www.ijarp.com ISSN 2456-9992 Page: 01-11

International Journal Advanced Research Publications

Research Article

Volume: 02
Issue: 04

A LOW-COST EDUCATIONAL MODEL FOR MEASURING LATENCY AND JITTER IN WI-FI NETWORKS USING PYTHON AND UDP SOCKETS

Nguyen Anh Thai

Metrology Department, Faculty of Fundamental Technology, AD-AF Academy of Vietnam.

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

***Corresponding Author: Nguyen Anh Thai**

Metrology Department, Faculty of Fundamental Technology, AD-AF Academy of Vietnam.

DOI: https://doi-doi.org/101555/ijarp.4374

ABSTRACT

This paper proposes a low-cost educational model for measuring and analyzing two key Quality of Service (QoS) parameters—latency and jitter—in Wi-Fi networks using Python and UDP socket communication. The model is implemented through a simple client–server architecture, where the sender transmits UTF-8 encoded data packets embedded with timestamps and SHA-256 hash values to ensure data integrity, while the receiver captures packets, verifies their correctness, and computes latency and jitter based on transmission time differences. To ensure statistical reliability, data exchange is repeated multiple times under various network conditions, and the collected results are analyzed and visualized using tools such as Wireshark and the matplotlib library. Experimental results show that the proposed system effectively reflects real-world Wi-Fi transmission characteristics, including the influence of network load, signal interference, and device configuration on performance. With its simplicity, low implementation cost, and integration of integrity verification, the model serves as a practical platform for teaching and learning in telecommunications and networking courses, while also offering potential for extension to advanced topics such as network optimization, time synchronization, and real-time communication systems.

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

1. INTRODUCTION AND RELATED WORK

In recent years, the rapid evolution of digital communication systems and computer networks has intensified interest in Quality of Service (QoS) metrics, particularly latency and jitter, which are critical to the performance of real-time applications such as industrial automation,

multimedia streaming, and Internet of Things (IoT) systems. Modern networking standards, including IEEE 802.1Q, provide mechanisms for traffic classification and prioritization to control latency in switched networks [1], while IEEE 802.1AS establishes precise time synchronization among network nodes, forming a basis for minimizing jitter and ensuring deterministic communication [2]. Building upon these standards, the concept of Time-Sensitive Networking (TSN) has emerged as a comprehensive solution for low-latency and highly reliable data transmission, integrating traffic scheduling, synchronization, and shaping mechanisms to enhance real-time system performance [3], [5]. At the same time, synchronization techniques in wireless TSN environments have been developed to reduce timing discrepancies and improve system stability [4].

In parallel, wireless technologies such as Wi-Fi and 5G are being actively advanced to support real-time communications. Improvements in IEEE 802.11 standards (e.g., Wi-Fi 6 and Wi-Fi 7) focus on reducing latency and increasing transmission efficiency [7], while controlled channel access mechanisms in Wi-Fi-based TSN networks further strengthen QoS guarantees [6]. Experimental studies have demonstrated the feasibility of applying TSN over Wi-Fi to enhance reliability in distributed systems [8], and 5G networks, particularly through Ultra-Reliable Low-Latency Communications, offer extremely low latency and high reliability for real-time applications [9]. Furthermore, the integration of wired and wireless TSN networks has been investigated to ensure end-to-end QoS across heterogeneous systems [10].

Despite these technological advances, practical measurement and evaluation of latency and jitter in real-world environments remain challenging, especially in UDP/IP-based LAN systems where performance is influenced by factors such as network load, congestion, and transmission medium characteristics, as well as the complexity introduced by integrating wired and wireless networks [10]. In this context, the development of a low-cost and practical experimental model is essential for both educational and research purposes. Leveraging Python programming and UDP socket communication enables rapid implementation of flexible measurement systems, allowing users to directly observe and analyze the impact of network conditions on QoS parameters.

From a research perspective, existing approaches for measuring latency and jitter can generally be classified into hardware-based and software-based methods. Hardware-based solutions rely on specialized equipment, such as protocol analyzers or high-precision clocks, to achieve high measurement accuracy; however, they are often costly and less suitable for

educational deployment. In contrast, software-based methods utilize programming tools and standard protocols (e.g., TCP/UDP) to estimate packet transmission time, offering greater flexibility and lower cost, although their accuracy may be affected by operating system overhead, processing delays, and scheduling mechanisms.

Several experimental models have been proposed to evaluate latency and jitter, mainly focusing on TSN systems or industrial network environments. While these models provide valuable insights, they are often complex, hardware-dependent, or platform-specific, which limits their applicability in teaching contexts. Moreover, many existing studies emphasize protocol optimization or architectural improvements, whereas relatively limited attention has been paid to developing simple and accessible experimental models tailored for educational purposes.

Based on the above analysis, it can be observed that although significant progress has been made in QoS assurance mechanisms and performance evaluation, there remains a gap in the availability of low-cost, easy-to-implement experimental platforms. In particular, the use of widely accessible tools such as Python combined with UDP sockets for intuitive measurement modeling has not been fully exploited. Therefore, this paper proposes a simple yet effective experimental model that enables learners and researchers to easily measure and analyze latency and jitter in Wi-Fi LAN environments, thereby providing both a practical teaching platform and a foundation for further research in computer networking and digital communications.

2. SYSTEM MODEL AND PROBLEM FORMULATION

2.1. 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.

2.2. 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.

2.3. 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.

2.4. 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.

3. PROPOSED EXPERIMENTAL MODEL

3.1. System Architecture and Operation Procedure

The proposed experimental model is based on a simple client–server architecture implemented in Python using UDP socket communication. The system consists of two main components: the server, which listens for incoming packets on a predefined port and immediately responds to minimize processing delay, and the client, which periodically transmits packets and records both sending and receiving timestamps for performance measurement. This lightweight architecture ensures ease of deployment and aligns well with educational objectives in socket programming and UDP/IP communication.

The system operates through a structured sequence of steps. First, the client initializes a UDP connection to the server using a predefined IP address and port. It then transmits packets containing basic data such as sequence numbers or text messages. Immediately before sending, the client records the transmission timestamp, and upon receiving the server's response, it records the reception timestamp. Based on these values, the Round Trip Time (RTT) is computed. This process is repeated multiple times to collect sufficient data for jitter analysis. The iterative nature of the procedure allows for statistical evaluation of latency variation under different network conditions, ensuring both reliability and reproducibility of results.

3.2. Measurement Algorithm and Implementation Details

The measurement process is implemented at the client side using a simple yet effective algorithm. The client initializes a UDP socket, sets a timeout value, and iteratively sends packets to the server while measuring RTT for each successful transmission. Lost packets are identified through timeout events, enabling calculation of packet loss rate. Jitter is computed as the absolute difference between consecutive RTT values, and statistical metrics such as average RTT, average jitter, and packet loss rate are derived from the collected data.

The algorithm is summarized 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
3. Initialize RTT_list = [], Lost_count = 0
4. For i = 1 to N:

Record t_send

Send packet_i (Seq_No, Timestamp, SHA256_Hash)

If response received:

Record t_recv

Compute RTT_i = t_recv - t_send

Append RTT_i

Else:

Increment Lost_count

5. Compute RTT_avg, Jitter_avg, Packet_Loss_Rate
6. Return results

The implementation utilizes Python's socket library for UDP communication and the time module (specifically time.perf_counter()) for high-resolution timing. Additional libraries such as matplotlib and numpy can be used for visualization and statistical analysis.

To ensure data integrity in the connectionless UDP environment, each transmitted packet follows a structured format including a sequence number, timestamp, data payload (UTF-8 encoded), and a SHA-256 checksum. Upon reception, the checksum is recomputed and compared with the received value. Only valid packets are used for RTT and jitter calculations, while corrupted packets are discarded. This mechanism enhances the reliability of the experimental results while maintaining the simplicity and low computational cost required for educational applications.

4. IMPLEMENTATION

4.1. Experimental Setup and Configuration

The proposed model is implemented on two personal computers running the Windows operating system and connected within the same Local Area Network (LAN), supporting both wired (Ethernet) and wireless (Wi-Fi) scenarios. The wired connection provides a stable environment with minimal interference, while the Wi-Fi setup reflects typical real-world educational conditions. Both machines are equipped with Python (version 3.x) and run the corresponding client–server programs, ensuring that the system remains low-cost, accessible, and easy to reproduce in laboratory or classroom settings.

Key system parameters include the server IP address (e.g., 192.168.1.10), communication port (e.g., 5000), and the use of the UDP protocol (SOCK_DGRAM). Packet sizes are configurable (e.g., 64, 256, and 1024 bytes), and the number of transmitted packets typically

ranges from 100 to 1000. These parameters can be flexibly adjusted to analyze the impact of transmission characteristics on latency and jitter. To evaluate performance under different conditions, two experimental scenarios are considered: an idle network, where only the measurement program is active to establish baseline performance, and a loaded network, where additional traffic (e.g., file transfers or video streaming) is introduced to simulate congestion and assess its impact on QoS metrics.

4.2. System Operation and Program Design

The system is developed in Python using a client–server architecture based on UDP socket communication. The server program initializes a UDP socket, binds it to a predefined IP address and port, and continuously listens for incoming packets from the client. Upon receiving a packet, the server immediately sends a response, enabling round-trip time (RTT) measurement.

The client program is responsible for initiating communication, sequentially transmitting packets to the server, and recording precise sending and receiving timestamps using high-resolution timing functions such as time.perf_counter(). Based on these timestamps, the client computes RTT for each packet and stores the results for further analysis of latency and jitter. This design ensures a simple yet effective mechanism for capturing transmission performance while maintaining flexibility for experimentation and extension.

5. RESULTS AND DISCUSSION

5.1. Experimental Results

Measurement results were collected under two scenarios—idle network and loaded network—with 200 transmitted packets in each case. The statistical results 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%

The results clearly show that network conditions have a significant impact on transmission performance. Under idle conditions, latency remains low and stable, with RTT fluctuating

within a narrow range and minimal jitter, indicating a stable communication environment. In contrast, under loaded conditions, both latency and jitter increase substantially, with RTT reaching up to 12.85 ms and jitter rising nearly eightfold. The higher standard deviation of RTT in this scenario further confirms increased variability and reduced system stability under congestion.

Packet loss analysis also reveals notable differences: while no packet loss is observed in the idle scenario, a loss rate of 2.50% occurs under load, likely due to buffer overflow or channel contention in Wi-Fi environments. Moreover, the increase in packet loss is accompanied by a sharp rise in jitter, reflecting unstable queueing delays and degraded performance, particularly for real-time applications such as VoIP and video conferencing.

These findings confirm that network load is a dominant factor affecting both latency and jitter. In wireless environments, interference and shared medium access further amplify delay variability, while packet queueing under congestion significantly contributes to increased RTT and jitter.

5.2. Discussion and Theoretical Implications

The experimental results are consistent with QoS and Time-Sensitive Networking (TSN) theory, which emphasizes the importance of mechanisms such as traffic prioritization, scheduling, and precise time synchronization to ensure low latency and minimal jitter. In contrast, the proposed UDP/IP-based model does not implement packet prioritization, synchronization, or congestion control mechanisms, resulting in inherently non-deterministic performance that strongly depends on network conditions. This highlights the fundamental differences between conventional networks and TSN-enabled systems.

Several additional observations can be drawn. Increasing packet size (e.g., 1024 bytes) leads to a slight rise in latency due to longer transmission time, particularly in wireless environments. The use of UDP, while simple and efficient, does not guarantee reliability and lacks QoS control, which contributes to variability in latency and potential packet loss. While jitter remains low under ideal conditions, it increases significantly under load, directly affecting the quality of real-time services.

From a methodological perspective, the jitter calculation in this study is based on differences between consecutive RTT measurements, which is simpler than the definition in RFC 3550 (RTP) and does not fully capture one-way delay variation. Nevertheless, the model

effectively reflects key characteristics of UDP/IP networks and provides clear insights into performance degradation under congestion.

Overall, the proposed model demonstrates its capability to accurately measure latency and jitter, reflect the impact of network conditions, and serve as a practical tool for both teaching and research in computer networking and digital communications.

6. CONCLUSION

This paper presented a low-cost experimental model for measuring latency and jitter in Wi-Fi/LAN environments using Python and UDP socket communication within a client–server architecture. The model enables RTT-based latency measurement and jitter estimation from variations between consecutive packets. Experimental results show that the system effectively reflects real network behavior, with low latency and stable jitter under idle conditions, and noticeable performance degradation under network load, consistent with QoS theory.

The proposed approach is simple, easy to deploy, and does not require specialized hardware, making it well suited for educational use while still providing meaningful insights for research. However, limitations remain, including the inability to directly measure one-way delay without time synchronization and potential inaccuracies due to 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