Limitations

Although this study has achieved optimal round-trip times closely matching actual machine processes, several fundamental limitations should be noted. All experiments were conducted in a strictly controlled environment using only Boolean variables; in this way, it was not extended to other variable types and sizes. That limits the potential applicability of the results to real industrial scenarios where the communication involves more diverse and heterogeneous data types. Another limitation not considered in this work is multiuser scenarios or an integration of multiple digital twins in parallel. The scalability of the protocols under such conditions remains untested; therefore, further research is necessary to judge their performance under high-concurrency user environments with heavy data loads. Thirdly, the scope of the study was constrained to specific protocols, signalling that its findings might not hold for other configurations or protocols of communication. More research in this direction, with a broader range of protocols and different scenarios, is thus justified. Finally, the hardware and network settings of this study, while representative of a testbed environment, can considerably diverge from those of industrial deployments, and therefore, the results may not be entirely reproducible. Moreover, the study has focused mainly on performance-related aspects, such as latency and resource consumption, and ignored other important factors, such as security or operation costs. Future research efforts should broaden their scope to include these aspects so that more generalized inferences about the application of protocols in industrial systems can be drawn.

4. Conclusion

This study adopted the Node-RED servers, which are widely applied tools, to realize the communication between the digital twin and PLCs. Node-RED has greatly simplified communication because it dispels static IP addresses, VPN applications, or port forwarding. Its nodes provide remote and distributed environment integration with easy deployment without complicated network configurations. The applications in industry and research abound with this approach, hence providing much-needed flexibility and scalability for handling different communication protocols. Four protocols—WebSocket/S7, MQTT, Modbus, and OPC UA—are compared to each other in terms of their pros and cons in the context of real-time integration of digital twins in industrial automation. WebSocket achieved the best latency performance by keeping low round-trip times (RTT) in local and cloud environments. However, WebSocket does not directly connect to the PLC but acts as a proxy, forwarding messages to remote destinations where the S7 nodes communicate with the Siemens S7 PLCs. As seen through the functionality, this mediating role underscores the value of WebSocket for real-time, time-dependent applications even though it is not an industry-standard industrial protocol.

MQTT provided a reasonably good performance, particularly in the necessity of a trade-off between latency and reliability. Its lightweight design makes it easy to adapt to complicated network environments, positioning it as a strong candidate for resource-limited remote applications. Being robust and straightforward, Modbus showed the highest latency, especially in cloud-based setups. Its low price and robustness make it appropriate for narrow networks or nonsensitive tasks, yet it hinders its real-time use. Despite higher latency, OPC UA presents more advanced data modeling capabilities, which, though not the primary concern of this study, are very advantageous in highly complex industrial applications. Real-world applications demonstrate the practical relevance of these protocols.

For instance, WebSocket's short latency is well-suited for real-time QC and monitoring of conveyor belts; scalability and reliability of MQTT are well-suited for predictive maintenance and remote control of devices. Modbus remains the de facto standard for legacy systems in water and power treatment areas. Still, for smart factories and across an array of vendors for industrial plants, the layered data management and interoperability of OPC UA is a significant advantage. However, the research has certain limitations. The controlled experimental setup ensured uniform network conditions, which may not accurately reflect the variations in real industrial scenarios. In addition, the paper did not include any aspect of security, which, for industrial applications, is of particular importance. While some protocols, like OPC UA, include robust security features, the analysis focused solely on latency and resource efficiency. Furthermore, the contribution did not consider the multiuser scalability, which must be taken

into account to properly assess and compare the performance weaknesses of a protocol in a multiuser environment. Additionally, the performance of Node-RED server nodes in protocol management in dynamic network conditions could be enhanced. The increase in the processing efficiency of the nodes and the decrease in latency can be considered. Long-term studies could also provide insights into the protocol's stability and reliability based on the progressive operation. Additionally, understanding the role of emerging technologies, 5G, and edge computing on the increased delay in real-time communication between the digital twins may yield some disruptive change. For example, the ultra-low latencies and high bandwidths of 5G networks can support high-throughput lossless data transmission in resource-constrained applications, and edge computing can make local processing possible by reducing the need to rely on cloud systems and shortening the response time to a regional system. Adding in artificial intelligence and machine learning to optimize protocol parameters in a continuously adaptive manner in response to real-time, dynamic network topology is another promising direction along the lines of increasing scalability and enhancing efficiency.

The communication protocol should be chosen based on the application's needs and the aspects of latency, security (if available), and the operative environment. Although WebSocket is the most suitable means for real-time applications in remote scenarios, its use as a relay for PLC activation rather than direct PLC communication should not be overlooked. MQTT and OPC UA offer reliability and scalability, whereas Modbus continues to be part of the solution for legacy systems with local network requirements. It is essential to optimize Node-RED server nodes to better support protocol integration and scalability in real-time industrial systems.

Acknowledgements

The authors express their gratitude to the Faculty of Computer Science and Information Technology (FSKTM) at Universiti Tun Hussein Onn Malaysia (UTHM), the Fatima College of Health Sciences (FCHS) at the Institute of Applied Technology (IAT), and the Department of Computer Science and Software Engineering (CSSE) at the United Arab Emirates University (UAEU) for generously providing the necessary facilities.

References

[1] Concetta Semeraro et al., “Digital Twin Paradigm: A Systematic Literature Review,” *Computers in Industry*, vol. 130, 2021. [CrossRef] [Google Scholar] [Publisher Link]

[2] Ziqi Huang et al., “A Survey on Ai-Driven Digital Twins in Industry 4.0: Smart Manufacturing and Advanced Robotics,” *Sensors*, vol. 21, no. 19. pp. 1-35, 2021. [CrossRef] [Google Scholar] [Publisher Link]

[3] Morteza Ghobakhloo, “Industry 4.0, Digitization, and Opportunities for Sustainability,” *Journal of Cleaner Production*, vol. 252, 2020. [CrossRef] [Google Scholar] [Publisher Link]

[4] Thiago Lopes da Silva, and Urbano Chagas, “How Digital Twins Is Being Used in Industry 4.0,” *Intechopen*, 2023. [CrossRef] [Google Scholar] [Publisher Link]

[5] Michael W. Grieves, *Digital Twins: Past, Present, and Future*, The Digital Twin, pp. 97-121, Springer, 2023. [CrossRef] [Google Scholar] [Publisher Link]

[6] “*ISO 23247-1, ISO 23247-4:2021: Automation Systems and Integration - Digital Twin Framework for Manufacturing, Part 1: Overview and General Principle*,” Report, International Organization for Standardization, 2021. [Google Scholar] [Publisher Link]

[7] Khalifa Alremeithi, Hassan Almaeeni, and Winston Sealy, “Virtualized Digital Twin (DT) of a Reconfigurable Programmable Logic Controller (PLC),” *2024 6th International Conference on Reconfigurable Mechanisms and Robots (ReMAR)*, Chicago, IL, USA, pp. 349-354, 2024. [CrossRef] [Google Scholar] [Publisher Link]

[8] Luis Freitas et al., “OPC-UA in Digital Twins - A Performance Comparative Analysis,” *International Conference Innovations in Mechatronics Engineering III*, pp. 113-123, 2024. [CrossRef] [Google Scholar] [Publisher Link]

[9] Eduardo Zancul, “A Digital Twin Design and Implementation Approach for Industrial Application Leveraging Programmable Logic Controllers,” *International Journal on Interactive Design and Manufacturing (IJIDeM)*, pp. 1-9, 2024. [CrossRef] [Google Scholar] [Publisher Link]

[10] Carlos Cremonini et al., “Design and Implementation of a Digital Twin for a Stone-Cutting Machine: A Case Study,” *Procedia Computer Science*, vol. 232, pp. 990-1000, 2024. [CrossRef] [Google Scholar] [Publisher Link]

[11] Haoran Wang et al., “A Digital Twin Platform Integrating Process Parameter Simulation Solution for Intelligent Manufacturing,” *Electronics*, vol. 13, no. 4, pp. 1-21, 2024. [CrossRef] [Google Scholar] [Publisher Link]

[12] Hanna Molin et al., “Automated Data Transfer for Digital Twin Applications: Two Case Studies,” *Water Environment Research*, vol. 96, no. 7, pp. 1-10, 2024. [CrossRef] [Google Scholar] [Publisher Link]

[13] MQTT - The Standard for IoT Messaging, MQTT, 2024. [Online]. Available: https://mqtt.org/

[14] The OPC Unified Architecture (UA), OPC Foundation, 2024. [Online]. Available: https://opcfoundation.org/

[15] How do You Configure and Program an S7 Connection and the "PUT" and "GET" Instructions for Data Transfer between Two S7 CPUs?, Siemens, 2020. [Online]. Available: https://support.industry.siemens.com/cs/document/82212115/how-do-you-configure-and-program-an-s7-connection-and-the-%E2%80%9Cput%E2%80%9D-and-%E2%80%9Cget%E2%80%9D-instructions-for-data-transfer-between-two-s7-cpus-?dti=0&lc=en-US

[16] Modbus Organization, Modbus, 2024. [Online]. Available: https://www.modbus.org/

[17] Claire Palmer et al., *Virtual Reality Based Digital Twin System for Remote Laboratories and Online Practical Learning*, Advances in Transdisciplinary Engineering, pp. 277-283, 2021. [CrossRef] [Google Scholar] [Publisher Link]

[18] M. Melo de Carvalho et al., "Industrial Real-Time Digital Twin System for Remote Teaching Using Node-Red," *14th Annual International Conference of Education, Research and Innovation*, pp. 6623-6632, Online Conference, 2021. [CrossRef] [Google Scholar] [Publisher Link]

[19] IEC 61158-1: Fieldbus for Use in Industrial Control Systems - Data-Link Layer Service Definition, Industrial Communication Networks, International Electrotechnical Commission, 2007. [Online]. Available: https://webstore.iec.ch/en/publication/19154

[20] Diego R. C. Silva et al., "Latency Evaluation for MQTT and WebSocket Protocols: An Industry 4.0 Perspective," *2018 IEEE Symposium on Computers and Communications (ISCC)*, Natal, Brazil, pp. 1233-1238, 2018. [CrossRef] [Google Scholar] [Publisher Link]

[21] B.A. Enache, C.K. Banica, and Ana Geroge Bogdan, "Performance Analysis of MQTT Over Websocket for IoT Applications," *The Scientific Bulletin of Electrical Engineering Faculty*, vol. 23, no. 1, pp. 46-49, 2023. [CrossRef] [Google Scholar] [Publisher Link]

[22] Nick O'Leary, and Dave Conway-Jones, Node-RED, IBM Emerging Technology, Wikipedia, 2024. [Online]. Available: https://en.wikipedia.org/wiki/Node-RED

[23] Overview of Amazon Web Services - AWS Whitepaper, Amazon Web Services, 2023. [Online]. Available: https://docs.aws.amazon.com/whitepapers/latest/aws-overview/introduction.html

[24] Murilo Silveira Rocha et al., "On the Performance of OPC UA and MQTT for Data Exchange between Industrial Plants and Cloud Servers," *Acta IMEKO*, vol. 8, no. 2, pp. 1-8, 2019. [CrossRef] [Google Scholar] [Publisher Link]

[25] Arda Kocamuftuoglu, Okan Akbay, and Serkan Kaba, "A Comparative Study on Industrial Communication Protocols Using IoT Platforms," *The Eurasia Proceedings of Science Technology Engineering and Mathematics*, vol. 14, pp. 57-65, 2021. [CrossRef] [Google Scholar] [Publisher Link]

[26] Narongsak Putpuek, Apiradee Putpuek, and Sansern Phawandee, "Performance Evaluation of OPC UA and MQTT for ETAT Smart Lab (ESL)," *2023 7th International Conference on Information Technology, (InCIT)*, Chiang Rai, Thailand, pp. 17-21, 2023. [CrossRef] [Google Scholar] [Publisher Link]

[27] Cornel Ventuneac, and Vasile Gheorghita Gaitan, "Industrial Internet of Things Gateway with OPC UA Based on Sitara AM335X with ModbusE Acquisition Cycle Performance Analysis," *Sensors*, vol. 24, no. 7, pp. 1-13, 2024. [CrossRef] [Google Scholar] [Publisher Link]

[28] Walczak Maciej, and Hetmanczyk Mariusz, "The Performance of IIOT Communication Standards," *Diagnostics*, vol. 24, no. 3, pp. 1-8, 2023. [CrossRef] [Google Scholar] [Publisher Link]

[29] Alexey Melnikov, and Ian Fette, RFC6455 - The WebSocket Protocol, IETF Datatracker, 2011. [Online]. Available: https://datatracker.ietf.org/doc/html/rfc6455

[30] Unity, Unity Platform, (2021). [Google Scholar] [Publisher Link]

[31] S7-PLCSIM Advanced - Simulation for Virtual Commissioning and Testing, Siemens, 2024. [Online]. Available: https://www.siemens.com/global/en/products/automation/systems/industrial/plc/s7-plcsim-advanced.html

[32] GitHub - OPCFoundation/UA-.NETStandard: OPC Unified Architecture, OPC Foundation, 2024. [Online]. Available: https://github.com/OPCFoundation/UA-.NETStandard

[33] GitHub - Rossmann-Engineering/EasyModbusTCP.NET: Modbus TCP, Rossmann-Engineering, 2021. [Online]. Available: https://github.com/rossmann-engineering/EasyModbusTCP.NET

[34] GitHub - Eclipse-Paho/Paho.mqtt.m2mqtt, MqttClient, Paolo Patierno, 2017. [Online]. Available: https://github.com/eclipse/paho.mqtt.m2mqtt

[35] Websocket-Sharp, STA, 2024. [Online]. Available: https://sta.github.io/websocket-sharp/