การประชุมวิชาการทางวิศวกรรมไฟฟ้า ครั้งที่ 48
The 48th Electrical Engineering Conference (EECON-48)
วันที่ 19-21 พฤศจิกายน 2568 ณ โรงแรมฟูราม่า จังหวัดเชียงใหม่

Comparative Analysis of MQTT and MQTT-SN Protocols in Wireless Sensor Network

Juthathip Wannawan[1], Weeraphon Yana[2], Buntueng Yana[3] *

[1]Department of Electrical Engineering, School of Engineering, University of Phayao, Thailand,
juthathipwannawan@gmail.com[1], weeraphonyana@gmail.com[2], mr.buntueng@gmail.com[3*]

Abstract

The growth of resource-constrained devices within the Internet of Things (IoT) and Wireless Sensor Networks (WSNs) necessitates the use of highly efficient communication protocols. This study compares the performance of the MQTT protocol and its lighter version, MQTT-SN, on the popular ESP32 platform. The goal is to see if they are good enough for real-world IoT applications by testing important performance metrics including latency, throughput, and packet loss with different packet sizes and Quality of Service levels. An experimental testbed was established using an ESP32 microcontroller as the client. It communicates with a host PC running a Mosquitto broker and a Paho MQTT-SN gateway. The experimental results support that MQTT-SN offers significant performance advantages in terms of speed and efficiency. It reliably shows lower average latency and higher data throughput compared to standard MQTT across most tested conditions. The findings suggest that the option between MQTT and MQTT-SN presents a clear engineering trade-off. MQTT-SN is the more suitable option for applications prioritizing low latency and high efficiency with small data packets. The present resilience of standard MQTT makes it more suitable for applications requiring guaranteed delivery of larger data payloads.

Keywords: MQTT, MQTT-SN, Internet of Things, Wireless Sensor Networks

1. Introduction

Wireless Sensor Networks (WSNs) are now a basic technology for gathering data in places like industries, farms, hospitals, and cities. A typical WSN has a lot of small sensor nodes that are spread out over a wide area. Each node can sense, process, and communicate. Because these nodes often have very limited power and memory, the choice of communication protocol is crucial. For these small devices, standard internet protocols like HTTP and FTP are often too demanding[1]. As a result, the field has moved toward lighter options like CoAP, 6LoWPAN, and Message Queuing Telemetry Transport (MQTT), which are all designed to work within tight power, memory, and processing limits[2].

MQTT is a popular option in the messaging protocol space. The publish-subscribe model works well for many Internets of Things applications, and this protocol is lightweight. It is a good choice for data transmission between devices and the cloud due to features like its ease of use, low overhead, and support for various Quality of Service (QoS) levels[3]. However, MQTT still has a problem with TCP/IP networks. It needs a stable connection and good bandwidth. This reliance on TCP/IP[3] is a major hurdle in constrained of WSNs, where nodes often use UDP and have to deal with unreliable connections, low bandwidth, and a high chance of losing packets[2].

To solve the problems, MQTT-SN was created. It's an optimized version of MQTT that works well on non-TCP/IP networks like Zigbee, Bluetooth Low Energy (BLE), and 6LoWPAN, mostly by using UDP as its transport layer. One of its main improvements is the ability to pre-register a topic ID, which makes messages much smaller. It also includes a "sleep mode" for clients, saving a lot of power by letting devices disconnect and reconnect as needed. These features make MQTT-SN a better, more energy-efficient choice for battery-powered sensors working in difficult wireless conditions[4].

This study implements a WSN using MQTT and MQTT-SN to demonstrate real-world performance. We considered ESP32 microcontrollers to implement the MQTT and MQTT-SN protocols in our configuration. The ESP32 is a widespread choice for IoT projects[5]. Since it has built-in Wi-Fi and Bluetooth. Then we conducted performance tests, comparing MQTT-SN directly to regular MQTT. The point is to evaluate the performance of MQTT-SN by measuring latency, throughput, and packet loss rate.

2. Literature Review

Advancements in digital electronics and wireless communications have enabled the significant rise of the field of WSNs. The setting up of a WSN is about the deployment of multiple sensor nodes. Their devices are tiny and low power. They can sense, compute, and communicate. To ensure the durability and efficiency of a WSN, it is important to take into account several major parameters during the design process. These include network topologies such as star, tree, or mesh. These influences connectivity, communication paths, and energy efficiency[6]. It may have a problem because the sensor nodes often rely on batteries and are deployed in remote or hardly to access areas. In addition, some factors such as scalability, security, and QoS are important. It has a function to maintain data reliability and timely delivery. The design strategy must consider the specific application context, whether terrestrial, underground, or mobile[7].

WSNs rely on a variety of protocols. Each operating protocol is at a specific layer in the network stack. At the data-link layer, WSNs are often dependent on MAC protocols such as Carrier Sense Multiple Access (CSMA) and Time Division Multiple Access (TDMA). It used to coordinate which nodes transmit at any given time. This control factors reduces the chance of collisions and reduce

*Corresponding Author

energy usage[8]. At the network layer, routing protocols are used to determine. It helps data to travel from individual sensor nodes to the base station. Reactive protocols like Ad hoc On-Demand Distance Vector (AODV) and Dynamic Source Routing (DSR) create routes only when they are suitable for dynamic or frequently changing network needed. This strategy makes topologies[9]. In addition to these, several standard protocol stacks have become common. It includes Zigbee, which is based on the IEEE 802.15.4 standard. It is widespread for low power consumption and mesh networking capabilities system. 6LoWPAN is one of the standard protocol tools. It enables IPv6 packets to be carried over low-power wireless networks[10, 11].

The MQTT protocol has developed into a common messaging protocol for the IoT. It has been widely adopted in WSN applications. MQTT is a lightweight, publish-subscribe protocol that operates on top of TCP/IP. Its design is optimized for environments with high latency and low bandwidth. MQTT can also handle many sensor networks. The publish-subscribe model decouples publishers to subscribers via a central broker. It also allows for scalable and flexible one-to-many and many-to-one communication patterns. This architecture simplifies the communication logic for resource-constrained sensor nodes [12].

In WSNs where traditional the TCP/IP stacks are impractical. Some protocol such as Zigbee, Bluetooth, and MQTT-SN offers a more suitable choice. MQTT-SN is specifically designed to be a lightweight protocol for machine-to-machine communication in environments where holding a TCP/IP stack. It is unfeasible because of limitations in memory, computing capacity, or power availability[13]. It keeps the fundamental publish-subscribe architecture from MQTT. But it modified to function over connectionless protocols such as UDP. A core architectural component of MQTT-SN is the gateway. It connects the MQTT-SN clients in the sensor network to the MQTT broker [1] that is either in the cloud or on a local server. This gateway translates between MQTT-SN to MQTT protocols. This allows sensor nodes to reliably forward data into broader IoT ecosystems through the MQTT-SN gateway.

MQTT-SN has several advantages compared to the standard MQTT protocol. It is specifically designed for battery-operated and resource-constrained wireless sensor networks. The most significant benefit is the reduction in packet size. Instead of using long, human-readable topic names in every published message. MQTT-SN allows clients to register a topic name with the gateway and receive a shorter two-byte topic ID. Subsequent communications utilize this topic ID, significantly reducing the overhead. Consequently, the energy required for transmission decreases[11]. Furthermore, MQTT-SN is transport-agnostic and is commonly used over UDP. It is more lightweight than the connection-oriented TCP used by MQTT[3]. MQTT-SN includes features designed for sleeping devices. It includes an offline keep-alive mechanism, in which the gateway buffers messages for sleeping clients until they reconnect. A further benefit is the gateway finding technique. It enables clients to dynamically discover a gateway inside the network without previous configuration of its address. These features collectively make MQTT-SN more efficient and practical choice for many WSN deployments[13, 14].

3. System Design

This section speaks about how the system was set up for the performance test. There are two different test settings for the experimental work. One is for the regular MQTT protocol, and the other is for the MQTT-SN protocol. Each environment is meant to look at the unique features of the protocols being compared.

3.1 MQTT Protocol

In this architecture, the client device communicates directly with the broker. Its working principle is as follows:

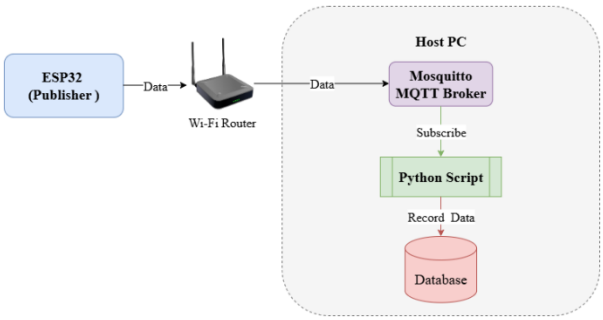

Fig. 1 MQTT Protocol model

This setup shows the direct communication line that a client device uses to submit data through a central broker. The ESP32, which acts as the client, starts the process by making the data payload. Then it connects to the local network through a Wi-Fi router, which is the device that sends data over the LAN to its final destination, the Host PC.

The Host PC is the main computer that handles all of the system's processing. The MQTT Broker, which is the main part of the system that handles all message flow, runs on this machine. The ESP32 connects directly to this broker over TCP/IP to publish its message. This reliance on TCP/IP is particularly important because its confident delivery method is what makes the system so reliable. The subscriber runs on the same host computer. It subscribes to the MQTT topic and establishes a connection with the broker. Upon receiving a message, the broker promptly forwards it to the script. The script checks the data, then figures out the throughput and latency and saves the results in MariaDB.

3.2 MQTT-SN Protocol

This architecture is adapted for resource-constrained devices, with the gateway as a key additional component.

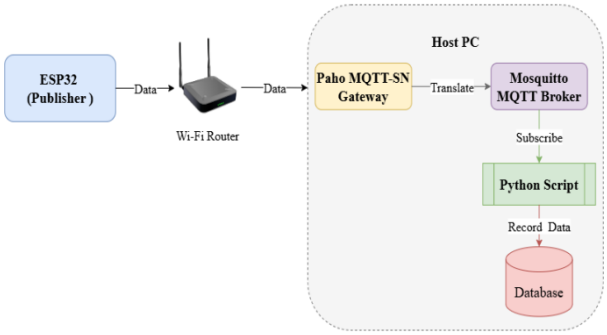

Fig. 2 MQTT-SN Protocol model

The proposed system contains ESP32 nodes, Router, and computer to run python script and software. The main change to this architecture is the addition of a gateway. The ESP32 client cannot connect directly to the MQTT broker. Instead, it sends MQTT-SN messages to the Gateway. Then, the Gateway changes these MQTT-SN messages into standard MQTT messages and sends them over TCP to the MQTT Broker. The Broker views the Gateway as just another MQTT client.

4. Methodology

The methodology for this research is experimental, based on a quantitative performance comparison between the MQTT and MQTT-SN protocols. The evaluation is structured around three distinct tests, including latency, throughput, and packet loss rate. It is all designed to analyze the performance of the protocols under different scenarios. The data gathered from these tests will form the basis for our conclusions, allowing for a direct comparison to determine the suitability of each protocol for resource.

Table 1: Independent Variables

Parameter	**Specification / Value**
Protocol Tested	MQTT, MQTT-SN
Quality of Service	0, 1, 2
Payload Size	20, 60, 100, 140, 180, 220 bytes

Table 1 shows the variables that were adjusted in each experiment. We run following a structured approach to observe and measure their impact on protocol performance such as latency, throughput, and packet loss.

Table 2: Controlled Variables

Parameter	Specification / Value
Client Device	ESP32-WROOM-32 Development Board
MQTT Broker	Mosquitto v2.0.22
MQTT-SN Gateway	Eclipse Paho MQTT-SN Gateway v1.1
Data Collection Script	Python v3.13.3 with Paho-MQTT Library
Database	MariaDB v11.8.2
Messages per Run	500 messages
Time Synchronization	ESP32 synchronized time with NTP server
Client Libraries	PubSubClient(MQTT); WiFiUDP client (MQTT-SN)
Network	5 GHz Wi-Fi

Table 2 lists parameters and components that were maintained constant throughout the experiment. This ensures that the comparison between different conditions is impartial and accurate. linking any observed changes in results solely to the effects of the independent variables

4.1 Latency Measurement

We utilized timestamps to verify latency in this experiment to ensure its accuracy. The initial action of the ESP32 is to synchronize with an NTP server.[15]. The payload contains the current time in milliseconds, the Message ID, and the packet size for each transmission. The Host PC, serving as the data receiver, communicates the experimental results upon receiving this payload over MQTT or MQTT-SN. The computer logs the timestamp upon message receipt. To ascertain the transmission latency, one must compare the arrival time with the message timestamp. MariaDB retains the outcomes. Numerous test sets for the experiment are predicated on specific conditions. MQTT and MQTT-SN were employed, utilizing QoS levels 0, 1, and 2 for MQTT, with packet sizes varying from 20 to 220 bytes. The experiment's packet size was limited to a maximum of 220 bytes because the configuration used a single-octet Length field, which supports a total message length of up to 255 bytes. Testing larger sizes would be outside the specification of this length-encoding mechanism. In every experiment, 500 messages are transmitted continuously to ensure the statistical reliability of the data. The data was subsequently utilized to calculate the mean, minimum, maximum, and standard deviation of the delay. Graphs facilitate the comparison of packet sizes and their respective arrival times. It demonstrates the efficacy of each regimen.

4.2 Throughput Measurement

This test evaluates the volume of data the ESP32 device can transmit to the PC gateway within a certain timeframe. The PC does the measurements, alleviating the ESP32 of this task due to its limited resources. The computer's Python program subscribes to the ESP32's

data topic. The script enumerates the number of messages successfully received throughout each testing interval. It monitors all messages, from the initial to the final. The mean data transfer rate in messages per second is derived from this data. It demonstrates the system's operational speed across several scenarios. The protocol (MQTT/MQTT-SN), the QoS levels (0, 1, 2), and the payload size (20–220 bytes) exemplify certain scenarios. MariaDB retains the outcomes. Subsequently, analyze throughput in relation to packet size or Quality of Service (QoS) level and construct a graph to evaluate the efficacy of each protocol.

4.3 Packet Loss Measurement

We used sequential message identifiers to assess protocol transmission reliability. Each experiment had the ESP32 client send 500 messages. Each message's payload had a unique, incrementing message ID (1–500). It gave the receiver a predictable sequence. The host PC Python subscriber script collected and logged all incoming messages and message IDs. Breaks in the received number sequence were used to quantify packet loss. The packet loss rate was calculated by comparing the total number of messages received to the 500 delivered messages[16]:

$$PLR = \frac{(TTM - TRM)}{TTM} X100\% \qquad (1)$$

PLR is a packet loss rate, TTM is a total transmitted message, and TRM is the total received messages. This measurement was systematically performed for every combination of protocol MQTT, MQTT-SN, Quality of Service (QoS) level, and payload size. These results are allowing for a direct and accurate comparison of their reliability under varying operational conditions

5. Result

Using graphs, this section compares the performance of MQTT and MQTT-SN protocols experimentally.

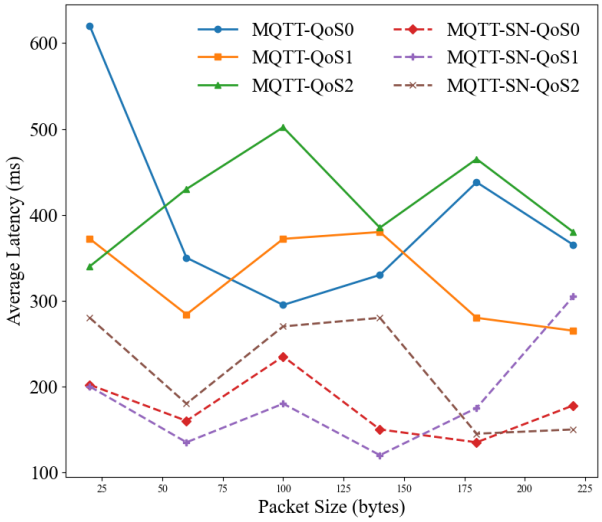

Fig.3 Latency vs. Packet Size of MQTT and MQTT-SN

5.1 Latency

Fig.3 indicates that the average latency of almost all MQTT-SN protocols is lower than the average latency of the standard MQTT protocol at low packet size, because MQTT-SN is lightweight and uses UDP for communication, which is faster than MQTT's use of TCP.

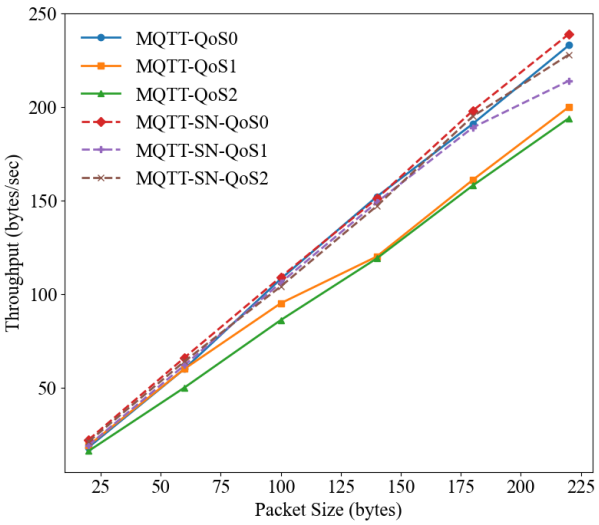

Fig.4 Throughput vs Packet Size of MQTT and MQTT-SN

5.2 Throughput

Fig. 4 shows that the throughput for both MQTT and MQTT-SN keeps going up as the packet size becomes bigger. In the first range with smaller packets, MQTT-SN always has a higher throughput than MQTT, which means it is better for sending small amounts of data at all QoS levels. This is because MQTT-SN is a lightweight protocol that uses a short topic ID instead of a long topic name, which uses less data, and it typically runs over the faster UDP protocol.

5.3 Packet Loss

The results show that for packet sizes ranging from 20 to 220 bytes, there was zero packet loss across all QoS levels for both protocols.

6. Conclusions

This study presented a performance comparison of the MQTT and MQTT-SN protocols on the resource constrained ESP32 platform. The purpose was to assess their applicability to IoT applications by measuring key performance metrics.

The experimental findings showed that MQTT-SN had big performance benefits in terms of speed and efficiency. For tiny to medium-sized packets, it always had reduced average latency and higher data throughput. The lightweight UDP transport protocol and topic ID registration technique used by MQTT-SN to cut down on packet overhead were both tested and found to work as planned. MQTT-SN is the best solution for apps that need to be fast and use as little bandwidth as possible.

In the end, the decision between MQTT and MQTT-SN comes down to a trade-off between speed and dependability. The conventional MQTT protocol via TCP is the better solution for any application where guaranteed delivery of large payloads is important. MQTT-SN works well in low-power, resource-constrained applications where low latency for small data packets is important.

7. References

[1] Al-Fuqaha, A., et al., Internet of Things: A Survey on Enabling Technologies, Protocols, and Applications. IEEE Communications Surveys & Tutorials, 2015. 17(4): p. 2347-2376.
[2] Yalli, J., et al., A Systematic Review For Evaluating IoT Security: A Focus On Authentication, Protocols and Enabling Technologies. IEEE Internet of Things Journal, 2025. PP: p. 1-1.
[3] Akshatha, P.S., S.M.D. Kumar, and K.R. Venugopal, MQTT Implementations, Open Issues, and Challenges: A Detailed Comparison and Survey. International Journal of Sensors, Wireless Communications and Control, 2022. 12(8): p. 553-576.
[4] Stangaciu, V., et al., Integrating Real-Time Wireless Sensor Networks into IoT Using MQTT-SN. Journal of Network and Systems Management, 2025. 33.
[5] Amirkhanov, B., et al., Evaluating HTTP, MQTT over TCP and MQTT over WEBSOCKET for digital twin applications: A comparative analysis on latency, stability, and integration. International Journal of Innovative Research and Scientific Studies, 2025. 8(1): p. 679-694.
[6] Singla, P. and A. Munjal, Topology Control Algorithms for Wireless Sensor Networks: A Review. Wirel. Pers. Commun., 2020. 113(4): p. 2363–2385.
[7] Khatami, S.S., et al. Energy-Efficient and Secure Double RIS-Aided Wireless Sensor Networks: A QoS-Aware Fuzzy Deep Reinforcement Learning Approach. Journal of Sensor and Actuator Networks, 2025. 14, DOI: 10.3390/jsan14010018.
[8] Olatinwo, D.D., et al., Energy-Efficient Multichannel Hybrid MAC Protocol for IoT-Enabled WBAN Systems. IEEE Sensors Journal, 2023. 23(22): p. 27967-27983.
[9] Tan, X., et al., Performance Analysis of Routing Protocols for UAV Communication Networks. IEEE Access, 2020. 8: p. 92212-92224.
[10] Alyami, S., R. Alharbi, and F. Azzedin Fragmentation Attacks and Countermeasures on 6LoWPAN Internet of Things Networks: Survey and Simulation. Sensors, 2022. 22, DOI: 10.3390/s22249825.
[11] Bruniaux, A., et al. Defragmenting the 6LoWPAN Fragmentation Landscape: A Performance Evaluation. Sensors, 2021. 21, DOI: 10.3390/s21051711.
[12] Shahrokhi, A. and M. Ahmadi. Power Evaluation of IOT Application Layer Protocols. in 2023 7th International Conference on Internet of Things and Applications (IoT). 2023.
[13] Roldán-Gómez, J., et al. Security Analysis of the MQTT-SN Protocol for the Internet of Things. Applied Sciences, 2022. 12, DOI: 10.3390/app122110991.
[14] Palmese, F., A.E.C. Redondi, and M. Cesana, Adaptive Quality of Service Control for MQTT-SN. Sensors (Basel), 2022. 22(22).
[15] Gamage, K.A.A., et al. A Dynamic Framework for Internet-Based Network Time Protocol. Sensors, 2024. 24, DOI: 10.3390/s24020691.
[16] Sonklin, K. and C. Sonklin, A performance evaluation of the internet of things-message queue telemetry transport protocol based water level warning system. International Journal of Electrical and Computer Engineering (IJECE), 2024. 14: p. 7178.

Juthathip Wannawan She is an undergraduate student in the Faculty of Engineering, majoring in Electrical Engineering at the University of Phayao, Thailand. Current research interests: Electrical Communication System

Weeraphon Yana He is an undergraduate student in the Faculty of Engineering, majoring in Electrical Engineering at the University of Phayao, Thailand. Current research interests: Electrical Communication System

Buntueng Yana (Member, IEEE) received B.Eng. and M.Eng. degrees in Electrical Engineering from Chiang Mai University, Thailand, in 2006 and 2009, respectively, and a Ph.D. in Information Science and Technology from Osaka University, Japan, in 2019. He is currently a lecturer with the Department of Electrical Engineering at the University of Phayao, Thailand, where he began his academic career. His research interests include robotics, automatic control, power generation, and the application of machine learning to medical applications.