www.ijarp.com ISSN 2456-9992 Page: 01-15

International Journal Advanced Research Publications

Research Article

Volume: 02
Issue: 04

A PRACTICAL MODEL FOR MEASURING LATENCY AND JITTER IN DATA TRANSMISSION USING PYTHON AND UDP SOCKETS

Truong Van Thu

Faculty of National Defense - Security Education, HaNoi University of Science and technology, Viet Nam.

Article Received: 03 March 2026, Article Revised: 21 March 2026, Published on: 11 April 2026

***Corresponding Author: Truong Van Thu**

Faculty of National Defense - Security Education, HaNoi University of Science and technology, Viet Nam.

DOI: https://doi-doi.org/101555/ijarp.2830

ABSTRACT

This paper presents a practical model for measuring and analyzing two critical parameters in network communications, namely latency and jitter, through the use of the Python programming language in conjunction with UDP socket mechanisms. The objective of this study is to develop a simple, low-cost experimental system that can accurately reflect transmission characteristics in a Wi-Fi LAN environment, while also effectively supporting education and research in the field of telecommunications engineering.

The system is designed based on a client–server architecture, in which the sender transmits UTF-8 encoded textual data along with timestamps and SHA-256 hash values to ensure data integrity. The receiver, on the other hand, captures the transmitted packets, verifies their integrity, and computes latency and jitter based on the differences between sending and receiving times. Data transmission is repeated multiple times to collect a sufficiently large dataset for statistical analysis and evaluation of channel stability. Supporting tools such as Wireshark and the matplotlib library are integrated to visualize and validate the measurement results.

Experimental results demonstrate that the proposed system is capable of effectively measuring latency and jitter under typical Wi-Fi network conditions. Furthermore, it reflects the impact of factors such as network load, signal interference, and device configuration on transmission performance. In addition, the integration of data integrity verification enhances the reliability of the system. The proposed model is not only valuable for practical teaching purposes but can also be extended to more advanced research areas such as network optimization, time synchronization, and real-time communication.

The novelty of this study lies in proposing a simple yet effective experimental model capable of reproducing key Quality of Service (QoS) characteristics in UDP/IP networks, making it particularly suitable for educational applications.

KEYWORDS: Latency, Jitter, UDP/IP Transmission, Wi-Fi LAN.

1. INTRODUCTION

In recent years, the rapid development of digital communication systems and computer networks has driven increasing interest in the study of Quality of Service (QoS) metrics, particularly latency and jitter in data transmission processes. These two parameters play a critical role in determining the performance of real-time applications such as industrial control systems, multimedia communications, and Internet of Things (IoT) systems. Modern networking standards, such as IEEE 802.1Q, define traffic classification and prioritization mechanisms to control latency in switched networks [1]. In addition, IEEE 802.1AS provides precise time synchronization among network nodes, forming a foundation for minimizing jitter and ensuring system determinism [2].

Based on these developments, the concept of Time-Sensitive Networking (TSN) has been introduced to meet the requirements of low-latency and highly reliable communications. TSN extends traditional Ethernet by integrating traffic scheduling, time synchronization, and traffic shaping mechanisms, thereby significantly improving the performance of real-time systems [3]. Recent studies have demonstrated that TSN is capable of delivering deterministic communication with low latency and minimal jitter, making it suitable for industrial and automation applications [5]. Furthermore, precise time synchronization techniques in wireless TSN networks have also been proposed to reduce timing discrepancies among devices, thereby enhancing system stability [4].

In parallel with wired networks, wireless technologies such as Wi-Fi and 5G are also being actively investigated to support real-time communications. Enhancements in IEEE 802.11 standards, particularly Wi-Fi 6 and Wi-Fi 7, have focused on reducing latency and improving data transmission efficiency [7]. In addition, controlled channel access mechanisms in Wi-Fi-based TSN networks have significantly improved the ability to guarantee QoS [6]. Experimental studies have shown that the application of TSN over Wi-Fi can enhance the performance of distributed systems, especially in scenarios requiring high reliability [8]. In the domain of mobile communications, 5G technology, with its Ultra-Reliable Low-Latency

Communications (URLLC) feature, enables extremely low latency and high reliability for real-time applications [9].

Despite the advancement of these standards and technologies, the practical deployment and experimental evaluation of latency and jitter in real-world environments remain challenging. In particular, in LAN systems using UDP/IP protocols, latency and jitter are influenced by various factors such as network load, congestion control mechanisms, and transmission medium characteristics. Moreover, the integration of wired and wireless networks introduces additional complexities related to synchronization and QoS assurance [10].

In this context, the development of a practical model for measuring latency and jitter in data transmission is of significant importance for both research and educational purposes. Programming tools such as Python, combined with UDP socket mechanisms, enable rapid implementation of experimental communication models while providing flexible measurement and data analysis capabilities. Through such a model, learners and researchers can directly observe the impact of network conditions on latency and jitter, thereby gaining deeper insights into the underlying phenomena in digital communications.

This paper focuses on the development of an experimental model using Python and UDP sockets to measure and evaluate latency and jitter in data transmission over a LAN environment. The proposed model is designed to be simple and easy to implement, while still capable of capturing the fundamental characteristics of real-world communication systems. The obtained results not only illustrate theoretical QoS concepts but also provide an experimental foundation for research and teaching in computer networking and digital communications.

2. RELATED WORK

The measurement and evaluation of latency and jitter in communication networks have attracted significant attention in both academic research and practical applications. These two metrics are fundamental indicators for assessing Quality of Service (QoS), particularly in systems with real-time requirements. In modern Ethernet networks, the IEEE 802.1Q standard provides traffic classification and prioritization mechanisms to control latency, while IEEE 802.1AS focuses on precise time synchronization among network nodes, thereby reducing jitter and improving system determinism [1], [2].

Building upon these standards, Time-Sensitive Networking (TSN) has been developed as a comprehensive solution to ensure low-latency and highly reliable communications. As described in [3], TSN extends traditional Ethernet by incorporating transmission scheduling,

traffic shaping, and time synchronization mechanisms, enabling deterministic communication. Subsequent studies have demonstrated that TSN can significantly reduce latency and jitter in industrial and automation systems [5]. In particular, precise synchronization methods in wireless TSN have also been proposed to mitigate timing discrepancies among devices, thereby enhancing overall system performance [4].

In addition to wired networks, wireless technologies such as Wi-Fi and 5G are being actively explored to meet real-time communication requirements. Enhancements in the IEEE 802.11 family, particularly toward Wi-Fi 7, focus on reducing latency and improving data transmission efficiency [7]. Moreover, controlled channel access mechanisms in Wi-Fi-based TSN networks enhance QoS guarantees and reduce jitter in wireless environments [6]. Experimental studies indicate that applying TSN over Wi-Fi can significantly improve the performance of distributed systems [8]. Meanwhile, 5G technology, with its Ultra-Reliable Low-Latency Communications (URLLC) capability, introduces new opportunities for applications requiring stringent real-time performance [9]. Furthermore, the integration of wired and wireless TSN networks has been investigated to ensure end-to-end QoS across heterogeneous systems [10].

Regarding measurement approaches, existing studies can generally be classified into two main categories: hardware-based and software-based methods. Hardware-based solutions typically rely on specialized equipment such as protocol analyzers or high-precision clocks to measure latency and jitter with high accuracy. However, these approaches are often costly and difficult to deploy in educational environments. In contrast, software-based methods utilize programming tools and network protocols (e.g., TCP/UDP) to measure packet transmission time, offering greater flexibility and lower cost. Nevertheless, their accuracy may be affected by factors such as operating system overhead, processing delays, and task scheduling mechanisms.

Several experimental measurement models have been proposed, primarily focusing on TSN systems or industrial networks to evaluate performance in real-world scenarios. However, these models are often complex, require specialized hardware, or depend on specific platforms. In addition, many studies emphasize protocol optimization or architectural improvements, while relatively little attention has been given to developing simple and accessible experimental models for educational purposes.

From the above analysis, it can be observed that despite extensive research on latency and jitter measurement and QoS assurance mechanisms, there remains a significant gap in the development of simple, easy-to-implement experimental models suitable for training

environments. In particular, the use of widely accessible tools such as Python combined with UDP sockets for intuitive measurement modeling has not been fully explored. Therefore, this paper aims to propose a practical and highly applicable experimental model that enables learners and researchers to easily access and better understand the characteristics of latency and jitter in communication networks.

3. SYSTEM MODEL AND PROBLEM FORMULATION

3.1. SYSTEM MODEL

The system is designed based on a client–server architecture consisting of two computers connected within the same Local Area Network (LAN), which may operate over either Ethernet or Wi-Fi. One node acts as the server, responsible for listening to incoming packets and sending responses, while the other node functions as the client, which initiates requests and collects measurement data. Communication between the two nodes is implemented using the UDP/IP protocol, prioritizing speed and real-time characteristics, thereby enabling direct observation of transmission channel variations.

During operation, the client periodically transmits packets to the server. Upon receiving a packet, the server immediately sends a response back to the client. The client records both the transmission and reception timestamps, which are then used to compute latency- and jitter-related metrics. This model is well aligned with educational content on UDP/IP communication and Python socket programming, and can be easily deployed in practical laboratory environments.

3.2. PROBLEM FORMULATION

The objective of this study is to measure and evaluate two key parameters in network communications, namely latency and jitter, based on the data collected from packet exchanges between the client and the server.

Latency: In theory, latency is defined as the time required for a packet to travel from the source to the destination (i.e., one-way delay). However, in the proposed model, due to the absence of time synchronization between the two nodes, latency is indirectly measured using the Round Trip Time (RTT), which represents the time interval from when a packet is sent by the client to when the corresponding response is received. Therefore, RTT is used as an approximation of latency in this study.

Jitter: Jitter refers to the variation in latency between consecutive packets and reflects the stability of the data transmission process. In this model, jitter is calculated based on the

differences between successive RTT values, thereby representing the variation in round-trip time rather than one-way delay.

It is important to note that the measured values correspond to RTT rather than true one-way latency. RTT includes both forward and return transmission delays, as well as processing delays at the server. In addition, the measurement results may be influenced by factors such as operating system scheduling mechanisms and processing delays at network nodes. Consequently, the obtained results primarily reflect the variation trends of latency and jitter in the system, rather than the absolute values of one-way delay.

3.3. MATHEMATICAL FORMULATION

Assume that the client sends the i-th packet at time t_i^{send} and receives the corresponding response at time t_i^{recv}. The Round Trip Time is defined as:

$$RTT_i = t_i^{recv} - t_i^{send}$$

The average RTT over NNN packets is computed as:

$$RTT_{avg} = \frac{1}{N}\sum_{i=1}^{N} RTT_i$$

Jitter is defined as the absolute difference between two consecutive RTT values:

$$Jitter_i = | RTT_i - RTT_{i-1} |$$

The average jitter is calculated as:

$$Jitter_{avg} = \frac{1}{N-1}\sum_{i=2}^{N} |RTT_i - RTT_{i-1}|$$

In addition, the standard deviation of RTT is used to evaluate the variability of latency:

$$\sigma = \sqrt{\frac{1}{N}\sum (RTT_i - RTT_{avg})^2}$$

These formulations enable the evaluation of both the overall latency level and the stability of the communication system.

It should be noted that jitter is derived from RTT measurements and therefore reflects the combined effects of forward and return transmission delays, as well as processing delays at the server. Moreover, measurement errors may arise from operating system scheduling mechanisms and the resolution of system timers.

3.4. ASSUMPTIONS

To simplify the model and ensure suitability for practical implementation, the following assumptions are made:

The system operates within a LAN or Wi-Fi environment, where latency is relatively low and stable.

The UDP protocol is selected to eliminate flow control and retransmission mechanisms present in TCP, thereby allowing a more direct observation of transmission delay characteristics. Under typical LAN conditions, the packet loss rate is assumed to be low.

Clock synchronization between the client and server is not required, as RTT measurements rely solely on timestamps recorded at the client side.

The processing delay at the server is assumed to be small and negligible compared to transmission delay. This assumption is reasonable since the server performs only simple response operations without complex data processing. However, in real-world systems, this component may have a non-negligible impact on RTT.

Advanced factors such as severe network congestion or the influence of complex QoS mechanisms are not considered in this model.

4. PROPOSED EXPERIMENTAL MODEL

4.1. System Architecture

The proposed experimental model is based on a simple client–server architecture, implemented using the Python programming language and UDP socket mechanisms. The system consists of two main components:

Server: The server is deployed on a host within the LAN and is responsible for listening for incoming packets from the client via a predefined port. Upon receiving data, the server immediately sends a response back to the client without performing any complex processing, in order to minimize internal processing delay.

Client: The client is implemented on another host in the network. It periodically transmits packets to the server and records both the sending and receiving timestamps. These timestamps are then used to compute latency and jitter metrics.

This architecture aligns well with educational content on socket programming and UDP/IP communication, while maintaining simplicity and ease of deployment in laboratory environments.

4.2. Operation Procedure

The system operates according to the following procedure:

Step 1. Connection Initialization: The client initializes a UDP connection to the server using a predefined IP address and port.

Step 2. Packet Transmission: The client sends a data packet, which may contain a text string or a sequence number.

Step 3. Timestamp Recording (Send): Immediately before transmission, the client records the sending time t_i^{send}

Step 4. Response Reception: The server receives the packet and immediately sends a response back to the client.

Step 5. Timestamp Recording (Receive): Upon receiving the response, the client records the reception time t_i^{recv}.

Step 6. Latency Computation: The client computes the RTT value and stores it for further analysis.

Step 7. Iteration: The above steps are repeated multiple times to collect a sufficiently large dataset for jitter analysis.

4.3. Measurement Algorithm (Pseudo-code)

The measurement algorithm is implemented at the client side as follows:

Input: Server_IP, Port, N (number of packets), Timeout_Value

Output: RTT list, Jitter, Packet Loss Rate

1. Initialize UDP socket (SOCK_DGRAM)

2. Set socket timeout: socket.settimeout(Timeout_Value)

3. Initialize: RTT_list = [], Lost_count = 0

4. For i = 1 to N do:

t_send = current_time()

Send packet_i (including Seq_No, Timestamp, SHA256_Hash)

Try:

Receive response_i from server

t_recv = current_time()

RTT_i = t_recv - t_send

Append RTT_i to RTT_list

Catch Timeout:

Lost_count = Lost_count + 1

Mark packet i as LOST

End For

5. Compute statistics:

RTT_avg = average(RTT_list)

For j = 2 to length(RTT_list) do:

Jitter[j] = abs(RTT_list[j] - RTT_list[j-1])

End For

Jitter_avg = average(Jitter)

Packet_Loss_Rate = (Lost_count / N) × 100%

6. Return RTT_avg, Jitter_avg, Packet_Loss_Rate

The algorithm has low computational complexity, is easy to understand, and is well suited for educational purposes.

4.4. Implementation Tools

The proposed model is implemented using the following tools and libraries:

Programming Language: Python

Networking Library: The socket library is used to establish UDP communication between the client and the server.

Timing Library: The time module, particularly time.perf_counter(), is used to achieve high-resolution timing measurements at millisecond or microsecond levels. However, measurement accuracy may still be affected by operating system scheduling mechanisms.

Additionally, the following libraries may be used:

matplotlib: for visualizing RTT and jitter results

numpy: for statistical data processing

4.5. Data Encapsulation and Formatting

To ensure accurate measurement and data integrity verification in a UDP environment (which lacks strict error control mechanisms compared to TCP), transmitted data is encapsulated using a custom packet structure. Each packet consists of four main components:

Sequence Number: Identifies the i-th packet among the total N packets. This field enables detection of lost or out-of-order packets.

Timestamp: Records the sending time tsend at the application layer immediately before invoking the sendto() function.

Data Payload: The actual message content encoded in UTF-8 format (e.g., “Hello Server”).

SHA-256 Checksum: A hash value computed from the above components to ensure data integrity.

The logical packet structure is defined as:

$$Packet = \{Seq\_No \square Timestamp \square Data \square SHA256\_Hash\}$$

Integrity Verification Procedure (at Receiver):

Step 1: Extract individual components from the received packet.

Step 2: Recompute the SHA-256 hash using {Seq_No, Timestamp, Data}.

Step 3: Compare the computed hash with the received hash value.

If matched: The packet is considered valid, and the timestamp is used for RTT computation.

If not matched: The packet is considered corrupted and is excluded from the statistical dataset.

5. IMPLEMENTATION

5.1. Experimental Environment

The proposed model is implemented on two personal computers running the Windows operating system, connected within the same Local Area Network (LAN). Two connection scenarios are considered:

Wired connection (Ethernet LAN): provides high stability with minimal interference.

Wireless connection (Wi-Fi): reflects common conditions in real-world educational environments.

Both machines are equipped with Python (version 3.x) and execute the corresponding client–server programs. The use of a common and accessible environment ensures that the model can be easily reproduced in laboratory or classroom settings.

5.2. System Configuration

The main system configuration parameters are as follows:

Server IP address: e.g., 192.168.1.10

Communication port: e.g., 5000

Protocol: UDP (SOCK_DGRAM)

Packet size: 64 bytes, 256 bytes, or 1024 bytes

Number of packets (N): ranging from 100 to 1000

These parameters can be flexibly adjusted to investigate the impact of packet size and transmission volume on latency and jitter.

5.3. Experimental Scenarios

To comprehensively evaluate system performance, two primary experimental scenarios are considered:

(a) Scenario 1 – Idle Network:

The two machines only run the measurement program, with no significant additional network traffic. This scenario is used to determine the baseline latency and jitter of the system.

(b) Scenario 2 – Loaded Network:

Network load is introduced through activities such as file transfers, video streaming, or traffic generation tools. This scenario aims to evaluate the impact of network congestion on latency and jitter.

The comparison between these scenarios provides insights into the relationship between network load and communication performance.

5.4. Program Description

The system is implemented in Python and consists of two main components:

(a) Server Program

Initializes a UDP socket and binds it to a specified IP address and port

Listens for incoming packets from the client

Sends an immediate response upon receiving data

Operates continuously until termination

(b) Client Program

Establishes communication with the server

Sequentially transmits packets

Records sending and receiving timestamps using time.perf_counter()

Computes RTT for each packet

Stores the results in a list for further processing.

6. RESULTS AND DISCUSSION

6.1. Experimental Results

Measurement results were collected under two scenarios: an idle network and a loaded network, with 200 packets transmitted in each case. The key statistical metrics of latency (RTT) and jitter are summarized in Table 1.

Table 1. Experimental results of latency and jitter under different network conditions.

Scenario	Avg RTT (ms)	Min RTT (ms)	Max RTT (ms)	Std Dev (ms)	Avg Jitter (ms)	Packet Loss (%)
Idle Network	2.15	1.80	3.10	0.32	0.25	0.00%
Loaded Network	5.72	3.40	12.85	1.85	1.95	2.50%

Packet Loss Analysis: In the idle scenario, the packet loss rate is 0%, indicating an ideal LAN/Wi-Fi environment where router/access point buffers are not saturated. In contrast, under loaded conditions, packet loss increases to 2.50%, which can be attributed to bufferbloat (queue overflow) or channel contention in Wi-Fi networks when multiple data flows coexist.

Relationship between Jitter and Packet Loss: As packet loss increases, jitter also exhibits significant variation (from 0.25 ms to 1.95 ms). This indicates that as the network becomes congested, queueing delays become unstable, leading to both packet loss and degraded performance in real-time applications such as VoIP and video conferencing.

Overall, the results demonstrate a substantial increase in both latency and jitter under network load, which is consistent with real-world communication system behavior.

The standard deviation of RTT in the loaded scenario is significantly higher than in the idle case, indicating greater dispersion of latency values under congestion. This observation is consistent with the increase in jitter and reflects reduced system stability.

6.2. Analysis of Latency Stability

Under idle conditions, RTT values fluctuate within a narrow range (1.80–3.10 ms), with low average jitter (0.25 ms), indicating a stable system with minimal external interference.

In contrast, under loaded conditions, RTT exhibits greater variability (up to 12.85 ms), while jitter increases by nearly eight times. This demonstrates that system stability deteriorates significantly in the presence of congestion or resource contention.

6.3. Impact of Network Conditions

The experimental results indicate that:

Network load is the dominant factor affecting both latency and jitter

In Wi-Fi environments, interference and shared medium access increase delay variability

Under load, packet queueing significantly increases both RTT and jitter

6.4. Comparison with TSN/QoS Theory

According to Time-Sensitive Networking (TSN) theory, mechanisms such as transmission scheduling, precise time synchronization, and traffic prioritization can ensure low latency and minimal jitter.

However, in the proposed experimental model based on conventional UDP/IP:

No packet prioritization mechanism is applied

No precise time synchronization is implemented

No congestion control is enforced

As a result, the measured latency and jitter are inherently non-deterministic, and system performance strongly depends on network conditions. This observation is consistent with TSN-related studies, which highlight that systems without advanced QoS mechanisms cannot guarantee stable latency.

A comparison with standard tools such as ping or iperf is left for future work.

6.5. DISCUSSION

Several important observations can be drawn from the results:

Increasing packet size (e.g., 1024 bytes) leads to a slight increase in average latency due to longer transmission time, especially in wireless environments.

UDP does not guarantee reliability (packet loss may occur) and lacks QoS control mechanisms, resulting in variable latency depending on network conditions.

Under ideal conditions, jitter remains low and acceptable; however, under load, jitter increases significantly, affecting real-time applications.

Practical significance: The model accurately reflects the characteristics of UDP/IP networks and clearly illustrates the differences between conventional networks and QoS/TSN-enabled systems.

Compared to the jitter definition in RFC 3550 (RTP), the jitter used in this study is a simplified form based on consecutive RTT differences, and thus does not fully capture one-way delay variation.

6.6. SUMMARY

The experimental results demonstrate that the proposed model is capable of:

Accurately measuring latency and jitter

Reflecting the impact of network conditions

Effectively supporting teaching and research objectives in networking and digital communications.

7. CONCLUSION

This paper has presented a practical experimental model for measuring latency and jitter in data transmission using Python and the UDP/IP protocol with socket-based communication. The model is implemented using a client–server architecture on two computers within the same LAN/Wi-Fi network, enabling transmission time measurement through Round Trip Time and jitter estimation based on variations between consecutive packets.

Experimental results demonstrate that the proposed model effectively captures the characteristics of real-world communication networks. Under idle conditions, the system achieves low latency and minimal jitter, indicating relatively stable performance. However, under network load, both latency and jitter increase significantly, highlighting the impact of congestion and network conditions on communication performance. These findings are consistent with QoS theory and illustrate the differences between conventional networks and systems designed for real-time applications.

The proposed model offers several key advantages: it is simple, easy to implement, does not require specialized hardware, and incurs low cost, making it well suited for educational environments. Furthermore, it provides an intuitive platform for measuring and analyzing essential network performance metrics.

Therefore, the model is not only valuable for research purposes but also serves as an effective educational tool for courses in computer networks, digital communications, and embedded systems.

Nevertheless, some limitations remain. In particular, one-way delay cannot be directly measured due to the lack of time synchronization between nodes, and the results may be affected by operating system scheduling and experimental conditions.

REFERENCES

1. IEEE. (2018). IEEE Standard for Local and Metropolitan Area Network—Bridges and Bridged Networks. IEEE Std 802.1Q-2018. https://doi.org/10.1109/IEEESTD.2018.8403927
2. IEEE. (2011). IEEE Standard for Local and Metropolitan Area Networks—Timing and Synchronization for Time-Sensitive Applications. IEEE Std 802.1AS-2011. https://doi.org/10.1109/IEEESTD.2011.5741896

3. Lo Bello, L., & Steiner, W. (2019). A perspective on IEEE time-sensitive networking for industrial communication and automation systems. Proceedings of the IEEE, 107(6), 1094–1120. https://doi.org/10.1109/JPROC.2019.2905334
4. Romanov, A., Gringoli, F., & Sikora, A. (2021). A precise synchronization method for future wireless TSN networks. IEEE Transactions on Industrial Informatics, 17(5), 3682–3692. https://doi.org/10.1109/TII.2020.3009995
5. Kang, Y., Lee, S., Gwak, S., Kim, T., & An, D. (2021). Time-sensitive networking technologies for industrial automation in wireless communication systems. Energies, 14(15), 4497. https://doi.org/10.3390/en14154497
6. Avallone, S., Imputato, P., & Magrin, D. (2023). Controlled channel access for IEEE 802.11-based wireless TSN networks. IEEE Internet of Things Magazine, 6(1), 90–95. https://doi.org/10.1109/IOTM.001.2200100
7. Adame, T., Carrascosa-Zamacois, M., & Bellalta, B. (2021). Time-sensitive networking in IEEE 802.11be: On the way to low-latency WiFi 7. Sensors, 21(15), 4954. https://doi.org/10.3390/s21154954
8. Morato, A., Vitturi, S., Tramarin, F., Zunino, C., & Cheminod, M. (2023). Time-sensitive networking to improve the performance of distributed functional safety systems implemented over Wi-Fi. Sensors, 23(17), 7825. https://doi.org/10.3390/s23177825
9. Ji, H., Park, S., Yeo, J., Kim, Y., Lee, J., & Shim, B. (2018). Ultra-reliable and low-latency communications in 5G downlink: Physical layer aspects. IEEE Wireless Communications, 25(3), 124–130. https://doi.org/10.1109/MWC.2018.1700294
10. Seijo, O., Iturbe, X., & Val, I. (2022). Tackling the challenges of the integration of wired and wireless TSN with a technology proof-of-concept. IEEE Transactions on Industrial Informatics, 18(10), 7361–7372. https://doi.org/10.1109/TII.2021.3135550