


# IloT Protocols for Edge/Fog and Cloud Computing in Industrial AI: A High Frequency Perspective

Telmo Fernández De Barrena Sarasola, Faculty of Engineering, University of Deusto, Mundaitz Kalea, 50, 20012 Donostia- San Sebastian, Spain & Department of Data Intelligence for Energy and Industrial Processes, Fundación Vicomtech, Basque Research and Technology Alliance (BRTA), Mikeletegi 57, 20009 Donostia-San Sebastian, Spain\*

 <https://orcid.org/0000-0001-8577-1995>

Ander García, Department of Data Intelligence for Energy and Industrial Processes, Fundación Vicomtech, Basque Research and Technology Alliance (BRTA), Mikeletegi 57, 20009 Donostia-San Sebastian, Spain & Faculty of Engineering, University of Deusto, Mundaitz Kalea, 50, 20012 Donostia- San Sebastian, Spain

Juan Luis Ferrando, Department of Data Intelligence for Energy and Industrial Processes, Fundación Vicomtech, Basque Research and Technology Alliance (BRTA), Mikeletegi 57, 20009 Donostia-San Sebastian, Spain

## ABSTRACT

Various industrial applications deal with high-frequency data. Traditionally, these systems have analyzed high-frequency data directly on the data source or at the commanding PLC. However, currently, Industry 4.0 technologies support new monitoring scenarios for high-frequency data monitoring where raw data is transmitted in soft-real time to an Edge/Fog or Cloud node for processing, enabling centralized computing. This demands efficient communication protocols capable of handling high-frequency, high-throughput data. This paper focuses on analyzing the performance of key IloT (Industrial Internet of Things) messaging protocols—AMQP, MQTT, KAFKA, ZeroMQ, and OPCUA—to evaluate their suitability, in terms of latency and jitter, for transmitting high-frequency data within these new scenarios. The analysis reveals MQTT, AMQP, and ZeroMQ as top performers in Edge/Fog computing, while ZeroMQ exhibits the lowest latency and jitter in Cloud computing. Finally, a guideline for protocol selection is proposed, aiding industrial enterprises in protocol selection for specific AI use cases.

## KEYWORDS

Cloud Computing, Edge/Fog Computing, High Frequency Signals, IloT Protocols, Industry 4.0

Traditional industrial automation engineering workflows have long relied on PLC (Programmable Logic Controller) or SCADA (Supervisory Control And Data Acquisition) systems to manage processes and machines. These systems, typically proprietary products from companies like Siemens, Beckhoff, or GE Digital, are programmed using specific tools provided by their manufacturers. They primarily offer control functionalities rooted in classical control engineering and often work in relative isolation, receiving external orders with limited internal data sharing.

DOI: 10.4018/IJAC.342128

\*Corresponding Author

This article published as an Open Access article distributed under the terms of the Creative Commons Attribution License (<http://creativecommons.org/licenses/by/4.0/>) which permits unrestricted use, distribution, and production in any medium, provided the author of the original work and original publication source are properly credited.

However, the advent of the Industry 4.0 paradigm has imposed new demands on these systems, necessitating enhanced data sharing and utilization. Manufacturing lines are generating an escalating volume of data, prompting the monitoring of more variables at higher frequencies. This shift involves capturing data that ranges from a few critical variables per batch to time series data of multiple variables recorded at frequencies of seconds or even greater (Garcia et al., 2023).

In typical industrial scenarios, the flow of high-frequency data begins with sensors strategically placed on machinery to capture relevant signals such as vibration, sound, or acceleration. These sensors collect data at high sampling rates, from thousands to millions of data points per second, generating a stream of raw information (Fernández de Barrena Sarasola et al., 2023; Kuntoğlu et al., 2021) and huge volumes of data (Kumar & Agrawal, 2023). This data usually undergoes preprocessing in the machine to filter out noise and irrelevant information, ensuring the accuracy and reliability of subsequent analysis. Advanced signal processing algorithms are then applied to extract meaningful features and patterns from the data, making possible the development of Artificial Intelligence (AI)-based models for the creation of predictive maintenance, quality control, and similar scenarios to classify anomalies, predict failure probabilities, and recommend appropriate maintenance actions.

However, the Industry 4.0 paradigm is changing the way in which data is processed, making it possible to send raw data either to the edge/fog or to the cloud from the sensors and centralize the processing and machine learning (ML) models or expert systems deployment steps. Thus, there is a need to create systems able to efficiently handle data at high throughputs. Here, the employed protocols and network play a key role.

To face these new requirements, new architectures are required to integrate the Information Technology (IT) and Operations Technology (OT) fields. The convergence of IT and OT represents a paradigm shift, breaking down historical barriers and fostering a unified framework where data, traditionally confined to either the enterprise or the operational field, can flow seamlessly between both. This integration has opened the way to a new era of interconnected and intelligent systems, enabling the optimization of operational processes and the development and integration of data-driven methods, which bring industries greater agility and competitiveness (Nath et al., 2020).

The combination of AI and ML in industrial processes further amplifies the potential of the IT and OT convergence. ML algorithms, deployed at the edge/fog or within the cloud, empower systems to detect patterns, predict failures, and optimize performance autonomously (Lecun et al., 2015). From predictive maintenance to anomaly detection, ML applications are reshaping the way industries operate, providing intelligence into every layer of the value chain (Alshehri & Muhammad, 2021).

In the pursuit of enhanced efficiency and soft-real-time decision making, the integration of edge/fog and cloud computing has emerged as a cornerstone in modern industrial communication frameworks (Kumar & Agrawal, 2023). Cloud computing is widely regarded as a crucial tool for meeting the computing needs of resource-intensive applications. However, the cloud shows communication delays and saturated networks due to a lack of bandwidth, which is due to the information overload caused by the scaling of Internet of Things (IoT) devices (Oñate & Sanz, 2023). The drawbacks associated with cloud computing present challenges in meeting the performance demands of time-sensitive applications, such as augmented reality, autonomous driving, and interactive online gaming (Mao et al., 2021).

Edge/fog computing brings computation closer to data sources, reducing latency and enabling rapid analysis of vast datasets (Mao et al., 2022; Shi et al., 2016). This proximity to data generation points is particularly vital in industries where timely insights can dictate operational success (Fernández de Barrena Sarasola et al., 2023).

Distributed deployment of real-time applications and high-speed dissemination of massive data are key features of Industrial Internet of Things (IIoT) platforms. IIoT applications typically adopt publish/subscribe (pub/sub) middleware for asynchronous and cross-platform communication. Communication protocols play a key role in this integration, enabling the secure and efficient exchange of data across heterogeneous systems. The landscape of available protocols is diverse, offering a spectrum of choices

tailored to different industrial needs (Ullah et al., 2020), such as Message Queuing Telemetry Transport (MQTT), Zero Message Queuing (ZeroMQ) and Data Distribution Service (DDS). Understanding the strengths, weaknesses, and interoperability of these protocols is crucial for building resilient, future-proof communication infrastructures (Kang & Dubey, 2020).

Under the context of IIoT, the manufacturing industry is moving toward the servitization of servers' allocation, to centralize the computing resources and to minimize the cost and effort associated with deploying and maintaining servers. This, along with the integration of communication protocols, enables the development and integration of data-driven methods, which bring industries greater agility and competitiveness. Thus, the knowledge and monetary cost of deploying AI models for different tasks, such as prognosis and health management (PHM) and production optimization, can be significantly reduced. For performing those tasks, communication protocols plays a key role, enabling secure and efficient exchange of data across heterogeneous systems.

However, after analyzing the existing research in the literature, the researchers of this paper observe that there is a need to analyze the performance of different protocols such as Advanced Messaging Queuing Protocol (AMQP), MQTT, KAFKA, ZeroMQ, and OP CUA, under cloud and edge/fog network and high frequency data scenarios, to understand the strengths and weakness of these protocols. Different features are relevant for this analysis, such as stream mean and standard deviation latency and jitter, the variation in the latency on a stream flow between two systems. These features must be measured and analyzed to gain a clear vision about which networking approach (edge/fog or cloud) and protocol each user should choose to fulfill its use case requirements. For example, jitter, which is not a widely analyzed element, plays a key role, especially in real-time communications, as high values of this element result from network congestion, timing drift and inconsistencies, which degrade the quality of communications.

The nature of IIoT protocols is one of the main drawbacks when using them in soft real-time performance scenarios. Figure 1 reveals the differences between OT and IT protocols latency distributions. OT-based solutions are designed to minimize jitter, which is the variation in time delay between when a signal is transmitted and when it is received over a network connection, and to provide near-real-time deterministic behavior down to the millisecond. When having a specific maximum latency requirement,  $t + \Delta t$ , due to its deterministic nature, OT protocols can ensure fixed range latencies. Figure 1 shows an ideal OT protocol latency distribution, having a jitter of 0, and thus a single latency value, represented as a blue vertical line. Unlike OT protocols, the data transfer times of IT ones follow a probability positively skewed time distribution, shown in blue in the right side of Figure 1. In this case, IT protocols cannot ensure fixed-range latencies, resulting in the possibility of having higher latencies than required.

For the abovementioned reasons, this paper compares the performance of different IIoT protocols working along with high frequency data in edge/fog and cloud scenarios. Thus, different key parameters such as latency, jitter, lost or not-ordered packages, etc. are measured under different working conditions, changing the sampling frequency and stream sizes, and consequently the throughput. As results in terms of absolute time values are highly dependent on the employed network characteristics, this research focuses on a normalized comparison of the protocols performance. Moreover, needed code for replicating the experiments and obtained results is provided here: <https://github.com/Vicomtech/IIOT-protocols-study-for-high-frequency-data-in-the-edge-and-cloud/tree/main>. Computational overhead of each of the tested protocols has also been analyzed.

The initialisms and acronyms used in this paper are listed in Table 1 with their definitions.

This research intends to provide guidelines on which protocol to employ, depending on the use case requirements, for developing efficient architectures, centralizing as much as possible the needed computing load in the edge/fog or cloud, where more resources are available. Thus, the main objective of this paper is to ease the selection of optimum IIoT communication protocols to obtain the best performance in terms of latency and jitter when working with high-frequency data and sending it to

Figure 1. OT and IT Data Transfer Times Distributions

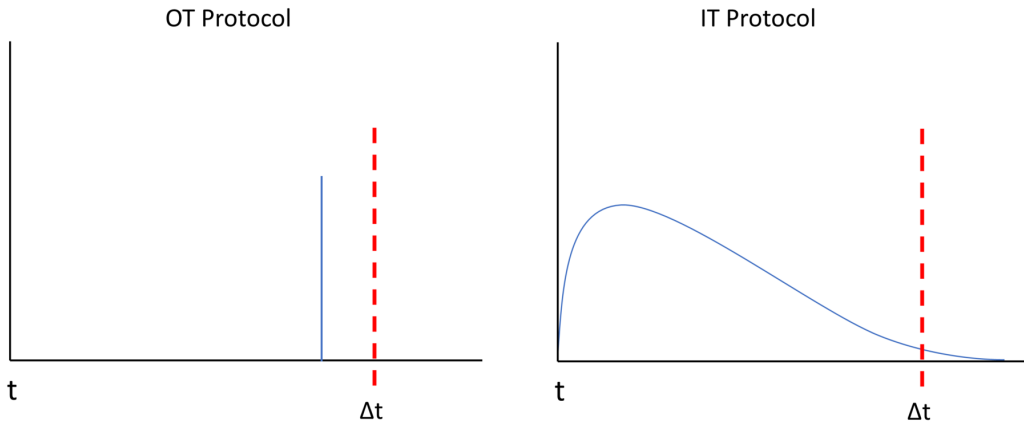


Table 1. List of Initialisms and Acronyms Used in This Paper

Abbreviation	Definition
PLC	Programmable Logic Controller
SCADA	Supervisory Control And Data Acquisition
AI	Artificial Intelligence
ML	Machine Learning
IT	Information Technology
OT	Operational Technology
IoT	Internet of Things
IIoT	Industrial Internet of Things
MQTT	Message Queuing Telemetry Transport
DDS	Data Distribution Service
AMQP	Advanced Message Queuing Protocol
PHM	Prognostics and Health Management
CPS	Cyber Physical Systems
CoAP	Constrained Applications Protocol
RTPS	Real Time Publish Subscribe
JMS	Java Message Service
ROS	Robot Operating System
RTT	Round-Trip Time

the edge/fog or cloud. This is a key point when integrating soft AI models, such as PHM models, into cyber-physical systems (CPS), and deploying data-based models in the edge/fog or cloud.

The remainder of this paper is organized as follows: Section 2 presents the state of the art. Section 3 presents an overview of the analyzed protocols. Section 4 describes the experimental setup for performing the experiments. Section 5 presents the methodology followed in performing the

experiments. Section 6 shows the results obtained by the performed experiments. Finally, Section 7 offers concluding remarks.

## STATE OF THE ART

Several studies have been found comparing the performance of different IoT communication protocols. The studies were performed employing different hardware components, such as Arduino, Raspberry Pi, etc.

Naik (2017) described the properties of MQTT, CoAP, AMQP, and HTTP protocols, comparing them looking at different parameters. The message size, frame size, energy consumption, resource requirement, bandwidth usage, latency, stability, supported platforms, security, IoT compliance, and standardization parameters were analyzed. The study was based on different scientific studies, not providing an experimental environment.

Bayılmış et al. (2022) claimed that there is no general model, approach, or benchmark for performance comparison because the ranges of the IoT-based applications are so versatile. In this study differences in energy consumption and throughput for IoT application layer communication protocols, such as Constrained Applications Protocol (CoAP), MQTT, and WebSocket on tiny IoT devices, were analyzed. Lombardi et al. (2021) discussed the existing framework architectures, technologies, protocols, and applications under the IoT paradigm.

Chaudhary et al. (2017) compared MQTT, CoAP, and AMQP protocols in wired, wireless and 4G connections, employing Raspberry Pi3 as IoT devices. Dizdarevic et al. (2019), performed a theoretical survey of IoT communication protocols, including MQTT, AMQP, XMPP, DDS, HTTP, and CoAP. Latency, energy consumption, and network throughput parameters were analyzed.

Safarov (2018), employing local Wi-Fi and a Raspberry Pi B on the client side and a laptop with i7 processor on the server side, compared WebSocket, MQTT, and CoAP. Using a mathematical method, throughput parameters on different stream loss rates and the effect of the frame sizes were examined.

Gavrilov et al. (2022) analyzed MQTT, Real Time Publish Subscribe protocol (RTPS, an interoperability protocol for DDS implementations), Java Message Service (JMS), and AMQP protocols to find out what tasks these protocols should be used for and whether they can be used in robotic and autonomous systems where high data transmission requirements are imposed. They concluded that RTPS is the best solution for real-time systems with different traffic and that MQTT performs well when transmitting short messages.

Profanter et al. (2019) gave an overview on the different features of OPC UA, Robot Operating System (ROS), DDS, and MQTT and compared their performance in several benchmarks. They concluded that open62541, which is an open source and free implementation of OPC UA, and eProxima FastRTPS for DDS, deliver high performance, whereas the MQTT and ROS implementations showed a significant slowdown in the round-trip time (RTT) of packages sent to the server.

Suri (2019) compared different protocols over an edge network. The results showed that the ZeroMQ implemented outperformed other protocols such as DDS, MQTT, AMQP, and Kafka in terms of bandwidth utilization and latency when they operated in a saturated communications environment.

Lazidis et al. (2022) conducted a survey and taxonomy of publish–subscribe systems, of their design features and technologies. The latency of Orion-LD, Stellio, Scorpio, Pushpin, Faye, Apache Kafka, and RabbitMQ were compared, concluding that for heavy workloads, Apache Kafka and RabbitMQ proved to be fast and scalable.

Finally, Kang and Dubey (2020) empirically evaluated the performance of three pub/sub technologies: OMG DDS, MQTT, and ZeroMQ, for representative IIoT scenarios (high-frequency, periodic, and sporadic) under a cluster with a bandwidth of 95 Mbps. Results showed that in the higher-frequency scenarios, when the message was smaller than 1 KB, ZeroMQ performed best. However, when the message was larger than 1KB, ZeroMQ throughput became lower than DDS. Compared

with DDS and ZeroMQ, the throughput of MQTT was poor at the broker-centric architecture used in the paper.

To the authors' knowledge, no analysis has been performed on the latency and jitter values of IIoT protocols working along with high frequency data in edge/fog and cloud scenarios. For this reason, the present research compares the performance of different IIoT protocols working along with high-frequency data in edge/fog and cloud scenarios, analyzing different key parameters such as latency, jitter, lost or not-ordered packages, etc. The authors consider the analysis of jitter in high-frequency scenarios a key point, as it significantly impacts network performance by introducing irregularities and variations in the transmission of data streams.

## **PROTOCOLS OVERVIEW**

In this section, an introduction of the tested protocols under edge/fog and cloud scenarios is presented. AMQP, MQTT, and KAFKA are broker dependent and are used with a pub/sub approach. ZeroMQ is employed using the same approach, but being brokerless and opening sockets. OP CUA is tested using a client/server approach.

### **AMQP**

The AMQP is an open standard for passing business messages between applications or organizations. It connects systems, feeds business processes with the information they need, and reliably transmits onward the instructions that achieve their goals (AQMP, 2023).

In this paper, for implementing the AMQP standard, RabbitMQ has been employed as an open-source message broker. Ionescu (2015) provides detailed information about this message broker. Moreover, Pika (version 1.3.1) client library, which is a RabbitMQ (AMQP 0-9-1) client library for Python, has been utilized.

### **MQTT**

MQTT is an OASIS standard messaging protocol for the Internet of Things (IoT). It is designed as an extremely lightweight pub/sub messaging transport that is ideal for connecting remote devices with a small code footprint and minimal network bandwidth. Today MQTT is used in a wide variety of industries, such as automotive, manufacturing, telecommunications, oil and gas, etc. (MQTT, 2022). Suri (2019) provides detailed information about this protocol.

In this paper, the MQTT standard has been implemented through Eclipse Mosquitto, as open-source message broker. Moreover, Paho MQTT (version 1.6.1) client library, which is an Eclipse Mosquitto client library for Python, has been employed.

### **Apache KAFKA**

Apache Kafka is an open-source distributed event streaming platform for high-performance data pipelines, streaming analytics, data integration, and mission-critical applications. It provides a pub/sub messaging model for data production and consumption and supports the ability to access data in real time for stream processing by allowing long-term storage of data (Apache Software Foundation, 2023). Kafka was designed from the ground up to provide long-term data storage and data replay. It has a unique approach to data persistence, fault tolerance, and replay. This approach can be seen in how it handles scalability by allowing data access using cross-partition data sharing, topics/partitions, data offsets, and consumer group names for data replication persistence in clusters, increased data volume, and load. Apache Kafka is also well suited for real-time stream processing applications because it is designed to act as a communication layer for real-time log processing. This capability makes Apache Kafka suitable for applications running on communications infrastructure that process large amounts of data in real time (Nam et al., 2022).

In this paper, for implementing this platform, Confluent-Kafka-Python (version 2.2.0), which is a Python client that provides a high-level producer, consumer, and AdminClient that are compatible with Kafka brokers, has been deployed.

## ZeroMQ

ZeroMQ looks like an embeddable networking library but acts like a concurrency framework. It provides sockets that carry atomic messages across various forms of transport such as in-process, inter-process, TCP, and multicast. Sockets can be connected as N-to-N with patterns like fan-out, pub/sub, task distribution, and request-reply. ZeroMQ's asynchronous I/O model allows scalable multicore applications, built as asynchronous message-processing tasks. It has a score of language APIs and runs on most operating systems (ZeroMQ, 2023).

In this paper, for implementing this protocol, pyzmq (version 25.1.1), which is a package containing Python bindings for ZeroMQ, has been employed, making use of the pub/sub approach.

## OPC UA

OPC is the interoperability standard for the secure and reliable exchange of data in the industrial automation space and in other industries. The OPC standard is a series of specifications developed by industry vendors, end-users, and software developers. These specifications define the interface between clients and servers, as well as between servers and servers, including access to real-time data, monitoring of alarms and events, access to historical data, and other applications (OPC Foundation, 2023).

Even though OPC UA is used mostly in higher automation levels for the purpose of monitoring and control, it also increases device connectivity via standard communication in lower automation levels (Hegazy & Hefeeda, 2015). Therefore, OPC UA was named as candidate for communication aspects in RAMI4.0. The OPC-UA approach abstracts data from the network technology and software application and offers a generic communication interface. It can be seen as one of the key technologies for a transparent data representation/ transmission between heterogeneous system components (Imtiaz & Jasperneite, 2013). Schleipen et al. (2016) provides detailed information about this protocol. This protocol has two operational models, server-client and pub/sub. This paper analyzes the server-client approach, in which each client establishes a connection with a server, as it is the most widely used. The pub/sub approach uses either UDP or MQTT as transport protocols. For the use presented in the paper, MQTT is the most suitable one. As results from OPC UA pub/sub based on MQTT would be remarkably similar to the ones obtained using MQTT, OPC UA pub/sub analysis has not been included in this paper. In this paper, for implementing the OPC UA standard, the Free OPC-UA (version 0.90.6) client library for Python has been employed.

## EXPERIMENTAL SETUP

To explore the benefits and drawbacks of the different IIoT protocols introduced in Section 3, under edge/fog and cloud scenarios, the following two setups, (shown in Figure 2) were employed for performing different experiments. The experiments consisted of two or three main components, depending on if the protocols were brokerless or not, and all of them dockerized and launched as docker containers:

- IIoT device simulator (colored in yellow):
  - Producer: Simulated the generation of high-frequency sensor data and sent data streams to the AI service located in the edge/fog or cloud. Detailed information about the data generation is explained in Section 5.

- AI component (colored in green or blue, depending on the scenario): Simulated an AI service. This service simply took the data from the IIoT simulator and sent a response back again, simulating an inference performed by a ML model. As the goal of this research was to evaluate only the protocols performance, to avoid unnecessary delays due to computational needs, no ML models were deployed in the edge/fog or cloud.
- IIoT device simulator (colored in yellow):
  - Consumer: Once data was processed by the AI service, the IIoT device simulator received the response of the AI service corresponding to the previously sent data stream.

The numbers in the corner of the services of the scenarios in Figure 2 represent the moments in which different times have been measured. These numbers represent the following:

1. Time when a data stream was sent from the IIoT simulator.
2. Time when the data stream was received in the AI service.
3. Time when the data stream was sent back from the AI service to the IIoT simulator.
4. Time when the data stream was received back in the IIoT simulator.

After measuring those times, performing the operation (1), the mean latency values were calculated.

$$mean\ latency = \frac{1}{N} \sum_{i=1}^N ((t_{4,i} - t_{1,i}) - (t_{3,i} - t_{2,i}) - (t_{save,i} - t_{4,i})) \quad (1)$$

where  $t_{n,i}$ ,  $n=1,2,3,4$ , is the timestamp corresponding to each of the explained moments corresponding to a data stream;  $t_{save,i}$  is the timestamp when that data stream is saved; and  $N$  is the total amount of streams corresponding to one experiment. By subtracting  $(t_{3,i} - t_{2,i})$  and  $(t_{save,i} - t_{4,i})$  to  $(t_{4,i} - t_{1,i})$ , we obtain the pure latency of the data stream, without considering the processing times of the IIoT and AI services.  $(t_{3,i} - t_{2,i})$  represents the processing time of the AI service, and  $(t_{save,i} - t_{4,i})$  represents the processing time of the IIoT simulator, which are values that must not be considered for the analysis.

Equation (2) is used to calculate the mean jitter:

$$mean\ jitter = \frac{1}{N} \sum_{i=1}^{N-1} |(l_{i+1} - l_i)| \quad (2)$$

where  $l_i$  is the real latency of the corresponding data stream and  $l_{i+1}$  is the real latency of the next data stream, both AI service and IIoT simulator processing times subtracted. Thus, we obtain the jitter corresponding to the protocol.

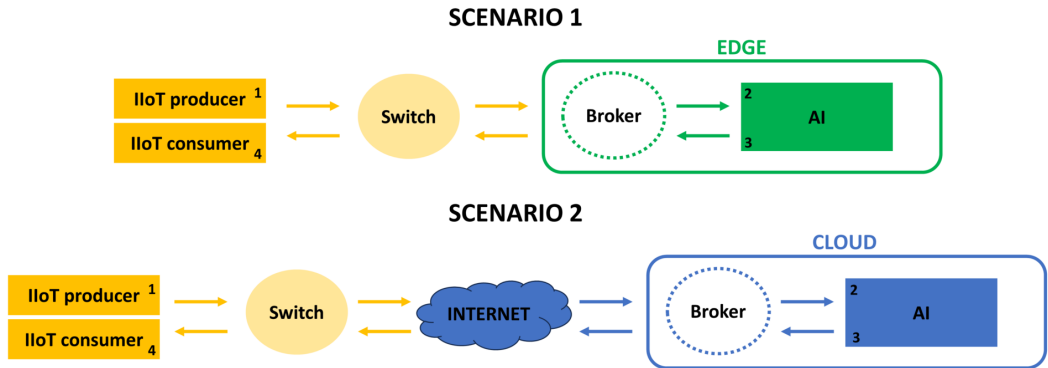
Moreover, the latency and jitter standard deviation values, not-ordered packages, and lost packages have been calculated.

In both scenarios, the IIoT simulator service launched under a computer with the following characteristics to simulate high frequency data: Processor Intel(R) Core(TM) i5-10400 CPU @ 2.90GHz, 2904 MHz, 6 Core(s), 12 Logical Processor(s), 16Gb RAM.

As represented in Figure 2, the first scenario simulated an edge/fog computing based scenario, where the broker, if necessary, and the AI service were launched in the edge/fog server. This edge/



Figure 2. Scenarios Employed for Performing the Experiments



fog server was simulated with another computer with the same characteristics as employed for the IIoT simulator. In this case, both computers worked within the same enterprise network. Contrary to scenario 1, in the second one, the broker, if necessary, and the AI service were launched on a cloud server. This server corresponded to a T2 medium EC2 instances launched in Amazon AWS containing 2vCPUs and 4Gb RAM.

## METHODOLOGY

This section presents the methodology employed to perform the experiments. Table 2 represents the different experiments performed for each of the communication protocols in both scenarios. In total, nine different experiments were performed for each protocol, stream size, and frequency, and consequently the throughput. Thus, we can compare different levels of throughput, which is the best protocol in different scenarios. For each of the possible throughput, different stream size and sampling frequency combinations were investigated, to analyze the impact of this combination in different protocols. The performed experiments used Int32 type data points, corresponding to 4 bytes each element. Thus, the experiments ranged from 250 kHz to 2 MHz sampling frequency. The authors

Table 2. Experiments Performed for Each of the Tested Protocols in Both Scenarios

THROUGHPUT (Mbytes/sec)	STREAM SIZE (Kbytes)	DATA POINTS/STREAM	STREAMS/s	SAMPLING FREQUENCY (kHz)
1	25	6.250	40	250
1	50	12.500	20	250
1	100	25.000	10	250
4	100	25.000	40	1.000
4	200	50.000	20	1.000
4	400	100.000	10	1.000
8	200	50.000	40	2.000
8	400	100.000	20	2.000
8	800	200.000	10	2.000

Figure 3. High-Frequency Data Generation Algorithm Pseudocode

---

**Algorithm 1** High Frequency Data Simulator

---

```
1: for iteration = 1, 2, ..., Cycles do
2:   Initialize variable T=0 to accumulate how much extra time has
   elapsed
3:   for stream = 1, 2, ..., SamplinFrequency do
4:     Initialize variable timestamp1
5:     Generate and send data stream
6:     Initialize variable timestamp2
7:     Initialize variable streamtime=timestamp2-timestamp1
8:     if streamtime + T > 1/SamplinFrequency then
9:       Do not sleep and update value of accumulated extra time
10:      T=streamtime+T-1/SamplinFrequency
11:     else
12:       Sleep (1/SamplinFrequency-streamtime-T)
13:       set T=0
14:     end if
15:   end for
16: end for
```

---

believe that this range covers a wide range of possibilities that could be realistic in high-frequency industrial scenarios.

Each of the experiments was performed with a duration of 30 seconds (cycles). In addition, to ensure the consistency of the experiments, each of them was performed 45 times, in batches of 15, on different days and times, and the mean of the results was obtained. The mean and standard deviation latency and jitter distributions were analyzed, to ensure that they were similar and that the employed network did not distort the results. As the network congestion could not be controlled, this approach was used to minimize the network performance variability and helps to demonstrate protocols performance differences in a more realistic manner.

To simulate the generation of data with high precision, the following approach (Figure 3) was developed. In most cases, the data was generated and sent faster than the theoretical sampling time of one stream, being equal to  $1/\text{SamplingFrequency}$ . However, due to the stochastic nature of the process, in a few instances it could occur that the data generation and sending lasted more than the theoretical sampling time. For that reason, an accumulative variable,  $T$ , was created to store the accumulated extra time and to try to compensate that time in future iterations.

As mentioned in Section 1, results absolute time values are highly dependent on the employed network characteristics and congestion. For that reason, in Section 6 this research presents a normalized comparison of the protocols' performance. Thus, the mean and standard deviation latency and jitter values of the presented experiments have been normalized. Moreover, all figures are presented in logarithmic scale. This helps to clearly visualize the results when values are low. For all the presented results, not-ordered and lost packages metrics are not shown, as, in all the experiments, there are not lost or unordered packages.

As explained in Section 3., OP CUA is only tested under edge/fog computing scenarios. In cloud computing scenarios, this protocol can be implemented with a pub/sub approach employing MQTT. However, it is not tested, as the obtained results would be remarkably similar to the ones obtained with Eclipse Mosquitto.

To analyze the computational overhead of each of the protocols, first, edge/fog/cloud device services memory and CPU utilization without data traffic were measured (see Figure 2). Next, IIoT and edge/fog/cloud devices services memory and CPU utilization were measured, under 100 Kbytes/s and 10 streams/s working conditions. To ensure the consistency of the experiments, each of them was performed 10 times, and the mean of the results was calculated.

## RESULTS

The following section is divided as follows: first, edge/fog experiment results are presented. Next, cloud experiment results are presented. Tables 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, and 20, in the Appendix, present edge/fog and cloud scenarios results in terms of absolute and normalized values. Finally, computational overhead results are shown.

### Edge/Fog Experiment Results

Figure 4 compares the IIoT messaging protocols performance for 1Mbytes/s throughput edge/fog experiments in mean and standard deviation latency terms. To clearly visualize the results, just the positive values of the standard deviation are represented as vertical black lines just above the bars, not showing the negative side of them. The legend of the figure is composed of the protocol name, the stream size value in Kbytes, and the stream frequency. Each of the protocols is shown in a different color. All the figures from that point on follow the same structure.

Sending 25 Kbytes streams at a frequency of 40 streams/s with MQTT is the scenario that obtains the lowest latency, closely followed by any of AMQP and ZeroMQ protocols. When employing 100 Kbytes streams at a sampling frequency of 10 streams/second, MQTT protocol is clearly the one that obtains the lowest latency, being 1.23 times faster than ZeroMQ and at least 2 times faster than the rest of the protocols.

Figure 5 compares the IIoT messaging protocols performance for 1Mbytes/s throughput edge/fog experiments in mean and standard deviation jitter terms.

Figure 4. Normalized Latency of 1 MBytes/s Edge/Fog Experiments

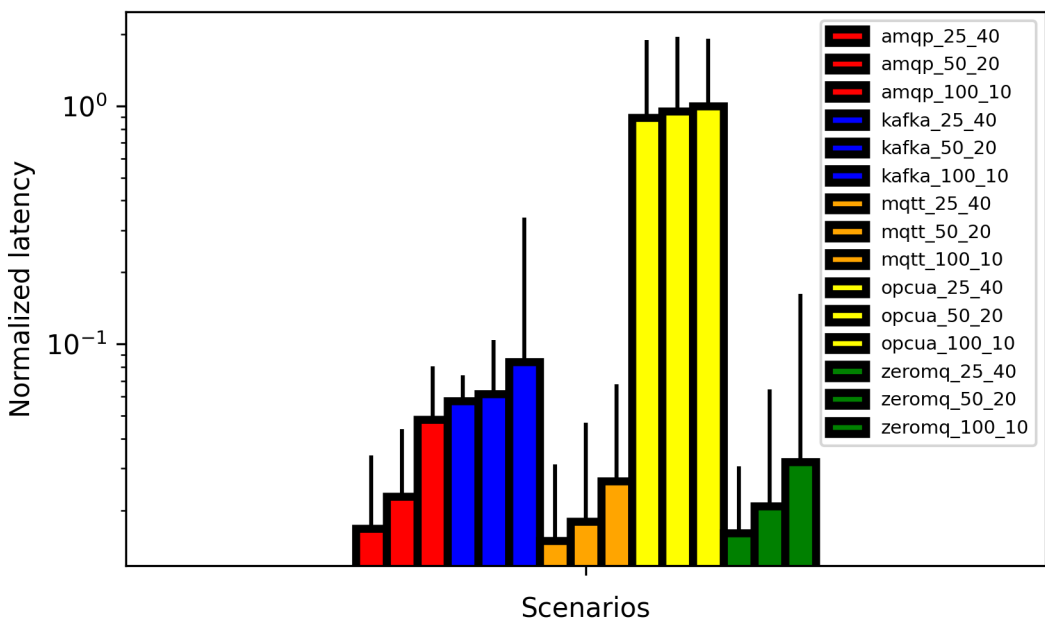
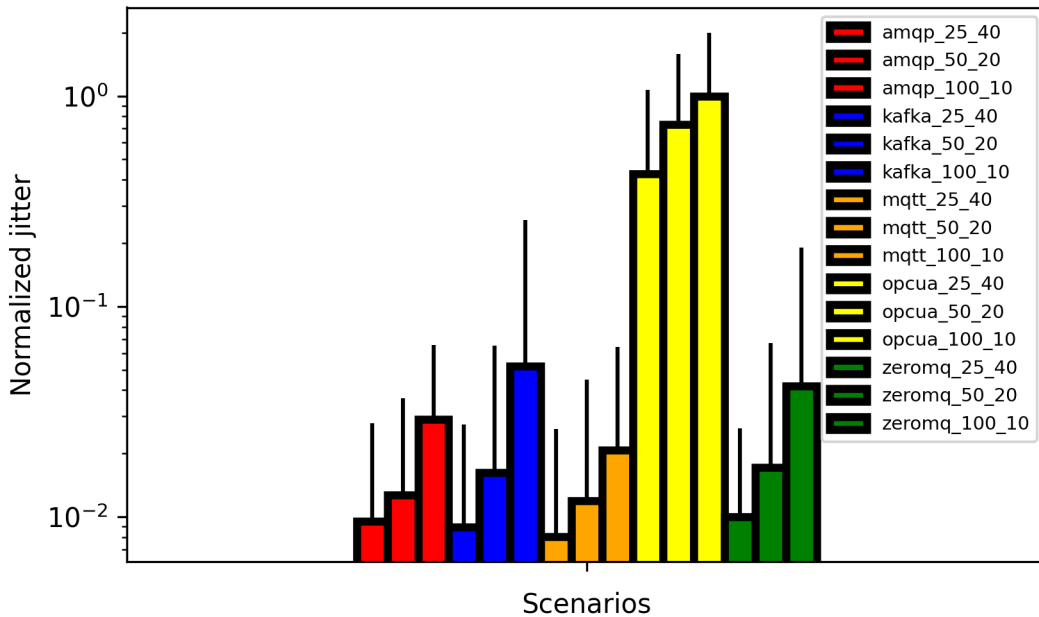


Figure 5. Normalized Jitter of 1 MBytes/s Edge/Fog Experiments



Sending 25 Kbytes streams at a frequency of 40 streams/s, with MQTT protocol, is the scenario that obtains the lowest jitter, closely followed by AMQP, KAFKA, and ZeroMQ protocols. When employing 100 Kbytes streams at a sampling frequency of 10 streams/second, MQTT protocol again obtains the lowest jitter value, being at least 1.395 times lower than the rest of the protocols.

Figure 6 compares the IIoT messaging protocols performance for 4Mbytes/s throughput edge/fog experiments in mean and standard deviation latency terms.

Sending 100 Kbytes streams at a frequency of 40 streams/s, with ZeroMQ protocol, is the scenario with the lowest latency, closely followed by AMQP. When employing 400 Kbytes streams at a sampling frequency of 10 streams/second, MQTT protocol again is one that obtains the lowest latency, being at least 1.395 times faster than the rest of the protocols.

Figure 7 compares the IIoT messaging protocols performance for 4Mbytes/s throughput edge/fog experiments in mean and standard deviation jitter terms.

Sending 100 Kbytes streams at a frequency of 40 streams/s, with AMQP protocol, is the scenario with the lowest jitter, closely followed by KAFKA, MQTT, and ZeroMQ protocols. When employing 400 Kbytes streams at a sampling frequency of 10 streams/second, AMQP protocol again obtains the lowest latency, being at least 2.17 times lower than the rest of the protocols.

Figure 8 compares the IIoT messaging protocols performance for 8Mbytes/s throughput edge/fog experiments in mean and standard deviation latency terms.

The scenario with lowest latency is sending 200 Kbytes streams at a frequency of 40 streams/s, with any of AMQP, or ZEROMQ protocols. However, under that scenario, AMQP performs 2.65 times better in latency standard deviation terms than ZeroMQ. When employing 800 Kbytes streams at a sampling frequency of 10 streams/second, AMQP protocol has the lowest latency. While latency values for that combination of AMQP, MQTT, and ZeroMQ protocols is similar, the standard deviation of MQTT and ZeroMQ protocols are 4.83 and 6.13 times higher, respectively.

Figure 9 compares the IIoT messaging protocols performance for 8Mbytes/s throughput edge/fog experiments in mean and standard deviation jitter terms.

Figure 6. Normalized Latency of 4 MBytes/s Edge/Fog Experiments

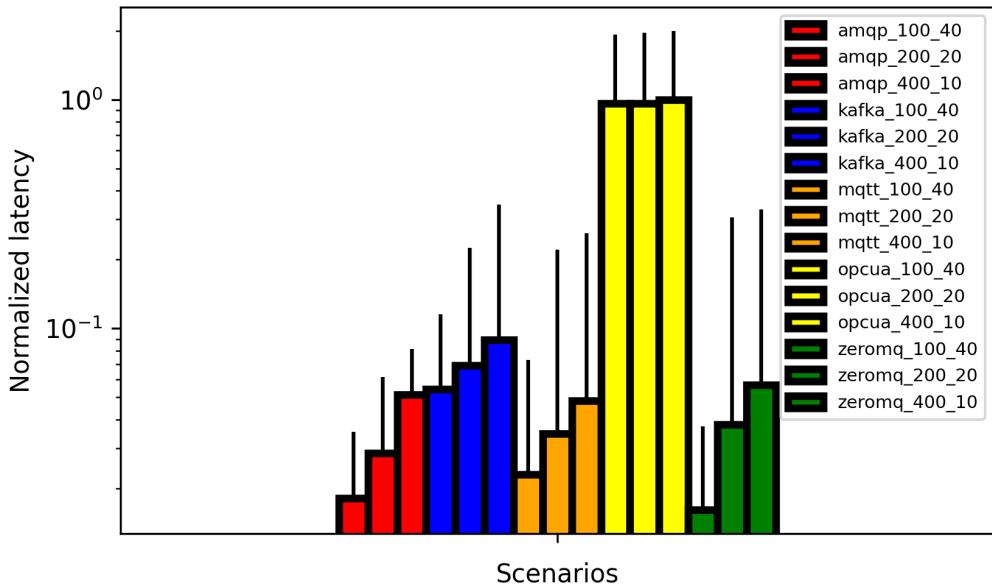
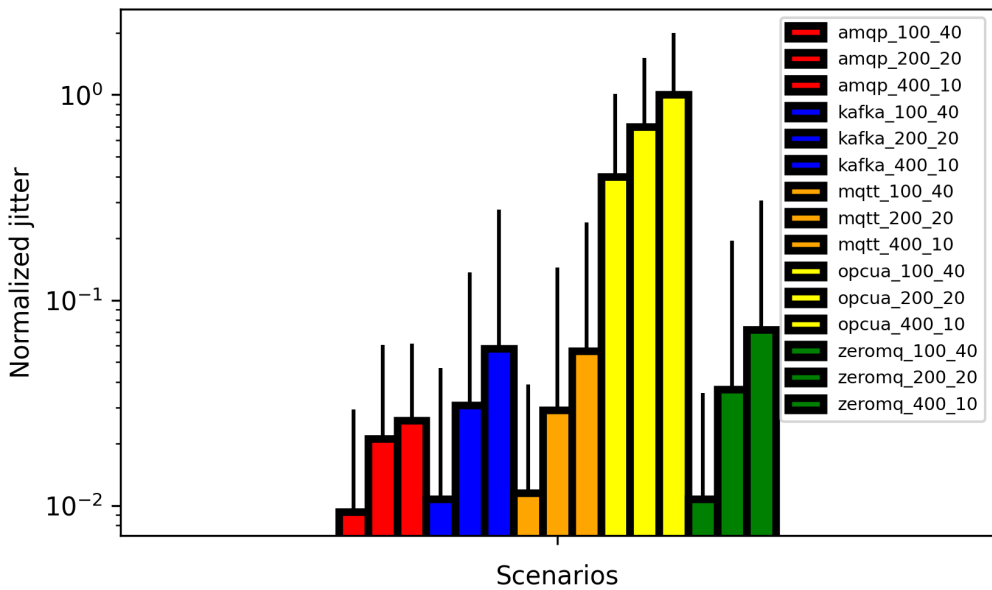


Figure 7. Normalized Jitter of 4 MBytes/s Edge/Fog Experiments



Sending 200 Kbytes streams at a frequency of 40 streams /s, with ZeroMQ and AMQP protocols, are the scenarios that obtain the lowest jitter. ZeroMQ performs 1.13 times better than MQTT in terms of jitter under that scenario. When employing 800 Kbytes streams at a sampling frequency of 10 streams/second, AMQP protocol clearly obtains the lowest jitter, being at least 2.53 times lower than the rest of the protocols.

Figure 8. Normalized Latency of 8 MBytes/s Edge/Fog Experiments

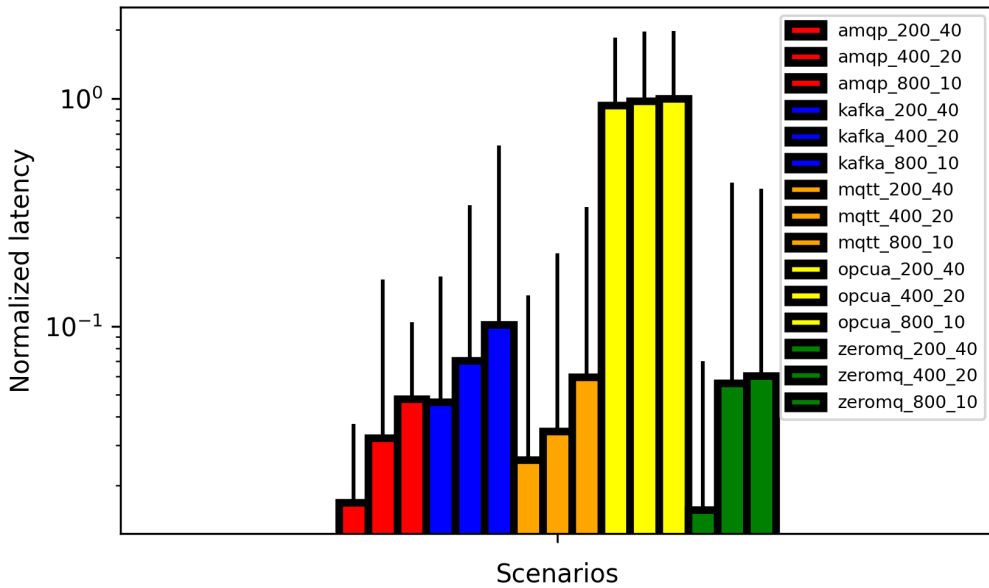
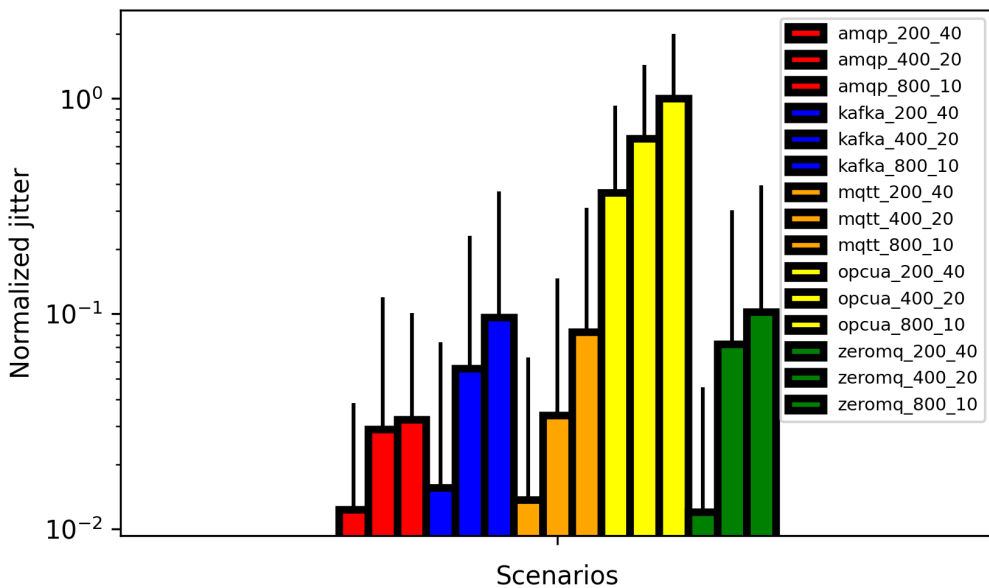


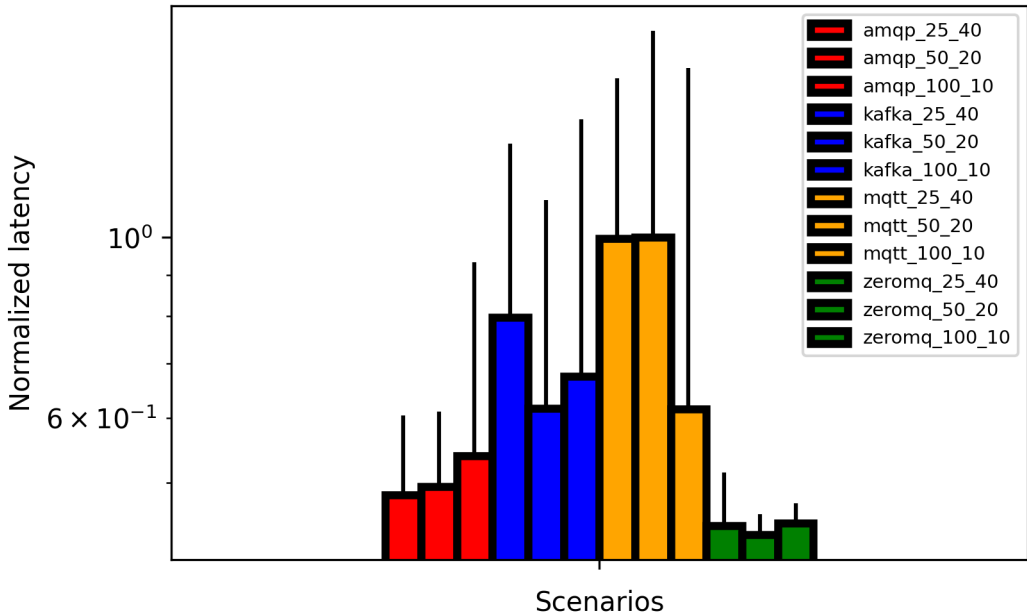
Figure 9. Normalized Jitter of 8 MBytes/s Edge/Fog Experiments



### Cloud Experiment Results

Figure 10 compares the IIoT messaging protocols performance for 1Mbytes/s throughput cloud experiments in mean and standard deviation latency terms. To clearly visualize the results, just the positive values of the standard deviation are represented as vertical black lines just above the bars, not showing the negative side of them. The legend of the figure is composed of the protocol name, the

Figure 10. Normalized Latency of 1 MBytes/s Cloud Experiments



stream size value in Kbytes, and the stream frequency. Each of the protocols is shown in a different color. All the figures from this point on will follow the same structure.

ZeroMQ is the protocol that obtains lower latency values, being the most stable one. Moreover, it is indifferent to the combination of stream size and rate. AMQP protocol, which is the next protocol obtaining lower latency, obtains at least 1.12 and 4.48 times worse results in mean and standard deviation latency terms.

Figure 11 compares the IIoT messaging protocols performance for 1Mbytes/s throughput cloud experiments in mean and standard deviation jitter terms.

Sending 50 Kbytes streams at a frequency of 20 streams /s, with ZeroMQ protocol, is the scenario that obtains lower jitter value, performing 1.54 times better than the next best combination, 100 Kbytes streams at a sampling frequency of 10 streams/second with ZeroMQ.

Figure 12 compares the IIoT messaging protocols performance for 4Mbytes/s throughput cloud experiments in mean and standard deviation latency terms.

The scenario with lowest latency is sending 100 Kbytes streams at a frequency of 40 streams /s, with ZeroMQ protocol. However, the difference with the rest of the scenarios employing ZeroMQ is significantly low. Under the 4 Mbytes/s use case, the difference between ZeroMQ and the rest of the protocols is higher than in the 1 Mbyte/second use case. In mean and standard deviation terms, ZeroMQ obtains at least 5.14 and 114.05 times lower results than AMQP, respectively. Kafka is clearly the protocol with worse latency, indicating that it is not suitable for high-frequency scenarios.

Figure 13 compares the IIoT messaging protocols performance for 4Mbytes/s throughput cloud experiments in mean and standard deviation jitter terms.

Sending 100 Kbytes streams at a frequency of 40 streams/s, with ZeroMQ protocol, is the scenario that obtains the lowest jitter results. Moreover, ZeroMQ performs better than the rest of the protocols employing any of the combinations. It performs at least 8.83 and 6.96 times better than AMQP in mean and standard deviation jitter terms.

Figure 14 compares the IIoT messaging protocols performance for 8Mbytes/s throughput cloud experiments in mean and standard deviation latency terms.

Figure 11. Normalized Jitter of 1 MBytes/s Cloud Experiments

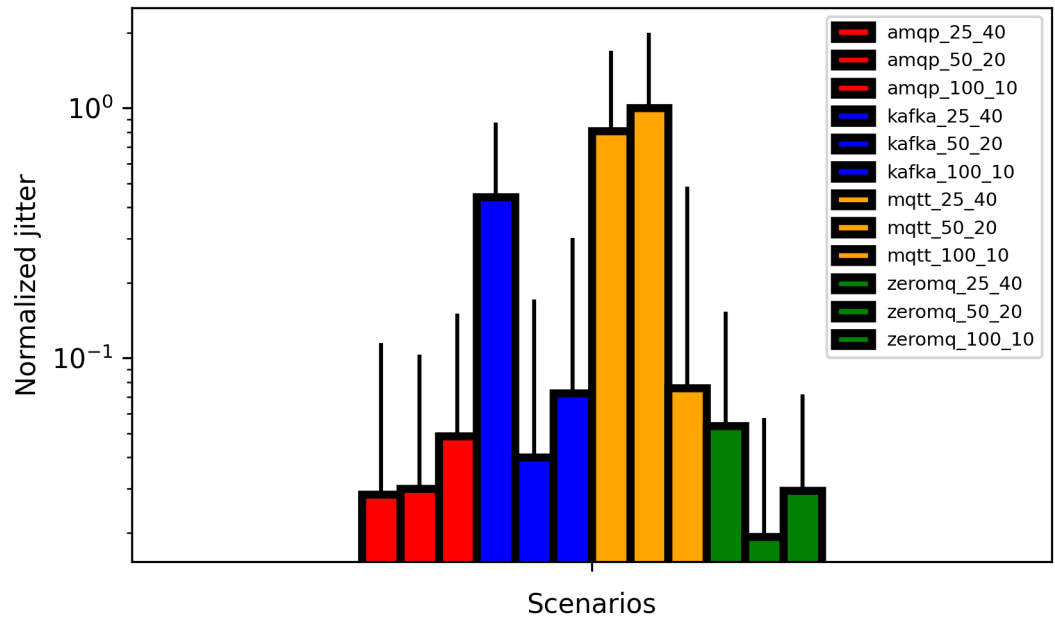
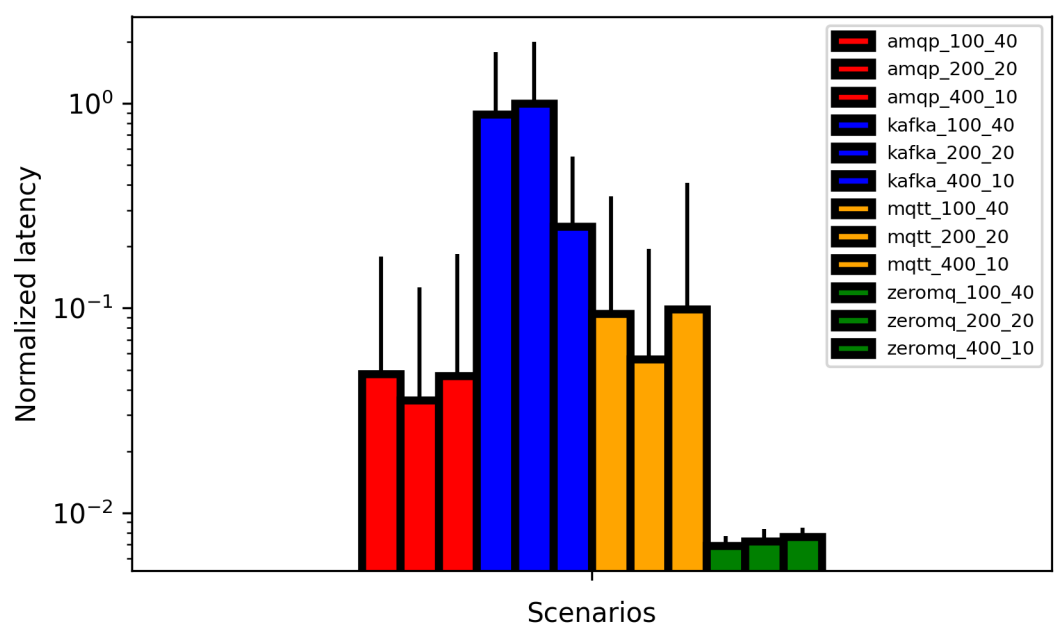


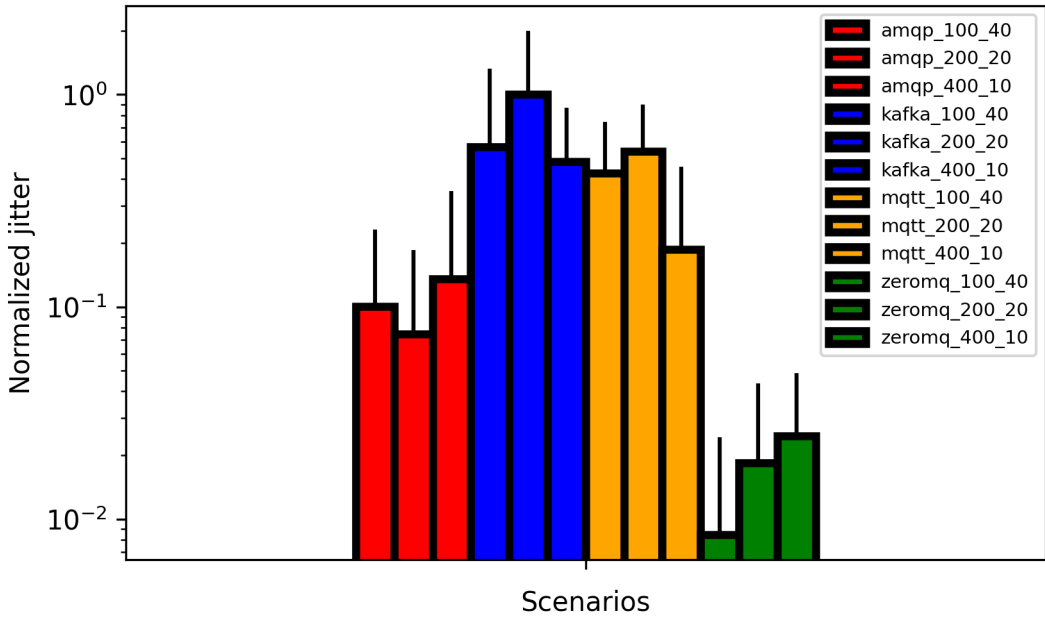
Figure 12. Normalized Latency of 4 MBytes/s Cloud Experiments



The scenario that obtains the lowest jitter is sending 200 Kbytes streams at a frequency of 40 streams/s ZEROMQ protocol. However, the difference with the rest of the scenarios employing ZeroMQ is significantly low. It performs at least 5.46 and 160.88 times better than AMQP in mean



Figure 13. Normalized Jitter of 4 MBytes/s Cloud Experiments



and standard deviation latency terms. Kafka is clearly the protocol that performs worse, indicating again that it is not suitable for high-frequency scenarios.

Figure 15 compares the IIoT messaging protocols performance for 8Mbytes/s throughput cloud experiments in mean and standard deviation jitter terms.

Figure 14. Normalized Latency of 8 MBytes/s Cloud Experiments

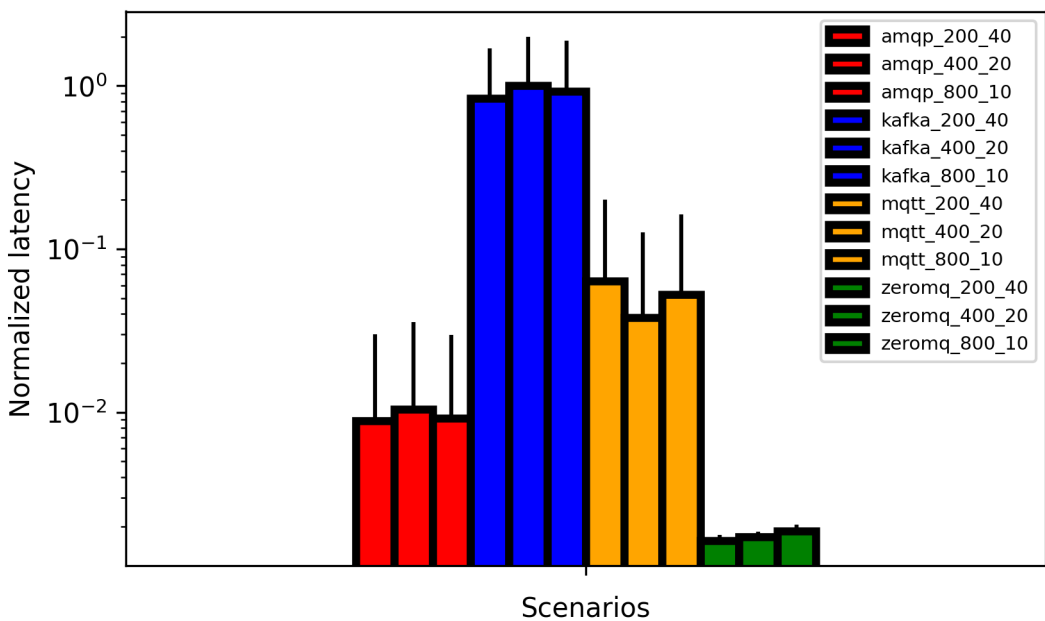
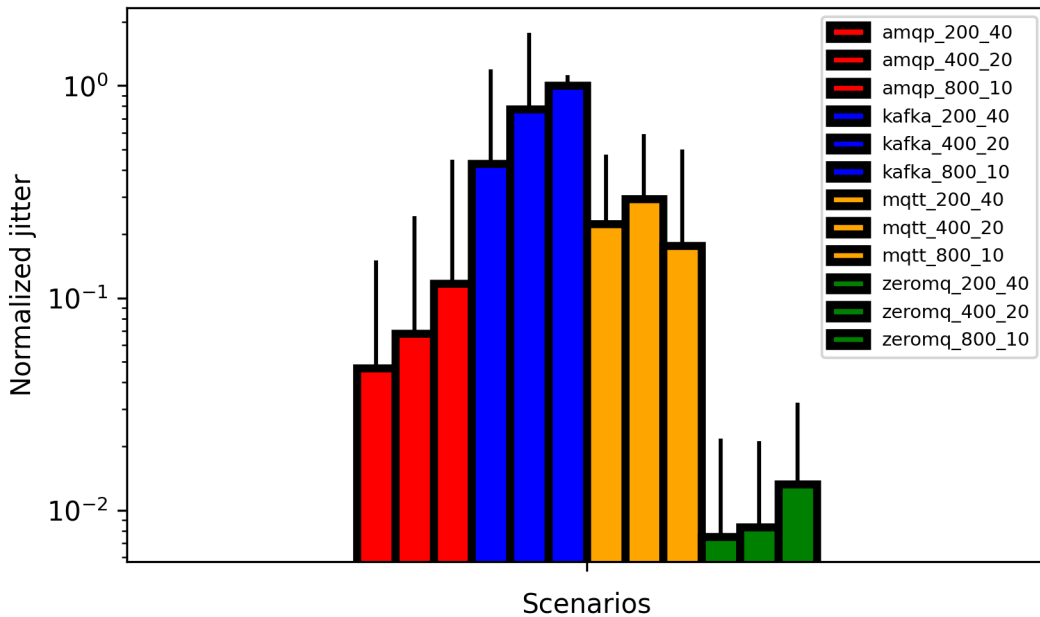


Figure 15. Normalized Jitter of 8 MBytes/s Cloud Experiments



Sending 200 Kbytes streams at a frequency of 40 streams/s with ZeroMQ is the scenario that obtains the lowest jitter. It performs at least 6.23 and 8.03 times better than AMQP in mean and standard deviation jitter terms.

### Computational Overhead Results

Table 3 presents the memory usage per service (MB) in the edge/fog/cloud device without data traffic. For broker-centric protocols, MQTT is the lighter one, employing 118.45 and 424.36 times less memory than AMQP and KAFKA brokers, respectively. For the AI service, all the protocols except OP CUA employ a similar amount of memory ZeroMQ being the lighter one. The overhead of the OP CUA client is due to the way in which it has been implemented, as the server is generating that overhead.

Table 4 presents the CPU usage per service, in % terms, in the edge/fog/cloud device without data traffic. For broker-centric protocols, MQTT is most efficient one, while the Kafka broker is the most demanding one. For the AI service, MQTT, AMQP, and ZeroMQ employ the lowest CPU.

Table 3. Memory Usage per Service (MB) in the Edge Device Without Data Traffic

	Broker	AI_service
MQTT	1.21	89.95
AMQP	143.33	90.87
KAFKA	513.48	90.62
ZeroMQ	-	88.82
OP CUA	-	146.85

Table 4. CPU Usage per Service (%) in the Edge Device Without Data Traffic

	Broker	AI_service
MQTT	0.00%	0.00%
AMQP	0.51%	0.00%
KAFKA	1.40%	0.02%
ZeroMQ	-	0.00%
OP CUA	-	0.06%

Table 5 presents the memory usage per service (MB) in the IIoT and edge/fog/cloud devices with data traffic. For broker-centric protocols, MQTT is the lighter one, while KAFKA is the most memory-demanding one. For the AI service, all the protocols except OP CUA employ a similar amount of memory, ZeroMQ being the lightest one. Regarding the IIoT producer service, MQTT, AMQP, and ZeroMQ protocols' clients employ a similar amount of memory, with KAFKA and OP CUA protocols' clients being heavier. Regarding the IIoT consumer service, MQTT, AMQP, KAFKA, and ZeroMQ protocols' clients employ a similar amount of memory, with OP CUA client being heavier.

Tables 6 and 7 present the CPU mean and maximum usage per service, in % terms, in the IIoT and edge/fog/cloud devices with data traffic. For broker-centric protocols, MQTT is most efficient one, employing 10 times less CPU than AMQP in mean terms. For the AI service, ZeroMQ is the one that employs the lowest CPU. Regarding the IIoT producer service, there are no significant differences between the performance among the different protocols. Regarding the IIoT consumer service, AMQP is the least efficient one and ZeroMQ the most efficient one.

Finally, table 8 provides guidelines for which of the protocols should be used in each of the scenarios. For a fixed throughput, server location, and objective, the best scenarios are colored green.

## CONCLUSION

In this paper, the differences between five different IIoT messaging protocols, AMQP, MQTT, KAFKA, ZeroMQ, and OP CUA, for high frequency data transmission in edge/fog and cloud scenarios, was investigated. Mean and standard deviation latency, mean and standard deviation jitter, lost or not-ordered packages, etc. have been analyzed under different working conditions, varying the throughput (from 1 to 8 Mbytes per second), sampling frequency, and stream sizes. Moreover, the computational overhead of each of them was analyzed.

Table 5. Memory Usage per Service (MB) With Data Traffic (100Kbytes Stream Size, 10 Streams/s Conditions)

	Broker	AI_serivce	IIoT_producer	IIoT_consumer
MQTT	1.45	93.06	102.97	103.57
AMQP	147.04	92.40	107.51	108.92
KAFKA	613.92	93.38	205.59	106.54
ZeroMQ	-	91.28	104.79	103.84
OP CUA	-	157.80	157.91	116.21

Table 6. Mean CPU Usage per Service (%) With Data Traffic (100Kbytes Stream Size, 10 Streams/s Conditions)

	Broker	AI_service	IIoT_producer	IIoT_consumer
MQTT_mean	0.09%	0.91%	10.70%	0.21%
AMQP_mean	0.99%	1.18%	11.40%	0.75%
KAFKA_mean	5.18%	0.95%	11.25%	0.56%
ZeroMQ_mean	-	0.72%	10.34%	0.12%
OP_CUA_mean	-	2.61%	11.88%	0.57%

Table 7. Maximum CPU Usage per Service (%) With Data Traffic (100Kbytes Stream Size, 10 Streams/s Conditions)

	Broker	AI_service	IIoT_producer	IIoT_consumer
MQTT_max	0.26%	2.59%	18.13%	0.68%
AMQP_max	1.70%	3.16%	18.03%	2.32%
KAFKA_max	7.90%	2.17%	10.32%	2.07%
ZeroMQ_max	-	1.99%	16.68%	0.34%
OP_CUA_max	-	4.75%	17.58%	1.54%

In edge/fog computing scenarios, AMQP, MQTT, and ZeroMQ are the protocols that show better performance. For experiments with the same throughput, the combination of smaller data streams at higher frequencies yields optimal performance across all three protocols. The three of them are suitable for soft-real-time scenarios, being able to handle 8 Mbytes/s data throughput. This can be translated into employing 1 ultrasonic sensor at 1MHz and 20 accelerometers working at 50kHz at the same time, acquiring 4 bytes data points. Particularly, for smaller data streams with higher frequencies, all three protocols perform identically. However, for larger data streams employing smaller sampling rates, AMQP consistently outperforms MQTT and ZEROMQ, emerging as the best choice, especially in jitter terms. It can be concluded that in this scenario, having a broker-centric architecture does not affect the performance of the applications. Moreover, in edge/fog scenarios where fewer CPU resources are usually available, it is advisable to use lighter protocols such as MQTT and ZeroMQ.

For cloud computing scenarios, the results provide valuable insights into the performance of various protocols. ZeroMQ and AMQP are the protocols that show better performance.

For lower (1 Mbyte/sec) throughput experiments, ZeroMQ exhibits slightly lower mean latency, but significantly higher stability over time in terms of latency standard deviation. As throughput increases, the differences between ZeroMQ and the rest of the protocols increases.

For higher (4 Mbyte/sec and 8 Mbyte/sec) throughput, ZeroMQ consistently outperforms the rest of the protocols in all aspects, showing significantly lower standard deviation in latency standard deviation. The best combination is sending smaller streams at a higher sampling frequency, demonstrating superior jitter performance compared to larger streams at lower frequencies. It can be concluded that in this scenario, having a broker-centric architecture affects the performance of the applications. On the other hand, the results indicate that ZeroMQ is appropriate for building soft real-time edge/fog computing applications for industry.

Future work will begin validating the presented results, working with different libraries for implementing the protocols, such as Redpanda for Kafka and QuickOPC for OP\_CUA pub/sub.

Table 8. Which Protocol, Stream Size, and Rate Combination to Use in Different Scenarios

Throughput (Mbytes/s)	Server location	Objective	Stream size (Kbytes)	Stream rate (streams/s)	Selected protocol
1	Edge/fog	Latency	<25	>40	AMQP, MQTT, ZEROMQ
			>100	<10	MQTT
		Jitter	<25	>40	MQTT
			>100	<10	MQTT
	Cloud	Latency	<25	>40	ZeroMQ
			>100	<10	ZeroMQ
		Jitter	<25	>40	ZeroMQ
			>100	<10	ZeroMQ
4	Edge/fog	Latency	<100	>40	AMQP, ZeroMQ
			>400	<10	MQTT
		Jitter	<100	>40	AMQP
			>400	<10	AMQP
	Cloud	Latency	<100	>40	ZeroMQ
			>400	<10	ZeroMQ
		Jitter	<100	>40	ZeroMQ
			>400	<10	ZeroMQ
8	Edge/fog	Latency	<200	>40	AMQP, ZEROMQ
			>800	<10	AMQP
		Jitter	<200	>40	AMQP, ZEROMQ
			>800	<10	AMQP
	Cloud	Latency	<200	>40	ZeroMQ
			>800	<10	ZeroMQ
		Jitter	<200	>40	ZeroMQ
			>800	<10	ZeroMQ

Moreover, the results will be validated under a real industrial scenario, acquiring high-frequency signals and processing them in real time. Thus, the IIoT simulator will be replaced by a data acquisition device, acquiring high-frequency signals from different sources such as accelerometers, acoustic emissions sensors, and energy meters. The AI simulator will be replaced by a real data-based model that will be deployed at the edge/fog/cloud, and the inference of the model will be employed for sending orders to a machine and generating alerts.

### AUTHOR CONTRIBUTIONS

Conceptualization: Ander García and Telmo Fernández de Barrena; Methodology: Telmo Fernández de Barrena; Software: Telmo Fernández de Barrena; Validation: Telmo Fernández De Barrena; Formal Analysis: Telmo Fernández De Barrena; Investigation: Telmo Fernández de Barrena; Data Curation: Telmo Fernández de Barrena; Writing—Original Draft Preparation: Telmo Fernández de Barrena; Writing—Review And Editing: Ander García, Juan Luis Ferrando, and Telmo Fernández de Barrena; Visualization: Telmo Fernández de Barrena; Supervision: Ander García and Juan Luis Ferrando; Project Administration: Ander García.

### FUNDING

This research was partially funded by the Department of Industry of the Basque Government within Elkartek programs (KK-2022/00119 and KK-2023/00038).

## **DATA AVAILABILITY STATEMENT**

The code employed for replicating the experiments and data presented in this study are available at <https://github.com/Vicomtech/IIOT-protocols-study-for-high-frequency-data-in-the-edge-and-cloud/tree/main>

## **CONFLICTS OF INTEREST**

The authors declare no conflicts of interest.

## REFERENCES

- Alshehri, F., & Muhammad, G. (2021). A Comprehensive survey of the internet of things (IoT) and AI-based smart healthcare. *IEEE Access : Practical Innovations, Open Solutions*, 9, 3660–3678. doi:10.1109/ACCESS.2020.3047960
- Apache Software Foundation. (2023). Apache Kafka. <https://kafka.apache.org>
- AQMP. (2023). *AQMP is the internet protocol for business messaging*. <https://www.amqp.org/about/what>
- Bayılmış, C., Ebleme, M. A., Çavuşoğlu, Ü., Küçük, K., & Sevin, A. (2022). A survey on communication protocols and performance evaluations for Internet of Things. *Digital Communications and Networks*, 8(6), 1094–1104. doi:10.1016/j.dcan.2022.03.013
- Chaudhary, A., Peddoju, S. K., & Kadarla, K. (2017). Study of Internet-of-Things messaging protocols used for exchanging data with external sources. In *2017 IEEE 14th International Conference on Mobile Ad Hoc and Sensor Systems (MASS)* (pp. 666–671). IEEE. doi:10.1109/MASS.2017.85
- Dizdarević, J., Carpio, F., Jukan, A., & Masip-Bruin, X. (2019). A survey of communication protocols for Internet of Things and related challenges of fog and cloud computing integration. *ACM Computing Surveys*, 51(6), 1–29. doi:10.1145/3292674
- Fernández de Barrena, T., Ferrando, J. L., García, A., Badiola, X., de Buruaga, M. S., & Vicente, J. (2023). Tool remaining useful life prediction using bidirectional recurrent neural networks (BRNN). *International Journal of Advanced Manufacturing Technology*, 125(9–10), 4027–4045. doi:10.1007/s00170-023-10811-9
- Foundation, O. P. C. *Unified Architecture* (2023). <https://opcfoundation.org/about/opc-technologies/opc-ua/>
- Garcia, A., Fernández de Barrena, T., Chacón, J. L. F., Oregui, X., & Etxegoin, Z. (2023). Edge architecture for the integration of soft models based industrial AI control into Industry 4.0 cyber-physical systems. *Lecture Notes in Networks and Systems*, 750 LNNS, 67–76. 10.1007/978-3-031-42536-3\_7
- Gavrilov, A., Bergaliyev, M., Tinyakov, S., Krinkin, K., & Popov, P. (2022). Using IoT protocols in real-time systems: Protocol analysis and evaluation of data transmission characteristics. *Journal of Computer Networks and Communications*, 2022, 1–18. Advance online publication. doi:10.1155/2022/7368691
- Hegazy, T., & Hefeeda, M. (2015). Industrial automation as a cloud service. *IEEE Transactions on Parallel and Distributed Systems*, 26(10), 2750–2763. doi:10.1109/TPDS.2014.2359894
- Imtiaz, J., & Jasperneite, J. (2013). Scalability of OPC-UA down to the chip level enables “Internet of Things.” In *2013 11th IEEE International Conference on Industrial Informatics (INDIN)* (pp. 500–505). IEEE. doi:10.1109/INDIN.2013.6622935
- Ionescu, V. M. (2015). The analysis of the performance of RabbitMQ and ActiveMQ. In *2015 14th RoEduNet International Conference - Networking in Education and Research (RoEduNet NER)* (pp. 132–137). IEEE. doi:10.1109/RoEduNet.2015.7311982
- Kang, Z., Canaday, R., Dubey, A., Gokhale, A., Shekhar, S., & Selacek, M. (2020). Evaluating DDS, MQTT, and ZeroMQ Under Different IoT Traffic Conditions. In *M4IoT '20: Proceedings of the International Workshop on Middleware and Applications for the Internet of Things* (pp. 7–12). Association for Computing Machinery. doi:10.1145/3429881.3430109
- Kumar, R., & Agrawal, N. (2023). Analysis of multi-dimensional Industrial IoT (IIoT) data in Edge–Fog–Cloud based architectural frameworks: A survey on current state and research
- Kuntoğlu, M., Salur, E., Gupta, M. K., Sarıkaya, M., & Pimenov, D. Yu. (2021). A state-of-the-art review on sensors and signal processing systems in mechanical machining processes. *International Journal of Advanced Manufacturing Technology*, 116(9–10), 2711–2735. doi:10.1007/s00170-021-07425-4 PMID:34092883
- Lazidis, A., Tsakos, K., & Petrakis, E. G. M. (2022). Publish–Subscribe approaches for the IoT and the cloud: Functional and performance evaluation of open-source systems. *Internet of Things : Engineering Cyber Physical Human Systems*, 19, 100538. doi:10.1016/j.iot.2022.100538
- Lecun, Y., Bengio, Y., & Hinton, G. (2015). Deep learning. *Nature*, 521(7553), 436–444. doi:10.1038/nature14539 PMID:26017442

Lombardi, M., Pascale, F., & Santaniello, D. (2021). Internet of Things: A general overview between architectures, protocols and applications. *Information (Basel)*, 12(2), 87. doi:10.3390/info12020087

Mao, S., He, S., & Wu, J. (2021). Joint UAV position optimization and resource scheduling in space-air-ground integrated networks with mixed cloud-edge computing. *IEEE Systems Journal*, 15(3), 3992–4002. doi:10.1109/JSYST.2020.3041706

Mao, S., Wu, J., Liu, L., Lan, D., & Taherkordi, A. (2022). Energy-efficient cooperative communication and computation for wireless powered mobile-edge computing. *IEEE Systems Journal*, 16(1), 287–298. doi:10.1109/JSYST.2020.3020474

MQTT. *MQTT: The Standard for IoT Messaging*. (2022). <https://mqtt.org/>

Naik, N. (2017). Choice of effective messaging protocols for IoT systems: MQTT, CoAP, AMQP and HTTP. In *2017 IEEE International Systems Engineering Symposium (ISSE)* (pp. 1–7). IEEE. doi:10.1109/SysEng.2017.8088251

Nam, J., Jun, Y., & Choi, M. (2022). High performance IoT cloud computing framework using Pub/Sub techniques. *Applied Sciences (Basel, Switzerland)*, 12(21), 11009. doi:10.3390/app122111009

Nath, S. V., Dunkin, A., Chowdhary, M., & Patel, N. (2020). *Industrial Digital Transformation*. Packt Publishing. <https://books.google.es/books?hl=es&lr=&id=26EGEAAAQBAJ&oi=fnd&pg=PP1&dq=Digital+Transformation+in+Manufacturing:+Gaining+a+Competitive+Edge+Through+IoT+and+the+AI&ots=Hfs8q4vPYR&sig=IIsOQ35sG7HJQ7zkrBf6cZWRgyY#v=onepage&q&f=false>

Oñate, W., & Sanz, R. (2023). Analysis of architectures implemented for IIoT. *Heliyon*, 9(1), e12868. doi:10.1016/j.heliyon.2023.e12868 PMID:36691530

Profanter, S., Tekat, A., Dorofeev, K., Rickert, M., & Knoll, A. (2019). OPC UA versus ROS, DDS, and MQTT: Performance evaluation of industry 4.0 protocols. In *Proceedings of the 2019 IEEE International Conference on Industrial Technology* (pp. 955–962). IEEE. doi:10.1109/ICIT.2019.8755050

Sarafov, V. (2018). Comparison of IoT Data Protocol Overhead. *Proceedings of the Seminars Future Internet (FI) and Innovative Internet Technologies and Mobile Communications (IITM) Scimago*, 7–14. doi:10.2313/NET-2018-03-1\_02

Schleipen, M., Gilani, S.-S., Bischoff, T., & Pfrommer, J. (2016). OPC UA & Industrie 4.0—Enabling technology with high diversity and variability. *Procedia CIRP*, 57, 315–320. doi:10.1016/j.procir.2016.11.055

Shi, W., Cao, J., Zhang, Q., Li, Y., & Xu, L. (2016). Edge computing: Vision and challenges. *IEEE Internet of Things Journal*, 3(5), 637–646. doi:10.1109/JIOT.2016.2579198

Suri, N., Breedy, M. R., Marcus, K. M., Fronteddu, R., Cramer, E., Morelli, A., Campioni, L., Provosty, M., Enders, C., Tortonesi, M., & Nilsson, J. (2019). Experimental Evaluation of group communications protocols for data dissemination at the tactical edge. In *2019 International Conference on Military Communications and Information Systems (ICMCIS)* (pp. 1–8). IEEE. doi:10.1109/ICMCIS.2019.8842801

Ullah, M., Kakakhel, S. R. U., Westerlund, T., Wolff, A., Carrillo, D., Plosila, J., & Nardelli, P. H. J. (2020). IoT protocol selection for smart grid applications: Merging qualitative and quantitative metrics. In *2020 43rd International Convention on Information, Communication and Electronic Technology, MIPRO 2020 - Proceedings* (pp. 993–998). IEEE. doi:10.23919/MIPRO48935.2020.9245238

Zero MQ, *ZeroMQ*. (2023). <https://zeromq.org>



**APPENDIX**

**Table 9. Latency, Latency Standard Deviation, Jitter, and Jitter Standard Deviation Absolute (ms) Values Under 1 Mbyte/s Edge/Fog Computing Scenario**

Stream size_rate	AMQP			KAFKA			MQTT			OP CUA			ZEROMQ		
	25_40	50_20	100_10	25_40	50_20	100_10	25_40	50_20	100_10	25_40	50_20	100_10	25_40	50_20	100_10
latency	4.099	5.575	11.735	14.090	15.089	20.608	3.643	4.391	6.483	218.682	232.840	244.391	3.919	5.078	7.805
latency_stdev	1.481	1.820	2.813	1.400	3.641	21.971	1.404	2.475	3.550	86.084	86.103	79.000	1.250	3.744	11.246
jitter	0.987	1.312	2.992	0.925	1.681	5.387	0.831	1.230	2.146	43.960	75.854	103.338	1.036	1.773	4.316
jitter_stdev	1.946	2.535	3.868	1.970	5.159	21.781	1.913	3.483	4.615	68.191	90.324	105.897	1.736	5.278	15.799

**Table 10. Latency, Latency Standard Deviation, Jitter, and Jitter Standard Deviation Normalized Values Under 1 Mbyte/s Edge/Fog Computing Scenario**

Stream size_rate	AMQP			KAFKA			MQTT			OP CUA			ZEROMQ		
	25_40	50_20	100_10	25_40	50_20	100_10	25_40	50_20	100_10	25_40	50_20	100_10	25_40	50_20	100_10
latency	0.017	0.023	0.048	0.058	0.062	0.084	0.015	0.018	0.027	0.895	0.953	1.000	0.016	0.021	0.032
latency_stdev	0.017	0.021	0.033	0.016	0.042	0.255	0.016	0.029	0.041	1.000	1.000	0.917	0.015	0.043	0.131
jitter	0.010	0.013	0.029	0.009	0.016	0.052	0.008	0.012	0.021	0.425	0.734	1.000	0.010	0.017	0.042
jitter_stdev	0.018	0.024	0.037	0.019	0.049	0.206	0.018	0.033	0.044	0.644	0.853	1.000	0.016	0.050	0.149

Table 11. Latency, Latency Standard Deviation, Jitter, and Jitter Standard Deviation Absolute (ms) Values Under 4 Mbyte/s Edge/Fog Computing Scenario

Stream size_rate	AMQP			KAFKA			MQTT			OP CUA			ZEROMQ		
	100_40	200_20	200_20	100_40	200_20	200_20	100_40	200_20	200_20	100_40	200_20	200_20	100_40	200_20	200_20
latency	5.133	8.070	14.558	15.377	19.491	<b>25.253</b>	6.506	9.823	13.710	272.192	273.090	<b>282.953</b>	4.563	10.770	16.005
latency_stddev	1.777	3.423	3.087	6.363	16.219	<b>26.896</b>	5.171	19.394	22.122	100.168	103.817	<b>103.730</b>	2.197	27.761	28.555
jitter	1.039	2.352	2.904	1.199	3.422	<b>6.459</b>	1.284	3.243	6.298	44.469	77.685	<b>111.409</b>	1.203	4.101	8.004
jitter_stddev	2.462	4.808	4.314	4.393	12.962	<b>26.567</b>	3.354	14.070	22.289	74.426	99.341	<b>122.306</b>	2.993	19.242	28.658

Table 12. Latency, Latency Standard Deviation, Jitter, and Jitter Standard Deviation Normalized Values Under 4 Mbyte/s Edge/Fog Computing Scenario

Stream size_rate	AMQP			KAFKA			MQTT			OP CUA			ZEROMQ		
	100_40	200_20	400_10	100_40	200_20	400_10	100_40	200_20	400_10	100_40	200_20	400_10	100_40	200_20	400_10
latency	0.018	0.029	0.051	0.054	0.069	0.089	0.023	0.035	0.048	0.962	0.965	1.000	0.016	0.038	0.057
latency_stddev	0.017	0.033	0.030	0.061	0.156	0.259	0.050	0.187	0.213	0.965	1.000	0.999	0.021	0.267	0.275
jitter	0.009	0.021	0.026	0.011	0.031	0.058	0.012	0.029	0.057	0.399	0.697	1.000	0.011	0.037	0.072
jitter_stddev	0.020	0.039	0.035	0.036	0.106	0.217	0.027	0.115	0.182	0.609	0.812	1.000	0.024	0.157	0.234

**Table 13. Latency, Latency Standard Deviation, Jitter, and Jitter Standard Deviation Absolute (ms) Values Under 8 Mbyte/s Edge/Fog Computing Scenario**

Stream size_rate	AMQP			KAFKA			MQTT			OP CUA			ZEROMQ		
	200_40	400_20	800_10	200_40	400_20	800_10	200_40	400_20	800_10	200_40	400_20	800_10	200_40	400_20	800_10
latency	6.577	12.623	18.779	18.198	27.654	39.864	10.121	13.510	23.478	365.468	381.747	392.335	6.117	22.048	23.727
latency_stddev	2.721	17.025	7.444	15.886	35.951	69.294	14.740	23.279	36.466	121.800	133.097	130.955	7.238	49.328	45.539
jitter	1.522	3.599	3.978	1.922	6.877	11.897	1.687	4.176	10.166	45.047	80.767	123.561	1.483	8.932	12.616
jitter_stddev	3.802	13.106	10.028	8.453	25.492	39.786	7.138	16.317	33.305	81.716	113.671	145.647	4.878	33.548	42.603

**Table 14. Latency, Latency Standard Deviation, Jitter, and Jitter Standard Deviation Normalized Values Under 8 Mbyte/s Edge/Fog Computing Scenario**

Stream size_rate	AMQP			KAFKA			MQTT			OP CUA			ZEROMQ		
	200_40	400_20	800_10	200_40	400_20	800_10	200_40	400_20	800_10	200_40	400_20	800_10	200_40	400_20	800_10
latency	0.017	0.032	0.048	0.046	0.070	0.102	0.026	0.034	0.060	0.932	0.973	1.000	0.016	0.056	0.060
latency_stddev	0.020	0.128	0.056	0.119	0.270	0.521	0.111	0.175	0.274	0.915	1.000	0.984	0.054	0.371	0.342
jitter	0.012	0.029	0.032	0.016	0.056	0.096	0.014	0.034	0.082	0.365	0.654	1.000	0.012	0.072	0.102
jitter_stddev	0.026	0.090	0.069	0.058	0.175	0.273	0.049	0.112	0.229	0.561	0.780	1.000	0.033	0.230	0.293

**Table 15. Latency, Latency Standard Deviation, Jitter, and Jitter Standard Deviation Absolute (ms) Values Under 1 Mbyte/s Cloud Computing Scenario**

Stream size_rate	AMQP			KAFKA			MQTT			ZEROMQ		
	25_40	50_20	100_10	25_40	50_20	100_10	25_40	50_20	100_10	25_40	50_20	100_10
latency	47.536	48.676	53.132	78.688	60.760	66.518	98.278	98.552	60.613	43.572	42.509	43.917
latency_stddev	6.890	6.589	22.129	28.496	27.828	40.661	32.172	44.759	56.284	4.053	1.470	1.473
jitter	1.275	1.347	2.180	19.749	1.797	3.242	36.259	44.861	3.414	2.405	0.865	1.321
jitter_stddev	3.998	3.386	4.732	20.264	6.087	10.643	40.954	46.496	18.926	4.623	1.782	1.953

**Table 16. Latency, Latency Standard Deviation, Jitter, and Jitter Standard Deviation Normalized Values Under 1 Mbyte/s Cloud Computing Scenario**

Stream size_rate	AMQP			KAFKA			MQTT			ZEROMQ		
	25_40	50_20	100_10	25_40	50_20	100_10	25_40	50_20	100_10	25_40	50_20	100_10
latency	0.482	0.494	0.539	0.798	0.617	0.675	0.997	1.000	0.615	0.442	0.431	0.446
latency_stddev	0.122	0.117	0.393	0.506	0.494	0.722	0.572	0.795	1.000	0.072	0.026	0.026
jitter	0.028	0.030	0.049	0.440	0.040	0.072	0.808	1.000	0.076	0.054	0.019	0.029
jitter_stddev	0.086	0.073	0.102	0.436	0.131	0.229	0.881	1.000	0.407	0.099	0.038	0.042

**Table 17. Latency, Latency Standard Deviation, Jitter, and Jitter Standard Deviation Absolute (ms) Values Under 4 Mbyte/s Cloud Computing Scenario**

Stream size_rate	AMQP			KAFKA			MQTT			ZEROMQ		
	100_40	200_20	400_10	100_40	200_20	400_10	100_40	200_20	400_10	100_40	200_20	400_10
latency	295.744	219.556	288.377	5476.131	6174.601	1542.460	579.785	347.323	611.866	42.695	44.926	47.390
latency_stddev	369.336	258.025	386.838	2520.044	2834.921	844.624	727.823	390.823	876.579	2.285	2.994	2.262
jitter	8.559	6.363	11.549	48.493	85.241	41.303	36.334	45.935	15.936	0.720	1.567	2.102
jitter_stddev	17.084	14.377	28.078	98.226	129.875	49.682	41.242	46.387	35.277	2.065	3.272	3.106

**Table 18. Latency, Latency Standard Deviation, Jitter, and Jitter Standard Deviation Normalized Values Under 4 Mbyte/s Cloud Computing Scenario**

Stream size_rate	AMQP			KAFKA			MQTT			ZEROMQ		
	100_40	200_20	400_10	100_40	200_20	400_10	100_40	200_20	400_10	100_40	200_20	400_10
latency	0.048	0.036	0.047	0.887	1.000	0.250	0.094	0.056	0.099	0.007	0.007	0.008
latency_stddev	0.130	0.091	0.136	0.889	1.000	0.298	0.257	0.138	0.309	0.001	0.001	0.001
jitter	0.100	0.075	0.135	0.569	1.000	0.485	0.426	0.539	0.187	0.008	0.018	0.025
jitter_stddev	0.132	0.111	0.216	0.756	1.000	0.383	0.318	0.357	0.272	0.016	0.025	0.024

**Table 19. Latency, Latency Standard Deviation, Jitter, and Jitter Standard Deviation Absolute (ms) Values Under 8 Mbyte/s Cloud Computing Scenario**

	AMQP			KAFKA			MQTT			ZEROMQ		
Stream size_rate	200_40	400_20	800_10	200_40	400_20	800_10	200_40	400_20	800_10	200_40	400_20	800_10
latency	240.584	282.061	248.589	22801.20	27106.191	25015.246	1721.371	1031.907	1422.743	44.251	46.633	50.916
latency_stddev	304.371	362.125	294.301	12189.817	14301.680	13829.161	1954.288	1265.350	1587.541	1.979	1.829	2.282
jitter	7.587	11.021	19.008	69.752	125.556	161.962	36.299	47.291	28.477	1.217	1.347	2.157
jitter_stddev	17.098	28.909	54.632	125.426	165.201	19.896	41.219	49.207	53.472	2.362	2.128	3.118

**Table 20. Latency, Latency Standard Deviation, Jitter, and Jitter Standard Deviation Normalized Values Under 8 Mbyte/s Cloud Computing Scenario**

	AMQP			KAFKA			MQTT			ZEROMQ		
Stream size_rate	200_40	400_20	800_10	200_40	400_20	800_10	200_40	400_20	800_10	200_40	400_20	800_10
latency	0.0089	0.0104	0.0092	0.8412	1.0000	0.9229	0.0635	0.0381	0.0525	0.0016	0.0017	0.0019
latency_stddev	0.0213	0.0253	0.0206	0.8523	1.0000	0.9670	0.1366	0.0885	0.1110	0.0001	0.0001	0.0002
jitter	0.0468	0.0681	0.1174	0.4307	0.7752	1.0000	0.2241	0.2920	0.1758	0.0075	0.0083	0.0133
jitter_stddev	0.1035	0.1750	0.3307	0.7592	1.0000	0.1204	0.2495	0.2979	0.3237	0.0143	0.0129	0.0189

*Telmo Fernández De Barrena Sarasola studied the degree of Industrial Organization Engineering at Tecnun - School of Engineering, University of Navarra (2016-2020). As a final degree project she conducted a study on the existing roles and necessary skills of the different professional profiles in the field of data science. Subsequently, she completed the Dual University Master in Digital Manufacturing (2020-2022) at the IMH (UPV). The final master's project was carried out in collaboration with the Vicomtech research center during a 2-year stay at the center. This project consisted of advanced signal preprocessing and development of data-driven models for wear monitoring of cutting inserts on a lathe. Since September 2022 he has been working as a research assistant at Vicomtech, in the Data Intelligence for Energy and Industrial Processes department, in the areas of predictive maintenance, ML model monitoring, development of architectures for data management and processing, and development of optimization systems. He is currently a PhD student with the University of Deusto.*

*Ander García (male) studied Telecommunication Engineering between 1997 and 2002 at the University of the Basque Country. He obtained his PhD from the same university in 2011. His thesis, which was focused on the application of operations research optimization algorithms, was entitled "Intelligent Personalised Tourist Route Generation". Since 2003 he has worked at Vicomtech as a senior researcher and project manager of several projects. After leading the leading the Intelligent Manufacturing research line, currently he works at the Data Intelligence for Energy and Industrial Processes department, leading the Connectivity and Cloud/Edge research line. He has been an associated professor of Data Base and Software Engineering at the Computer Science Language and Systems department of the University of the Basque Country during 2005, 2006, 2007, 2008, 2009 and 2016. Since 2022 he is an associated professor at the Computer Science Faculty of the University of Deusto.*

*Juan Luis holds a diploma (MEng) of Electric, Electronic and Automation Engineering and a diploma (BEng) of Electronic engineering. He obtained his PhD in 2015 in the area of signal processing and non-destructive testing and was awarded as 'Best PhD Student of the year' by the British institution 'The Welding Institute' in 2014. He has published more than 15 scientific articles in international journals and conferences. He has over 12 years of research experience, in areas such as predictive modelling, signal processing and machine learning. Juan Luis has leaded numerous proposals for national and international projects. From 2018 Juan Luis works as senior researcher in the Data Intelligence for Energy and Industrial processes department, at Vicomtech Research Centre.*