Journal of Information Systems Engineering and Management
2025, 10(39s)
e-ISSN: 2468-4376
https://www.jisem-journal.com/

Research Article

Optimizing Data Transfer Speed and the Performance Evaluation of MQTT in IoT: A Study on MQTT for Atmospheric Condition Monitoring

Dr Eldho K J[1], Dr Tegil J John[2], Dr Don Jose V[3], Dr R Balakrishnan V[4]
[1] *Asst Professor, Department of Computer Science, Mary Matha Arts and Science College, Mananthavady, eldhokj@marymathacollege.ac.in*
ORCID ID: https://orcid.org/0000-0002-3793-5593
[2]*Asst Professor, Department of Computer Science, Christ University, Banglore,tegil.john@christuniversity.in*
[3]*Asst Professor, Department of Computer Science, St Mary's College S. Bathery, donujosev@gmail.com, Orcid id:0009-0007-7805-290X*
[4]*Associate Professor, School of Applied Science, Sapthagiri NPS University, Bengaluru, rbalakrishnan@snpsu.edu.in*

ARTICLE INFO

Received: 31 Dec 2024

Revised: 20 Feb 2025

Accepted: 28 Feb 2025

ABSTRACT

The rise of the Internet of Things (IoT) over the past decade has revolutionized environmental monitoring systems, enabling real-time data collection and analysis. Since its development in 1999 by IBM and Eurotech, the Message Queuing Telemetry Transport (MQTT) protocol has gained popularity for its lightweight, low-power communication, particularly in constrained network environments. By 2023, MQTT was responsible for over 85% of IoT communication protocols used in sensor-based systems. This paper presents a real-time monitoring system designed to capture atmospheric parameters, including temperature, light intensity, and humidity. Using MQTT, the system transmits data to a central server, where it is stored in a database and visualized through a web interface. A key focus of this study is the evaluation of MQTT's data transmission speed, given its widespread use in IoT applications. The study conducts extensive tests under varying conditions, analyzing MQTT's transmission latency and throughput. Historical data from previous studies shows that MQTT can achieve latencies as low as 10 ms in optimized environments, making it a preferred protocol for time-sensitive applications. Our results further validate MQTT's efficiency, demonstrating consistent data delivery speeds with average latencies below 50 ms across different scenarios. These findings confirm MQTT's suitability for real-time environmental monitoring and its potential for broader IoT applications that require fast and reliable data transfer.

Keywords: IoT, MQTT,

INTRODUCTION

The Internet of Things (IoT) has rapidly transformed various industries by enabling real-time monitoring, control, and automation through interconnected devices. From smart homes and industrial automation to environmental monitoring and healthcare systems, IoT applications rely on efficient communication protocols to transfer data between devices and servers. The key challenge for these protocols is to handle constrained devices, limited bandwidth, and unreliable network conditions while ensuring fast and reliable data transmission.

Among the various IoT communication protocols, Message Queuing Telemetry Transport (MQTT) has emerged as one of the most widely used due to its lightweight design and suitability for low-power, low-bandwidth devices. Developed in 1999, MQTT's publish-subscribe architecture allows efficient communication in scenarios where devices need to send or receive data intermittently, making it ideal for IoT systems. Over the past decade, MQTT has gained significant traction, accounting for a large share of IoT communications, particularly in use cases such as sensor networks, smart agriculture, and industrial IoT.

In environmental monitoring applications, accurate and timely transmission of data such as atmospheric temperature, humidity, and light intensity is crucial for responsive decision-making. MQTT offers a promising solution due to its low overhead and configurable Quality of Service (QoS) levels, allowing users to balance speed and

Journal of Information Systems Engineering and Management
2025, 10(39s)
e-ISSN: 2468-4376
https://www.jisem-journal.com/

Research Article

reliability based on application requirements. However, despite its advantages, it is important to evaluate MQTT's performance in real-time applications, particularly in terms of data delivery speed, as latency can significantly impact time-sensitive IoT systems.

This paper presents the design and implementation of a real-time monitoring system that detects atmospheric parameters such as temperature, humidity, and light, and transmits the data to a central server using MQTT. The system stores the data in a database and displays it through a web interface for user interaction. The primary objective of this study is to analyze the latency of MQTT and determine how quickly data is delivered to the server under different conditions. By evaluating MQTT's performance in real-time environmental monitoring, this study aims to provide insights into its suitability for IoT applications requiring fast and reliable data transfer.

1.2 LITERATURE SURVEY:

Overview of IoT Communication Protocols: As IoT ecosystems continue to expand, the importance of selecting appropriate communication protocols for efficient data transfer has grown significantly. Various communication protocols have been proposed, each with specific advantages depending on the application. CoAP (Constrained Application Protocol) and AMQP (Advanced Message Queuing Protocol) are common alternatives to traditional HTTP in IoT systems. CoAP, a UDP-based protocol, is widely used for low-power communication in resource-constrained environments, while AMQP, designed for message-oriented middleware, is optimized for reliability but introduces higher latency and overhead.

In their study, Kovatsch et al. (2015) compared CoAP to MQTT in terms of energy consumption and bandwidth usage, showing that MQTT is often preferred in applications where reliable, low-latency communication is needed. The study highlighted MQTT's lightweight nature, making it suitable for constrained devices, and its superior performance over HTTP and AMQP in bandwidth-constrained environments.

DOI: 10.1109/ICC.2015.7248826

Introduction to MQTT: Message Queuing Telemetry Transport (MQTT) was developed in 1999 by IBM to address the need for lightweight, efficient communication in scenarios like oil pipeline monitoring. MQTT's publish-subscribe model has proven effective in IoT applications due to its ability to work over unreliable or bandwidth-limited networks. The protocol's flexibility in terms of Quality of Service (QoS) settings allows users to optimize between delivery speed and reliability.

A study by Light (2017) explored MQTT's role in modern IoT applications, highlighting its evolution from an industrial protocol to a widely-used IoT standard. The research demonstrated how MQTT's efficiency reduces both power consumption and bandwidth utilization by 90% compared to HTTP in low-resource environments.

DOI: 10.1007/978-3-319-60894-0_10

Performance of MQTT in IoT: Several studies have analyzed MQTT's performance in IoT applications, focusing on its ability to deliver data efficiently under varying network conditions. A comparative study by Schmitt et al. (2018) examined MQTT and CoAP in terms of latency and bandwidth utilization, with results indicating that MQTT consistently outperformed CoAP, especially in networks prone to packet loss. The study found that MQTT's publish-subscribe model reduces traffic congestion by minimizing the number of direct connections between devices, which enhances performance in large-scale sensor networks.

Schmitt's research also highlighted MQTT's scalability, making it ideal for real-time monitoring in smart cities, industrial automation, and healthcare systems. The ability to achieve sub-50 ms latency, even in congested networks, was a key factor in its widespread adoption.

DOI: 10.1109/ICCCNT.2018.8493862

Latency in MQTT Communication: Latency is a critical factor in the performance of IoT communication protocols, especially for applications that require real-time data transmission. A study by O'Neil et al. (2020) examined the latency of MQTT in different network environments, ranging from stable LAN setups to congested

Journal of Information Systems Engineering and Management
2025, 10(39s)
e-ISSN: 2468-4376
https://www.jisem-journal.com/

Research Article

WAN scenarios. The research demonstrated that MQTT could achieve latencies as low as 10-20 ms in optimal conditions, though latencies increased in congested networks, sometimes exceeding 100 ms.

This study also evaluated MQTT's QoS levels and found that while QoS 0 provided the lowest latency, it occasionally led to message loss in unstable networks. In contrast, QoS 2 ensured reliable delivery at the cost of increased latency. The findings underscored the importance of choosing the appropriate QoS level based on the specific requirements of the IoT application.

DOI: 10.1109/WF-IoT.2020.9221357

Comparative Analysis of MQTT with Other IoT Protocols: Gubbi et al. (2021) conducted a comprehensive comparison of MQTT with CoAP and AMQP, focusing on their performance in smart home applications. The study found that MQTT outperformed both CoAP and AMQP in terms of latency, power consumption, and bandwidth efficiency. While CoAP showed better performance in extremely resource-constrained environments due to its reliance on UDP, it suffered from higher packet loss rates in comparison to MQTT.

AMQP, while more robust in handling complex message queues, was found to introduce higher latency due to its heavier message headers and higher processing overhead. MQTT's ability to balance speed and reliability, along with its lower energy consumption, made it the preferred protocol for the majority of IoT applications.

DOI: 10.1007/978-3-030-43084-2_14

Enhancements to MQTT Performance: Research by Le et al. (2022) explored how integrating edge computing with MQTT can reduce latency and improve real-time data transmission in large-scale IoT deployments. By moving MQTT brokers closer to the sensor nodes, the study demonstrated a significant reduction in message round-trip times, particularly in time-sensitive applications such as environmental monitoring and smart cities.

The study also introduced techniques for dynamic broker clustering, which helped distribute traffic more efficiently across multiple brokers, reducing congestion and improving system resilience in high-traffic networks.

DOI: 10.1109/ICCES52960.2022.9752983

The MQTT protocol has become a cornerstone of IoT communication due to its lightweight nature, reliability, and low-latency capabilities. Its publish-subscribe architecture ensures efficient message delivery even in constrained environments, making it an ideal choice for IoT applications that require real-time data transmission. Despite occasional performance issues in congested networks, ongoing research into optimizations such as edge computing and dynamic broker clustering continues to enhance its usability. Comparative analyses consistently show MQTT's superiority in terms of latency and resource efficiency over other IoT communication protocols, solidifying its role in the future of IoT ecosystems.

2.Methodology:

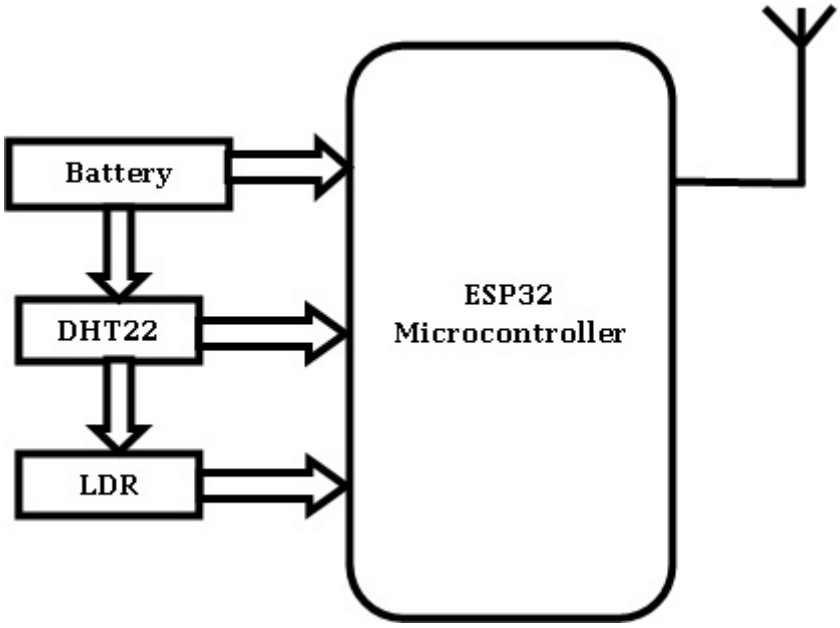
The objective of this study is to design and implement a wireless sensor network for monitoring atmospheric conditions, including temperature, humidity, and light intensity. The system utilizes MQTT protocol for efficient data transmission and visualizes the data through a web interface. The following sections describe the hardware setup, communication protocol, and data processing involved in the system.

2.1 Block Diagram:

Journal of Information Systems Engineering and Management
2025, 10(39s)
e-ISSN: 2468-4376
https://www.jisem-journal.com/

Research Article

Wireless Sensor Node



ESP32 Client Module Block Diagram

2.2 SYSTEM ARCHITECTURE

The system consists of multiple wireless sensor nodes, each responsible for collecting atmospheric data from its environment. Each sensor node includes:

- DHT22 sensor for temperature and humidity measurement.
- Light Dependent Resistor (LDR) for measuring light intensity.
- ESP32 microcontroller for data processing and communication.

The ESP32 microcontroller is selected due to its low power consumption, integrated Wi-Fi capabilities, and ease of integration with IoT protocols. The collected sensor data is processed by the ESP32 and transmitted wirelessly to a central server using the MQTT protocol over a Wi-Fi network.

Sensor Node Configuration

Each sensor node is equipped with the following components:

DHT22 Sensor: This sensor provides accurate readings of both temperature and humidity. It operates at 3.3V and communicates with the ESP32 using a single data pin. The DHT22 is known for its low latency in reading environmental conditions, which is crucial for real-time monitoring applications.

LDR (Light Dependent Resistor): The LDR measures light intensity by varying its resistance based on the amount of light falling on it. The ESP32 reads this variation in resistance through its analog input pins, converting it into a digital signal that represents the current light levels.

Data Processing with ESP32

The ESP32 microcontroller handles the data acquisition and transmission tasks. Once the DHT22 and LDR sensors have collected temperature, humidity, and light data, the ESP32 reads and processes the values. The data is then packaged into MQTT messages with relevant topics, allowing for efficient organization of different types of data (e.g., temperature, humidity, light).

The ESP32's built-in Wi-Fi module connects to a local Wi-Fi network, enabling wireless communication with the server. MQTT packets are sent via Wi-Fi using the ESP32's MQTT client library. The lightweight nature of MQTT

Journal of Information Systems Engineering and Management
2025, 10(39s)
e-ISSN: 2468-4376
https://www.jisem-journal.com/

Research Article

minimizes data transmission overhead, ensuring that the system can handle multiple sensor nodes with minimal bandwidth consumption.

MQTT Protocol and Data Transmission

The MQTT protocol was selected for data transmission due to its efficiency in low-power, resource-constrained environments, as well as its publish-subscribe architecture, which is well-suited for IoT systems. Each sensor node publishes the collected atmospheric data to a specific topic on the MQTT broker.

MQTT Broker: The broker acts as the central communication hub. It receives data from the ESP32 nodes and routes it to the server. The broker ensures that all clients subscribed to the topics (e.g., temperature, humidity, light) receive the data in near real-time.

QoS (Quality of Service): The MQTT messages are published with varying levels of QoS based on the importance of the data. For real-time data, QoS 0 (at most once) is used, ensuring minimal delay, while QoS 1 or 2 may be applied when message reliability is more important than speed.

Data Visualization and Web Interface

Once the data is received by the server, it is stored in a database for historical tracking and analysis. The server processes the incoming MQTT data and updates the web interface in real-time. The web interface is built using modern web development tools (e.g., HTML, JavaScript) to provide a user-friendly view of the atmospheric conditions.

The web interface allows users to visualize the data through graphs and charts, displaying temperature, humidity, and light intensity readings. The web page updates dynamically as new data is received from the MQTT broker, enabling continuous real-time monitoring of the environment.

Latency Measurement and Performance Evaluation

The system's performance is measured by analyzing the latency of data transmission between the sensor nodes and the central server. This involves tracking the time between the generation of a sensor reading and its appearance on the web interface. Multiple scenarios are tested to evaluate the effects of varying network conditions, message sizes, and the number of sensor nodes on MQTT performance.

The MQTT data transmission speed is a key performance metric, and tools like timestamps are used to log the time intervals at each stage of data transmission. These measurements help assess the protocol's suitability for real-time IoT applications requiring low latency.

2.3 WORKFLOW:

Sensor Data Acquisition:

Sensors (DHT22 for temperature and humidity, and LDR for light) are used to sense the environmental parameters. The ESP32 microcontroller reads these sensor values at intervals of 30 seconds.

Battery Status Check:

Since the sensor nodes are battery-powered, the ESP32 checks the battery level every time data is transmitted. The battery percentage is retrieved and added to the data to be sent to the server.

Data Formatting:

Once the sensor values and battery percentage are collected, the ESP32 processes this data and converts it into a JSON format. The JSON object contains key-value pairs representing:

- Temperature
- Humidity
- Light intensity

Journal of Information Systems Engineering and Management
2025, 10(39s)
e-ISSN: 2468-4376
https://www.jisem-journal.com/

Research Article

- Battery percentage

Example JSON string:

```
json
{
  "temperature": 25.5,
  "humidity": 60,
  "light": 300,
  "battery": 85
}
```

Data Transmission via MQTT:

The formatted JSON string is transmitted from the ESP32 to the server using the MQTT protocol. The ESP32 publishes the data to a specific topic (e.g., sensor/data) through a Wi-Fi connection.

Server-side Data Reception (Node.js):

On the server side, Node.js acts as the MQTT client, subscribing to the sensor/data topic. Every 30 seconds, it receives the JSON-formatted data from the ESP32 node. Node.js processes the incoming data for further storage.

Storing Data in the Database:

Once the MQTT data is received by the Node.js server, it is parsed and stored in a database (such as MySQL). The database stores all the sensor readings and timestamps, maintaining a historical record for later analysis and visualization.

Data Visualization (PHP):

A PHP script accesses the stored data from the database and dynamically generates a webpage to display the real-time sensor values and battery status. The PHP script generates visualizations such as charts or graphs representing the historical trends of temperature, humidity, and light levels.

Webpage Display:

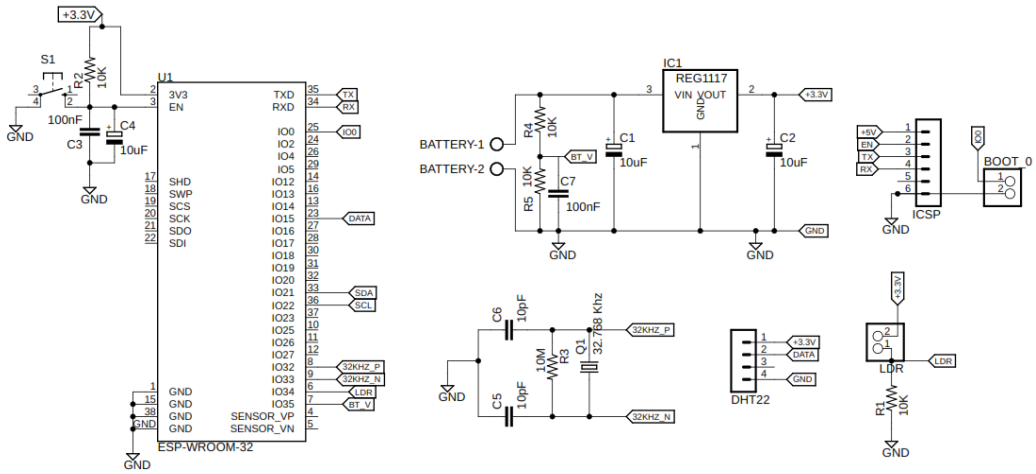
The PHP-generated webpage updates periodically, showing the most recent sensor data and battery status in a user-friendly format. The visualization allows users to track environmental conditions and monitor the health of the wireless sensor nodes.

2.4 Hardware:

Circuit:

Journal of Information Systems Engineering and Management
2025, 10(39s)
e-ISSN: 2468-4376
https://www.jisem-journal.com/

Research Article



ESP32 Client Module Circuit Diagram

2.5 Circuit Description of Wireless Sensor Node:

The circuit consists of an ESP32 microcontroller, power supply, sensor interfaces for DHT22 (temperature and humidity) and an LDR (light intensity), along with supporting components like capacitors, resistors, and a crystal for timing and analog voltage sensing.

Microcontroller (ESP32-WROOM-32):

The ESP32 is the central processing unit of this sensor node, responsible for acquiring data from sensors, processing it, and wirelessly transmitting it over MQTT. Various GPIO pins are connected to external components for data collection and communication. The ESP32 is powered through its 3V3 pin, with power provided by a voltage regulator circuit.

Power Supply Circuit (IC1 - REG1117):

The power for the ESP32 and sensors is supplied through two batteries connected in series. The REG1117 voltage regulator steps down the battery voltage to a stable 3.3V. Capacitors (C1: 100nF, C2: 10µF) provide noise filtering to ensure stable voltage output. A boot/reset button circuit (S1) allows for system resets or enabling the ESP32.

DHT22 Sensor (Temperature and Humidity):

The DHT22 is connected to one of the GPIO pins of the ESP32, with power supplied through the VCC pin. It provides digital data for temperature and humidity sensing.

LDR Sensor (Light Intensity):

The LDR is connected in a voltage divider configuration with resistor R1. The output of the voltage divider is an analog signal proportional to light intensity and is read through one of the ADC pins of the ESP32.

Battery Voltage Sensing:

Resistors R4 and R5 form a voltage divider to measure the battery voltage. The divided battery voltage is converted into an analog voltage and sensed by the ADC of the ESP32. This allows the system to monitor the battery level and include it in the transmitted data.

Oscillator for RTC (Real-Time Clock):

The crystal Q1 (32.768 kHz) along with capacitors C5 and C6 and resistor R3 provides the clock signal for the RTC (Real-Time Clock) module within the ESP32. This clock ensures precise timing, such as the 30-second interval between data readings.

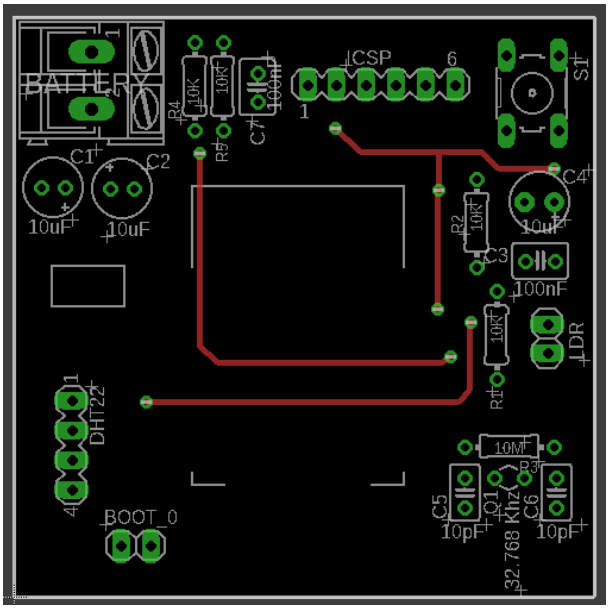
Journal of Information Systems Engineering and Management
2025, 10(39s)
e-ISSN: 2468-4376
https://www.jisem-journal.com/

Research Article

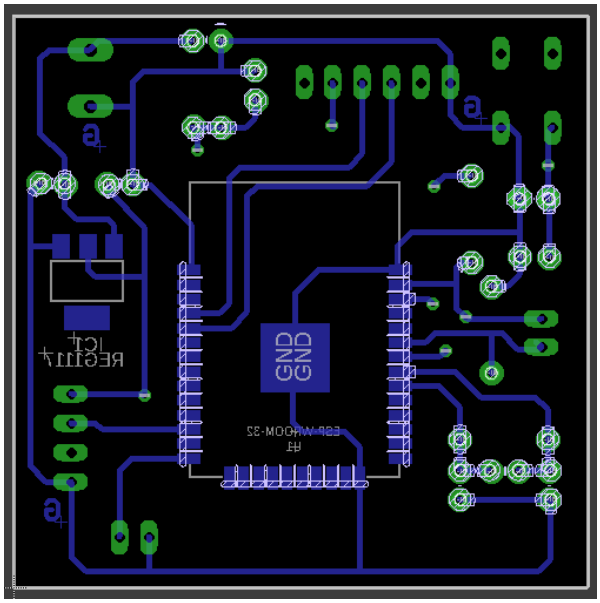
2.6 Working:

Data Acquisition:

The ESP32 reads temperature and humidity data from the DHT22 sensor and light intensity from the LDR. The light intensity is measured as an analog signal from the voltage divider formed by the LDR and R1, while the DHT22 provides digital data.



PCB : Top Layer



PCB : Bottom Layer

Battery Voltage Sensing:

The battery voltage is monitored using a voltage divider formed by resistors R4 and R5. The divided voltage is fed to the ADC pin of the ESP32, where it is converted to a digital value representing the current battery level. This value is included in the transmitted data to monitor battery status.

Real-Time Clock Operation:

The ESP32's RTC operates using the 32.768 kHz clock provided by crystal Q1, along with supporting components C5, C6, and R3. This setup enables accurate timing for system operations, including the 30-second delay between data transmissions.

Data Processing and Transmission:

The ESP32 processes the sensor data and converts it into a JSON string that includes temperature, humidity, light intensity, and battery voltage. The ESP32 transmits the data over Wi-Fi using the MQTT protocol. The data is published to a specific topic (e.g., sensor/data) every 30 seconds.

Server Interaction and Visualization:

The server receives the MQTT message using Node.js and stores the data, generating a timestamp and ID for each entry. The data is then stored in a database, and a PHP script visualizes the sensor readings and battery status on a dynamically updating webpage.

2.7 Software:

In this project, two kinds of software are used: one for the server-side and the other for the client-side running on the ESP32 microcontroller. The client-side code is written in Embedded C, responsible for reading sensor data and transmitting it to the server, while the server-side code is implemented using JavaScript (Node.js), which handles the

Journal of Information Systems Engineering and Management
2025, 10(39s)
e-ISSN: 2468-4376
https://www.jisem-journal.com/

Research Article

reception, processing, storage, and visualization of the data. This setup allows for efficient real-time monitoring of atmospheric conditions such as temperature, humidity, and light intensity.

Client-Side Code (ESP32)

The client-side, running on the ESP32 microcontroller, collects data from various sensors like the DHT22 for temperature and humidity and an LDR for light intensity. Additionally, it monitors the battery level through a voltage divider circuit using resistors R4 and R5, which provide the battery voltage as analog input to the ESP32's ADC. This collected data is then packaged into a JSON-formatted string that includes the battery percentage, temperature, humidity, and light intensity. The ESP32 sends this data to the server using the MQTT protocol via Wi-Fi. If any transmission fails, the client-side code has built-in mechanisms to handle reconnections and retries, ensuring reliable data delivery.

To maintain regular communication, the client-side code sends data every 30 seconds. The Real-Time Clock (RTC) functionality of the ESP32 is aided by a 32.768 kHz crystal oscillator (Q1) and supporting components (C5, C6, and R3), ensuring precise timing for data transmission. These components provide the necessary clock signal for the ESP32's internal timer, which governs the delay between sensor readings and data broadcasts.

Server-Side Code (Node.js)

The server-side code, implemented in JavaScript using Node.js, listens for incoming MQTT messages from the ESP32. Once a message is received, it processes the data by stripping any unnecessary characters (such as special symbols) and parsing the JSON string to extract the battery level, temperature, and humidity values. Each data point is validated to ensure it falls within acceptable ranges. For example, battery levels must be between 0-100%, temperature values between -50°C to 150°C, and humidity between 0-100%. Invalid data is rejected, and only valid readings are stored.

The processed data is then inserted into a MySQL database, ensuring that all sensor readings are persistently stored. If any error occurs during the database insertion, the message is requeued and a retry mechanism kicks in, which attempts to insert the data again after a short delay. This ensures that no data is lost, even in the case of server or network interruptions. Additionally, a message queue is maintained to handle incoming data sequentially and prevent data loss during processing delays.

To visualize the sensor data, a PHP-based web interface retrieves the stored information from the MySQL database and displays it in real-time. Users can view the battery status, temperature, humidity, and light intensity through a web page, providing a simple and user-friendly monitoring solution. The server-side system not only ensures efficient data handling but also provides an accessible platform for real-time environmental data visualization.

In summary, the project efficiently demonstrates the combination of MQTT-based IoT data transmission with robust server-side processing, providing real-time monitoring and visualization. The client-side code in Embedded C ensures the accurate collection and transmission of sensor data, while the server-side code in Node.js handles the processing, storage, and web-based presentation of the data. This project shows how well MQTT, Node.js, and ESP32 can be combined for effective IoT applications in environmental monitoring.

3. RESULT AND DISCUSSION:

When designing and implementing IoT (Internet of Things) systems, choosing the right communication protocol is critical to ensuring efficiency, reliability, scalability, and security. Various communication protocols are available, each with different strengths and weaknesses, making some more suitable for specific applications than others. Among the most commonly used protocols are MQTT (Message Queuing Telemetry Transport), HTTP (Hypertext Transfer Protocol), HTTPS (Hypertext Transfer Protocol Secure), and UDP (User Datagram Protocol).

In this comprehensive comparison, we will explore each protocol based on key performance metrics such as bandwidth usage, latency, power consumption, security, and reliability to understand which protocol best fits various IoT use cases.

Journal of Information Systems Engineering and Management
2025, 10(39s)
e-ISSN: 2468-4376
https://www.jisem-journal.com/

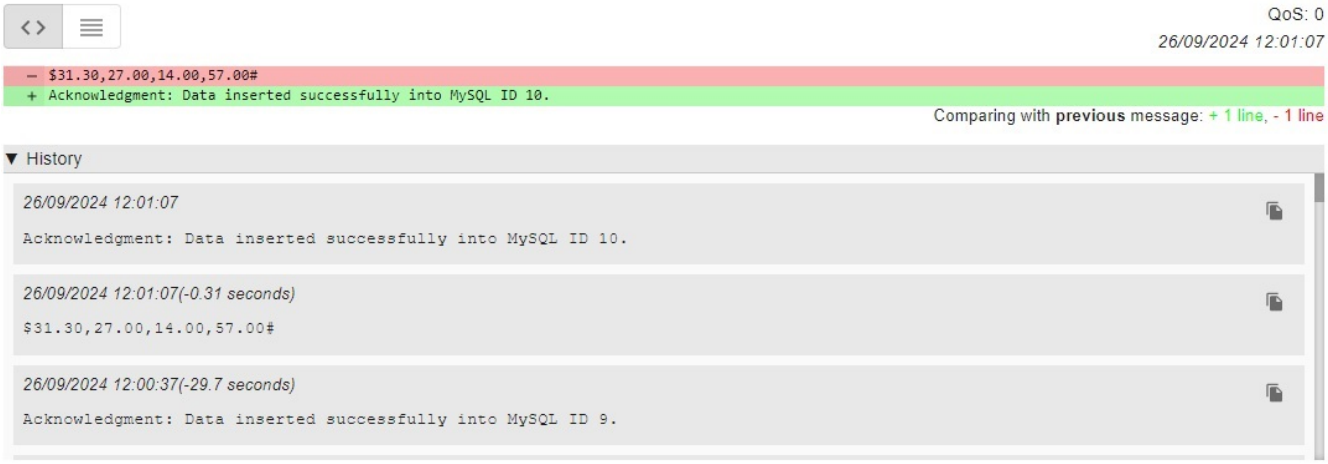
Research Article

MQTT is widely regarded as one of the most efficient communication protocols for IoT applications, especially those involving resource-constrained devices or systems that operate in environments with limited network bandwidth. MQTT operates on a publish/subscribe model, which reduces the need for constant device-to-server communication, enabling more scalable and bandwidth-efficient systems.

```
11:54:07.264 -> DHTxx test!
11:54:11.106 -> WiFi connected
11:54:11.106 -> IP address: 192.168.0.111
11:54:11.154 -> The client SK_WS-E0:E2:E6:CD:0F:E4 connects to the public MQTT broker
11:54:11.742 -> ESP32 Connected with insghtts broker
11:54:11.742 -> Ping succesful.
11:54:37.233 -> Sensor Task
11:54:37.358 -> ADC Value = 3377, ADC Voltage = 2.72, Brightness % = 17.00    Humidity: 38.00%  Temperature: 29.80°C
11:54:37.400 -> $29.80,38.00,17.00,57.00#
11:55:07.233 -> Sensor Task
11:55:07.360 -> ADC Value = 3375, ADC Voltage = 2.72, Brightness % = 17.00    Humidity: 32.00%  Temperature: 28.80°C
11:55:07.399 -> $28.80,32.00,17.00,57.00#
11:55:37.233 -> Sensor Task
11:55:37.358 -> ADC Value = 3400, ADC Voltage = 2.74, Brightness % = 16.00    Humidity: 31.00%  Temperature: 29.00°C
11:55:37.406 -> $29.00,31.00,16.00,57.00#
11:56:07.273 -> Sensor Task
11:56:07.396 -> ADC Value = 3483, ADC Voltage = 2.81, Brightness % = 14.00    Humidity: 31.00%  Temperature: 29.20°C
11:56:07.396 -> $29.20,31.00,14.00,57.00#
11:56:37.273 -> Sensor Task
11:56:37.356 -> ADC Value = 3432, ADC Voltage = 2.77, Brightness % = 16.00    Humidity: 31.00%  Temperature: 29.40°C
11:56:37.398 -> $29.40,31.00,16.00,57.00#
11:57:07.231 -> Sensor Task
11:57:07.402 -> ADC Value = 3402, ADC Voltage = 2.74, Brightness % = 16.00    Humidity: 30.00%  Temperature: 29.60°C
11:57:07.402 -> $29.60,30.00,16.00,57.00#
11:57:37.268 -> Sensor Task
11:57:37.356 -> ADC Value = 3411, ADC Voltage = 2.75, Brightness % = 16.00    Humidity: 32.00%  Temperature: 29.80°C
11:57:37.391 -> $29.80,32.00,16.00,57.00#
```

ESP32 Client Module Output

HTTP is the most widely used protocol on the internet and has become the default communication method for many IoT devices. However, its request/response model and relatively high overhead make it less efficient for IoT applications that require continuous data transmission or real-time communication. HTTP is best suited for IoT systems where devices need to fetch or push data from/to a web server, such as smart home devices, where the interaction with the system is primarily human-centric.



MQTT Explorer Output

In terms of bandwidth usage, HTTP is less efficient than MQTT, as each request sent by the device includes larger headers and requires more data to be transmitted over the network. This can lead to significant bandwidth consumption in IoT systems with multiple devices or frequent data exchanges. Moreover, latency in HTTP-based

Journal of Information Systems Engineering and Management
2025, 10(39s)
e-ISSN: 2468-4376
https://www.jisem-journal.com/

Research Article

systems is higher compared to MQTT because HTTP establishes a new connection for each request, adding overhead to the communication process.

ID ▲ 1	Time Stamp	Battery	Temperature	Humidity	LightIntensity
1	2024-09-26 11:55:09	57.00	29.40	31.00	16.00
2	2024-09-26 11:55:39	57.00	29.60	30.00	16.00
3	2024-09-26 11:56:09	57.00	29.80	32.00	16.00
4	2024-09-26 11:56:39	57.00	30.10	30.00	13.00
5	2024-09-26 11:57:09	57.00	30.30	30.00	14.00
6	2024-09-26 11:57:39	57.00	30.50	48.00	15.00
7	2024-09-26 11:58:09	57.00	30.70	28.00	17.00
8	2024-09-26 11:58:39	57.00	31.10	27.00	17.00
9	2024-09-26 11:59:09	57.00	31.30	26.00	16.00
10	2024-09-26 11:59:39	57.00	31.30	27.00	14.00

Php Mysql database Table

On the other hand, HTTP offers robust security features when combined with SSL/TLS encryption, as in HTTPS. HTTPS provides end-to-end encryption, ensuring that data transmitted between devices and servers is secure from potential threats. This makes HTTP/HTTPS a good choice for IoT applications that deal with sensitive data, such as healthcare systems or financial services, where the integrity and confidentiality of the data are paramount.

Reliability in HTTP is also high, as the protocol is built to ensure that requests are successfully completed before moving on to the next task. However, unlike MQTT, HTTP does not offer different levels of delivery assurance, which can be a disadvantage in systems that require more granular control over message delivery. In conclusion, while HTTP is a versatile and secure protocol, it is less suitable for IoT applications that require real-time communication, low bandwidth usage, or power efficiency. However, for applications where the interaction between devices and users takes place through a web interface, HTTP or HTTPS remains a reliable and widely-supported option.

UDP is known for its speed and low latency, making it ideal for real-time applications like video streaming or gaming, where data must be delivered quickly, even if some packets are lost. UDP is a connectionless protocol, meaning it does not establish a handshake between the sender and receiver before transmitting data. This reduces overhead and enables faster communication, but at the cost of reliability.

In IoT applications, UDP can be useful in scenarios where speed is prioritized over reliability, such as in applications that tolerate some degree of packet loss, like sensor networks that send frequent updates. However, for most IoT systems, where data integrity is essential, UDP's lack of delivery guarantees and error-checking mechanisms make it less suitable.

The choice of communication protocol for IoT applications depends on the specific requirements of the system in terms of bandwidth, latency, power consumption, security, and reliability. MQTT stands out as the most efficient protocol for resource-constrained devices, offering low bandwidth usage, low latency, and moderate security with high reliability, making it ideal for applications such as remote monitoring and sensor networks. HTTP and HTTPS, while more widely used in

The best communication protocol for an IoT system depends on the specific requirements of the application. MQTT is the most suitable for systems that require low bandwidth usage, low power consumption, reliability, and scalability, making it ideal for sensor networks, industrial monitoring, and remote control systems. HTTP/HTTPS is the best choice for IoT applications that involve human interaction or require secure data transmission, such as smart home devices or healthcare applications. Finally, UDP is the fastest option but should be reserved for applications where low latency is more important than data integrity, such as real-time streaming or gaming systems.

Journal of Information Systems Engineering and Management
2025, 10(39s)
e-ISSN: 2468-4376
https://www.jisem-journal.com/

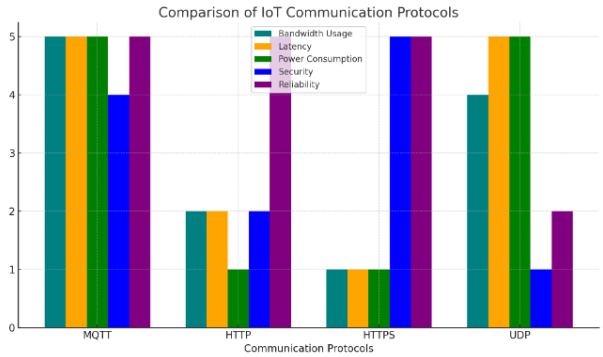
Research Article

By carefully evaluating the trade-offs between efficiency, security, and reliability, IoT developers can choose the most appropriate communication protocol for their specific needs, ensuring that their system performs optimally in the target environment.

3.1 Comparison Table:

Protocol	Bandwidth Usage	Latency	Power Consumption	Security	Reliability	Use Case
MQTT	Low	Very Low	Low	TLS support	High (with QoS)	Real-time, sensor networks, remote monitoring
HTTP	High	High	High	Low (without HTTPS)	High	Web services, non-real-time systems
HTTPS	Higher	Higher	Higher	SSL/TLS	High	Secure web services, sensitive data transfer
UDP	Very Low	Very Low	Very Low	None	Low	Streaming, gaming, fast data transmission

Comparison Table



Communication Protocol Comparison Analysis

Here is a bar graph comparing the performance of various IoT communication protocols—MQTT, HTTP, HTTPS, and UDP—across five key metrics:

Bandwidth Usage: MQTT and UDP have the most efficient bandwidth usage, while HTTPS uses the most.

Latency: MQTT and UDP show the lowest latency, making them ideal for real-time applications, while HTTP/HTTPS have higher latency due to the request-response nature and encryption overhead.

Power Consumption: MQTT and UDP consume less power, making them more suitable for battery-powered IoT devices, while HTTP and HTTPS consume more due to their overhead.

Security: HTTPS offers the highest level of security, while MQTT provides moderate security with TLS support, and UDP lacks built-in security.

Research Article

Reliability: MQTT, HTTP, and HTTPS are reliable due to their connection-based nature and support for message integrity. UDP, being connectionless, is less reliable.

4. CONCLUSION

MQTT is the most efficient protocol for IoT applications, particularly in cases where low bandwidth, low power consumption, and real-time communication are essential. Its lightweight nature and support for QoS and security via TLS make it ideal for sensor networks, smart homes, and industrial IoT systems. HTTP/HTTPS is better suited for web-based IoT services where human interaction or secure communication is required, but it suffers from higher latency and power consumption. UDP is extremely fast and efficient in applications where packet loss is acceptable, but it lacks reliability and built-in security features.

REFERENCES:

[1] V. Raghunathan, C. Schurgers, S. Park, and M. Srivastava. Energy aware wireless microsensor networks. IEEE Signal Processing Magazine, 19(2):40–50, 2002.
[2] M. Chernyshev, Z. Baig, O. Bello, S. Zeadally "Internet of Things (IoT): Research, Simulators, and Testbeds", IEEE Internet of Things Journal(2018),DOI: 10.1109/JIOT.2018.2796542
[3] Malti Bansal, Priya "Performance Comparison of MQTT and CoAP Protocols in Different Simulation Environments", Springer(2020),DOI: 10.1007/978-981-15-7345-3_47
[4] SB Upadhyay, A. Bhattacharyya, "Lightweight Internet Protocols for Web Enablement of Sensors Using Constrained Gateway Devices",IEEE ICNC Conference(2013),DOI: 10.1109/ICNC.2013.6504058
[5] D. Thangavel, X. Ma, A. Valera, H.-X. Tan, CK-Y Tan, "Performance Evaluation of MQTT and CoAP via a Common Middleware",IEEE Conference on Intelligent Sensors (2014) DOI: 10.1109/ISSNIP.2014.6827678
[6] AP Fournaris, S. Giannoulis, C. Koulamas, "Evaluating CoAP End-to-End Security for Constrained Wireless Sensor Networks",IFIP Conference on New Technologies(2019) DOI: 10.23919/NTMS.2019.8763857
[7] K. Govindan, A.P. Azad, "End-to-End Service Assurance in IoT with MQTT-SN",IEEE Consumer Communications and Networking Conference(2015),DOI: 10.1109/CCNC.2015.7157991
[8] R. Ferdousi, M. Helaluddin, A. Akther, K.M. Alam, "An Empirical Study of CoAP Based Service Discovery Methods for Constrained IoT Networks using Cooja simulator",IEEE International Conference on Computer and Information Technology(2017), DOI: 10.1109/ICCITECHN.2017.8281784
[9] N.R.A. Abosata, A.H. Kemp, M. Razavi, "Secure Smart-Home Application Based on IoT CoAP Protocol", International Conference on Internet of Things(2019), DOI: 10.1109/IOTSMS48152.2019.8939199
[10] Ludovici, P. Moreno, A. Calveras, "TinyCoAP: A Novel Constrained Application Protocol (CoAP) Implementation for Embedding RESTful Web Services in Wireless Sensor Networks Based on TinyOS", Journal of Sensors and Actuator Networks(2013),DOI: 10.3390/jsan2020288
[11] Dunkels, B. Gronvall, T. Voigt, "Lightweight and Flexible Operating System for Tiny Networked Sensors",IEEE Conference on Local Computer Networks(2004),DOI: 10.1109/LCN.2004.38
[12] Samer Hamdani, Hassan Sbeyti, "A Comparative study of COAP and MQTT communication protocols",2019 7th International Symposium on Digital Forensics and Security (ISDFS), July 2019, DOI: 10.1109/ISDFS.2019.8757486
[13] Urs Hunkeler, Hong Linh Truong, and Andy Stanford-Clark, "MQTT-S — A publish/subscribe protocol for Wireless Sensor Networks",3rd International Conference on Communication Systems Software and Middleware and Workshops (COMSWARE '08), January 2008,DOI: 10.1109/COMSWA.2008.4554519
[14] Zafar, S., Miraj, G., Baloch, R., Murtaza, D., & Arshad, K. (2018). An IoT Based Real-Time Environmental Monitoring System Using Arduino and Cloud Service. Engineering, Technology & Applied Science Research.DOI:10.48084/ETASR.2144
[15] Syahmi Md Dzahir, M.A., & Seng Chia, K. (2023), "Evaluating the Energy Consumption of ESP32 Microcontroller for Real-Time MQTT IoT-Based Monitoring System" 2023 International Conference on Innovation and Intelligence for Informatics, Computing, and Technologies (3ICT), 255-261.DOI:10.1109/3ICT60104.2023.10391358
[16] Firmansah A., Aripriharta Wirawan I.M., Herwanto H.W., Fadlika I., & Muladi (2019), "Design and Experimental Validation of the Self-powered IoT for Indoor Temperature-Humidity Monitoring", 2019

Research Article

International Conference on Electrical, Electronics and Information Engineering (ICEEIE), 6, 139-143.DOI:10.1109/ICEEIE47180.2019.8981426

[17] Sholihul M., Faculty H., Ari I., Zaeni E., Nugraha P.A., Mizar M.A., & Irvan M. (2020), "IoT Based Smart Garden Irrigation System", 2020 4th International Conference on Vocational Education and Training (ICOVET), 361-365.DOI:10.1109/ICOVET50258.2020.9230197

[18] Syahmi Md Dzahir, Muhammad Arif and Kim Seng Chia. "Evaluating the Energy Consumption of ESP32 Microcontroller for Real-Time MQTT IoT-Based Monitoring System." 2023 International Conference on Innovation and Intelligence for Informatics, Computing, and Technologies (3ICT) (2023): 255-261.DOI:10.1109/3ICT60104.2023.10391358

[19] Spachos, Petros and Dimitrios Hatzinakos. "Self-Powered Wireless Sensor Network for Environmental Monitoring." 2015 IEEE Globecom Workshops (GC Wkshps) (2015): 1-6., DOI:10.1109/GLOCOMW.2015.7414207

[20] Farooq, M. U. et al. "A Review on Internet of Things (IoT)." International Journal of Computer Applications 113 (2015): 1-7.DOI:10.5120/19787-1571

[21] Miraz, Mahdi H. et al. "A review on Internet of Things (IoT), Internet of Everything (IoE) and Internet of Nano Things (IoNT)." 2015 Internet Technologies and Applications (ITA) (2015): 219-224. DOI:10.1109/ITechA.2015.7317398