Journal of Theoretical and Applied Information Technology
15st September 2026. Vol.104. No.17

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

LOW-OVERHEAD, SERVER-ASSISTED TIME SYNCHRONIZATION MECHANISM FOR MQTT-SN–BASED IOT SENSOR NETWORKS

HOUSSEIN WEHBE[1], ALI MCHEIK[2], BILAL KANSO[2,3], LOREEN BAKERR[4]

[1] College of Engineering, American University of Iraq - Baghdad (AUIB), Iraq
[2] Faculty of Sciences, Lebanese University, Beirut, Lebanon
[3] School of Arts and Sciences, Lebanese International University, Nabatieh, Lebanon
[4] Faculty of Sciences, Lebanese University, Beirut, Lebanon
E-mail: [1] houssein.wehbe@auib.edu.iq, [2] ali_mch@hotmail.com, [3] bilal_kanso@hotmail.com, [4] loreenbakerr@gmail.com

ABSTRACT

In many Internet of Things (IoT) systems, low-power sensors distributed in the environment continuously collect data from their surroundings and forward it to an application backend, where it is analyzed and acted upon. Including a timestamp in every reading helps keep events consistent and properly ordered, but requires the sensors and server clocks to be synchronized. This common challenge is usually solved by relying on frequent exchanges of dedicated timing messages between nodes, which increase network traffic and processing overhead on resource-constrained sensors. In this paper, we introduce a lightweight time synchronization mechanism for IoT deployments using MQTT-SN as the data delivery protocol. It leverages existing MQTT-SN messages to carry data timestamps to a central server, which applies an adaptive algorithm to identify possible clock desynchronization, limit the effect of transient delays, and send clock-adjustment notifications to sensors only when necessary. By minimizing dedicated synchronization exchanges and shifting computation to the server, the proposed mechanism reduces additional communication traffic and limits processing at the sensor. Simulation results in OMNeT++ show lower or more stable clock offsets than SNTP during the later part of representative runs. The simulations also show approximately 96–98% fewer synchronization messages than SNTP without increasing simulated sensor energy consumption. Under approximately matched conditions, the mechanism produced lower average clock offsets than those reported for an MQTT-based approach.

Keywords: *Time Synchronization, MQTT-SN, Server-Assisted, Wireless Sensor Networks, Low Overhead*

1. INTRODUCTION

Nowadays, IoT systems are used widely in areas like smart cities, industrial automation, and healthcare monitoring [1], [2]. Different sensors or connected devices within these systems collect environmental or operational data, usually at regular intervals. The measurements are sent to a designated server located either at the network edge or in the cloud. To allow the server to correctly interpret the data, each reading must include its acquisition time. Without this information, common operations such as event correlation, joint compression, and anomaly detection become difficult to perform.

Maintaining precise timing across all sensors is often difficult in practice due to the behavior of their internal clocks. Such clocks are commonly implemented with low-cost quartz oscillators whose frequencies drift with temperature, component tolerances, and long-term aging [3]–[7]. Due to the cumulative effect of these small differences in frequency, over time sensor timestamps become out of sync. This random drift affects the reliability of any analysis subsequently performed on the measurements. As a result, the problem of time synchronization for low-power sensors continues to be a central issue in IoT systems design.

Many IoT systems try to handle this challenge by adopting well-known protocols such as the Network Time Protocol (NTP) [8], the Simple NTP (SNTP) [9], the Timing-sync Protocol for Sensor Networks (TPSN) [10], and the Flooding Time Synchronization Protocol (FTSP) [11]. These approaches provide acceptable clock accuracy at the price of frequent exchanges of timing messages among nodes. Each exchange consumes sensor energy and occupies part of the available bandwidth. The issue is aggravated when an application-layer

protocol such as Message Queuing Telemetry Transport (MQTT) [12], MQTT for Sensor Networks (MQTT-SN) [13], or the Constrained Application Protocol (CoAP) [14] is employed in parallel for data delivery. When synchronization and application messages are transmitted over the same network, they compete for bandwidth and energy. As a result, node lifetime and data throughput are reduced. A further issue with some existing solutions is that they require sensors to periodically measure Round-Trip Time (RTT) and estimate clock drift. These tasks may impose additional processing and memory requirements on low-power IoT devices. As a result, there is a need for synchronization mechanisms that achieve accurate timing while maintaining low communication and energy overhead.

This work targets IoT deployments that already use MQTT-SN to periodically deliver data from sensors to a server. We propose a server-assisted, low-overhead time synchronization mechanism that leverages existing application messages to carry timestamp information. The server maintains a sliding window of recent timestamp differences and applies a detection algorithm that distinguishes transient delay variations from possible clock desynchronization. Clock-adjustment notifications are sent only when needed, avoiding periodic dedicated synchronization traffic and limiting sensor-side synchronization processing.

The main contributions of this work are as follows. (i) A piggybacked mechanism for MQTT-SN that avoids periodic dedicated synchronization exchanges; (ii) a server-side algorithm with an adaptive sensitivity factor K; (iii) an event-driven notification policy that triggers when possible desynchronization is detected; (iv) an OMNeT++ evaluation of synchronization accuracy, communication overhead, and sensor energy consumption, including comparisons with SNTP and the reported SMQTT results.

The paper is organized as follows. Section 2 reviews related work. Section 3 presents the proposed mechanism and server-side detection algorithm. Section 4 reports the performance evaluation. Section 5 concludes.

2. RELATED WORK

The challenge of ensuring time synchronization between distributed devices has been extensively studied long before the emergence of IoT applications [5], [7], [15], [16]. Several approaches have been proposed in traditional computer networks, yet many of them are not suitable for sensor networks that operate under severe resource constraints. NTP [8] remains the most widely used solution to synchronize clocks of devices over packet-switched networks, using a hierarchical system of dedicated time servers. Each client interacts with a time server on a periodic basis by sending and receiving timestamped request/reply messages. These exchanges allow the client to determine the difference between its local clock and the time on the server. To reduce the effect of latency and jitter, clients may need to apply filtering techniques before adjusting their clocks. These techniques make NTP a relatively complex protocol. A lightweight alternative, SNTP [9], reduces the computational burden but still depends on the same client-server model, requiring periodic timing message exchanges. Both NTP and SNTP can introduce considerable communication overhead, which may limit their suitability for IoT systems where devices operate with very limited resources.

Common clock synchronization protocols also include TPSN, FTSP, and Reference Broadcast Synchronization (RBS), which are specifically designed for wireless sensor networks (WSNs). TPSN [10] requires sensors to be organized in the form of a spanning tree where each child node synchronizes with its parent using a two-way message exchange model. The drawback of this protocol is the need to continuously maintain the tree and reconstruct it at every topology change, which requires the exchange of extra control messages among sensors. In FTSP [11], [17], sensors rely on an elected reference node that periodically disseminates timestamped synchronization messages. Upon receiving these messages, sensors apply regression techniques to estimate offset and skew. These sensor-side computations, in addition to the continuous flooding mechanism, consume sensors' energy and generate high traffic, thereby limiting the scalability of FTSP. RBS [18] also requires a reference node but, unlike FTSP, broadcasts a message in the network at each synchronization event. Each receiver should exchange the message reception time with the other nodes. This allows every sensor to have enough information to compute offsets and adjust its clock accordingly. These exchanges, however, increase communication overhead, limiting the use of RBS to only small local broadcast domains.

Recent studies have also proposed synchronization mechanisms for WSNs based on scheduled packet exchanges and clock-control algorithms. The approach in [19] organizes sensors in a spanning-tree topology and represents the synchronization process using a state-space model that considers clock-frequency drift and communication delays. An output-feedback

controller is then used to adjust the clocks. The method in [20] uses packet-coupled oscillators and output-feedback control, while synchronization messages are exchanged through a time-division multiple access (TDMA) superframe. In [21], the number of exchanged synchronization messages is reduced through a one-way packet-exchange scheme. However, the method still uses TDMA scheduling and an adaptive proportional-integral controller to correct clock offset and skew.

Although these approaches improve synchronization and, in some cases, reduce the number of exchanged messages, they still depend on synchronization-specific exchanges and clock-adjustment processing at sensor nodes [19]–[21]. One way to eliminate this limitation is to leverage the messages from application-layer protocols that exist between sensors and servers to transmit synchronization information. This approach can only be implemented in cases where an application-layer protocol dedicated to resource-constrained systems is already in use. Selecting such a protocol is an essential step toward defining a synchronization mechanism that is compatible with it. Common options include Hypertext Transfer Protocol (HTTP), Advanced Message Queuing Protocol (AMQP), CoAP, and MQTT. Among these protocols, HTTP and AMQP rely on TCP as the transport-layer protocol. Neither protocol is suitable for a constrained network due to the high communication and processing overhead incurred by the creation of sessions and the necessary signaling messages. CoAP [14], [22], in turn, is a stateless protocol based on UDP, which follows a request/response model whereby the client must initiate a new request to the server for each interaction. Reliability is provided through the support of confirmable and acknowledgment messages. Nevertheless, confirmable exchanges can increase traffic in constrained environments. MQTT [12] uses a publish/subscribe model based on TCP and assumes the use of a broker to mediate communication among parties as well as provide session persistence. MQTT provides reliable publish/subscribe communication, but its use of TCP adds connection and signaling overhead that may be significant for constrained IoT devices. A lightweight variant of MQTT, MQTT-SN [13], was designed to better suit the limitations of low-power sensors. MQTT-SN adapts MQTT's publish/subscribe model to constrained sensor environments using compact messages over datagram-oriented networks. It ensures reliability through mechanisms such as quality-of-service levels, message acknowledgment, message persistence, and last will messages. MQTT-SN and CoAP provide different communication models and efficiency characteristics for constrained environments [23]. However, none of these application-layer protocols natively supports time synchronization.

Several studies have attempted to integrate a synchronization mechanism into communication protocols used in sensor networks [24]–[28]. Among these works is [25], in which the authors propose a protocol called SMQTT, designed for Industrial IoT (IIoT) devices that use MQTT for data transmission. In this approach, each sensor embeds a timestamp representing its local clock in the payload of every MQTT message sent to the server, and the server includes its local time in the corresponding response. After each exchange, the sensor must calculate the RTT and apply a specific algorithm to adjust its clock when necessary. Experimental results show that SMQTT yields higher efficiency than traditional synchronization protocols. However, this approach assumes that sensors can periodically perform statistical calculations and execute complex algorithms at the same time as collecting and transmitting data. This sensor-side processing may limit the suitability of SMQTT for resource-constrained IoT devices. Another proposal in [26] integrates timestamps into CoAP messages and shows higher efficiency than NTP. However, it still requires frequent message exchanges initiated by clients because CoAP inherently operates in a request-response manner. In addition, as CoAP does not have persistent sessions, the proposed approaches in [26], [27] have limitations when applied to many IoT systems.

These studies show that application-layer traffic can be used to carry timing information. However, the reviewed approaches still require repeated client-server exchanges, synchronization processing at the sensor, or both. These requirements add communication and processing overhead to the normal operation of the sensor. The research problem addressed in this study is how to maintain accurate synchronization in resource-constrained MQTT-SN-based sensor networks while limiting this additional overhead.

The objective of this study is therefore to propose and evaluate a time synchronization mechanism for MQTT-SN-based sensor networks. Accordingly, the study addresses two research questions. Can the proposed mechanism maintain synchronization accuracy comparable to SNTP and the reported SMQTT results? Can it reduce synchronization traffic compared with SNTP without increasing sensor energy consumption?

3. PROPOSED TIME SYNCHRONIZATION MECHANISM

We consider IoT systems where sensors periodically publish measurements to a server through MQTT-SN [13] and propose a time synchronization mechanism. The basic principle of our scheme is to exploit the sensors' MQTT-SN traffic for time information exchange and to perform desynchronization detection primarily at the server rather than at the distributed sensors.

We assume each sensor has a local clock to keep track of the acquisition time of each measurement. This timestamp is included in the payload portion of each outgoing MQTT-SN message without modifying the existing protocol specifications. Upon receiving a timestamped MQTT-SN PUBLISH message, the server executes the algorithm described in Algorithm 1 to identify possible clock desynchronization and issue a correction when needed. The server records the receipt time and calculates an observed timestamp difference as ObservedDifference = ReceiptTime − Timestamp. This sample is then added to a sliding window maintained by the server. The window stores the latest W observed timestamp differences. When a new sample is added and the window size exceeds W, the oldest sample is removed. Once the window is full, the server computes the mean value, denoted by AverageObservedDifference, and its standard deviation, denoted by σ, using the updated window. The former represents the average observed timestamp difference, while the latter represents the short-term variation. Using these values, the server attempts to distinguish usual variations in the observed timestamp difference from possible clock desynchronization at the sensor.

A desynchronization is declared only when the observed timestamp difference exceeds an adaptive threshold defined as

$$\text{Threshold} = \text{AverageObservedDifference} + K \times \sigma \quad (1)$$

where K acts as a sensitivity factor that determines how tolerant the algorithm is to delay variations.

A larger value of K widens the acceptable delay range, reducing unnecessary synchronization notifications, but a smaller value increases the algorithm's sensitivity and responsiveness to possible desynchronization. Under stable conditions, we set K = K_init = 2. When delay variability increases, the server makes the detector more sensitive by decreasing K by 0.5, subject to the minimum value K_min = 1. The server detects a change in timestamp-difference variability when the change in standard deviation between two successive windows exceeds a predefined instability threshold (Δσ_threshold = 2 ms), i.e., when $|\sigma - \text{prev}\_\sigma| > \Delta\sigma\_\text{threshold}$. Hence, K and, consequently, the threshold are dynamically tuned according to the observed network volatility.

Algorithm 1: Server-Side Desynchronization-Detection Algorithm and Adjustment Policy

Input:
- Messages: stream of timestamped sensor messages
- ServerTime: accurate server clock
- W: window size for timestamp-difference statistics
- K_init: initial sensitivity factor (default K_init = 2)
- K_min: minimum sensitivity factor (default K_min = 1)
- Δσ_threshold: predefined instability threshold (default Δσ_threshold = 2 ms)

Output:
- AdjustmentNotification: sent to each sensor if a clock correction is needed

Procedure:

```
For each sensor:
    Initialize ObservedDifferenceList ← empty
    K ← K_init
    prev_σ ← undefined
For each incoming message:
    ReceiptTime ← ServerTime
    SensorTime ← message.timestamp
    ObservedDifference ← ReceiptTime − SensorTime
    Append ObservedDifference to ObservedDifferenceList
    If size(ObservedDifferenceList) > W:
        Remove the oldest element from ObservedDifferenceList
    If size(ObservedDifferenceList) = W:
        AverageObservedDifference ←
                mean(ObservedDifferenceList)
        σ ← stddev(ObservedDifferenceList)
        If prev_σ is undefined:
            K ← K_init
        Else if |σ − prev_σ| > Δσ_threshold:
            K ← max(K_min, K − 0.5)
        Else:
            K ← K_init
        Threshold ← AverageObservedDifference + K × σ
        If ObservedDifference > Threshold:
            Send AdjustmentNotification to sensor
        prev_σ ← σ
```

When a message yields an ObservedDifference greater than the Threshold, the server interprets this as possible clock desynchronization and issues an adjustment notification containing the current server time plus the computed AverageObservedDifference value. This adjustment uses the average observed timestamp difference computed from recent sensor-to-server messages to determine the clock value included in the notification. When the sensor receives the notification, it updates its local clock accordingly. This process keeps the sensor closely

synchronized without requiring RTT estimation, filtering, or regression at the sensor.

This proposed mechanism provides several advantages that enhance IoT systems deployed on resource-constrained platforms. The main benefit is the elimination of periodic dedicated synchronization message exchanges, relying instead on infrequent adjustment notifications. Another benefit is that RTT estimation, filtering, and regression are not performed at the sensor, as the main synchronization processing is shifted to the server. An additional feature is the sensitivity parameter K, which is adjusted according to changes in the observed timestamp differences. Based on these characteristics, we hypothesize that the proposed server-assisted mechanism can reduce dedicated synchronization traffic without compromising synchronization accuracy or increasing sensor energy consumption.

4. MECHANISM EVALUATION

We carried out simulations with OMNeT++ [29] to evaluate the performance of our mechanism. The simulator offers a well-established framework to simulate message-oriented communication and allows fine-grained control of network conditions. We deployed the WSN in a star topology in which sensor nodes can communicate with a central server. For the MQTT-SN communication framework, we used the implementation described in [30], based on the MQTT-SN specification in [13], and added SNTP and the components required for our evaluation. We used a single sensor connected to the server to isolate the behavior of the synchronization mechanism and conduct controlled comparisons under the selected conditions. We repeated each 7000-second simulation of the proposed mechanism and SNTP ten times using different random seeds. The main simulation parameters and system components are listed in Table 1. The same simulation setup was used for all evaluated scenarios, while the network conditions were varied using the delay model described below.

We implemented a configurable delay injection process to represent different network conditions. The delay was inserted at three locations on a random basis as follows: before the sensor transmits a message, during transmission through the network, and immediately before processing at the server. An exponential distribution was used to generate the injected delays and represent variable queuing conditions.

Using different delay parameters, we modeled scenarios representing stable and congested network conditions. Each condition was characterized using the coefficient of variation (CV) as described in [25]. The formula is given in (2):

$$CV = \sigma\_RTT / \mu\_RTT \quad (2)$$

This coefficient is the ratio of the standard deviation of the RTT samples to their mean. A value less than 0.33 indicates a good network condition. Values between 0.33 and 0.66 represent fair network conditions, while values above 0.66 represent poor conditions with high delay variability [25].

Table 1: Main Simulation Configuration and MQTT-SN System Specifications

Component	Specification
Simulation Platform	OMNeT++ discrete-event simulator [29]
Network Topology	Star topology with one sensor node and one central server/gateway
Simulation Runs	Ten independent runs for each simulated scenario using different random seeds
Simulation Duration	7000 s
Client (Sensor)	MQTT-SN client node, battery-powered, with limited memory and processing resources; supports sleep and wake cycles for energy efficiency.
Server	MQTT-SN broker simulated at the gateway, acting as a bridge between the IEEE 802.15.4 network and the IP backbone.
Supported MQTT-SN Messages	CONNECT, CONNACK, REGISTER, REGACK, PUBLISH, PUBACK, SUBSCRIBE, SUBACK, PINGREQ, PINGRESP, DISCONNECT.
Timestamp Overhead	An 8-byte timestamp is added to each MQTT-SN PUBLISH payload
Physical Layer	IEEE 802.15.4 radio operating at 2.4 GHz, with a maximum data rate of 250 kbps and a maximum PHY packet size of 127 bytes
Link/Network Layer	Datagram-oriented wireless link using IEEE 802.15.4 MAC frames; suitable for low-power sensor communication
Sliding-Window Size	W = 40 observed timestamp-difference samples
Initial Sensitivity Factor	K_init = 2
Minimum Sensitivity Factor	K_min = 1
Sensitivity Adjustment Step	0.5
Variability Threshold	$\Delta\sigma$_threshold = 2 ms

During each simulation run, we measured the following three metrics: clock offset, communication overhead, and sensor energy consumption. Clock offset is a widely used metric

that reflects synchronization accuracy. It is measured at the sensor upon receiving a synchronization notification, using the standard NTP formula shown in (3) [8], [25].

$$Offset = ((T2 - T1) + (T3 - T4)) / 2 \quad (3)$$

where T1 is the transmission time at the sensor, T2 the arrival time at the server, T3 the departure time of the notification message from the server, and T4 the reception time at the sensor. In the proposed mechanism, the sensor is not required to perform this calculation during normal operations, as all the computations are shifted to the server. However, we enabled this measurement in the simulations to obtain directly comparable results with other synchronization techniques. A lower average and smaller variability of the offset indicate tighter synchronization in practice. In the presence of variable network delays, smaller average absolute offsets generally indicate better synchronization accuracy.

The second metric, communication overhead, represents the additional traffic introduced by the synchronization mechanism. Lower overhead reduces the amount of additional synchronization traffic. This metric is measured as the number of synchronization messages received by the sensor over the simulation, or as the total volume of synchronization traffic in bytes. The third metric, sensor energy consumption, is obtained from the energy model used in the OMNeT++ simulation and is reported in joules. Lower simulated energy consumption indicates that the mechanism does not introduce additional energy cost under the evaluated settings. Although this metric depends on multiple factors, including radio activity and CPU utilization, the evaluation examines whether the proposed mechanism increases it.

Regarding the algorithm parameters, we used the default initial values in all the experiments for the sensitivity factor ($K_{init} = 2$), the instability threshold ($\Delta\sigma_{threshold} = 2$ ms), and the window size ($W = 40$). The value of K_{init} defines the initial sensitivity of the algorithm, while K is adjusted during runtime according to changes in timestamp-difference variability. $\Delta\sigma_{threshold}$ represents the minimum change in timestamp-difference variability required to update K. In the preliminary runs used to configure the evaluated scenarios, short-term delay variation was approximately 1 ms. We therefore selected a slightly higher threshold value to reduce sensitivity to normal random jitter. The W value defines the number of data samples considered by the server when computing the timestamp-difference statistics. A larger W reduces decision noise but slows the initial convergence of the algorithm; a value of 40 was used as a balance between decision stability and initial convergence time in the evaluated setup.

In the rest of this section, we present the results of the two main comparisons. The first is the proposed mechanism against SNTP, and the second is the proposed mechanism against a representative MQTT-based synchronization method.

4.1 Comparison With SNTP

We implemented SNTP in our simulator as a baseline synchronization protocol for comparison with the proposed mechanism. The implemented SNTP baseline used the same periodic synchronization configuration in all evaluated scenarios. The message count reported in the results represents the synchronization messages received by the sensor during each simulation run. Comparisons were performed under the same network conditions. The injected delay parameters were varied to obtain different RTT means, standard deviations, and CV values. Three main network conditions were tested, as summarized in Table 2. Two scenarios were considered for each condition, each with a specific CV. For instance, both CV = 0.7 and CV = 0.8 have been used as representative cases for poor network conditions. For each comparison, the simulation parameters were adjusted so that SNTP and the proposed mechanism experienced approximately similar RTT means and standard deviations.

Table 2: RTT Statistics Used for the SNTP Comparison

Method	CV	Condition	Mean RTT (ms)	Std. RTT (ms)
Proposed Mechanism SNTP	0.16	Good	97.57 97.76	15.48 15.92
Proposed Mechanism SNTP	0.25	Good	83.3 83.01	20.87 21.24
Proposed Mechanism SNTP	0.4	Fair	84.16 84.5	33.8 34.8
Proposed Mechanism SNTP	0.52	Fair	86.96 86	45.51 45.9
Proposed Mechanism SNTP	0.7	Poor	100.46 97.58	70.75 71.8
Proposed Mechanism SNTP	0.8	Poor	98.22 99	79.15 79.36

4.1.1 Clock offset

Figures 1–3 summarize the results obtained under three representative network conditions selected from Table 2. In each figure, panel (a) shows the RTT variation over time, while panel (b) compares the clock offsets produced by SNTP and our mechanism. We present a visual representation

of RTT to show the differences among the network conditions, to ensure that both stable and variable delay scenarios are evaluated, and to illustrate how each solution behaves in response to delay variations.

From the plot of Figure 1(a), it can be observed that in a good network condition with CV = 0.16, RTT values are relatively stable and remain close to an average of 95 ms. The clock offset results indicate that SNTP starts with a value close to 200 ms and later stabilizes around 100.3 ms after some fluctuations, as shown in Figure 1(b). Our mechanism also begins with an offset of approximately 170 ms but later shows a smoother trajectory, with an average of 92.7 ms and fewer fluctuations in this representative run.

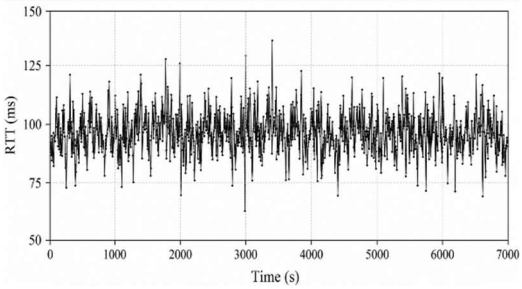

Figure 1(a): RTT Variation for CV = 0.16

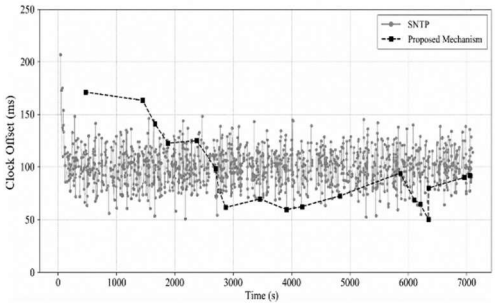

Figure 1(b): Clock Offset for CV = 0.16

In the case of a fair network (Figure 2) with CV = 0.4, the RTT plot shows mild variations in delay experienced with a few bursty spikes, especially within a time duration between 3000 s and 6000 s. During these spikes, SNTP experiences large offset deviations, as shown in Figure 2(b) at the corresponding time instants. This behavior occurs because the implemented SNTP baseline adjusts the sensor clock without filtering the abrupt delay changes observed in this scenario. In contrast, our mechanism mitigates the impact of these spikes and provides smaller offsets in this representative run. For instance, during a significant RTT increase, SNTP's offset exceeds 350 ms, while the proposed mechanism remains below 100 ms.

Figure 3 illustrates the two solutions in a worst-case scenario characterized by a CV = 0.8, with highly variable delays. The offsets of SNTP are heavily influenced by the delay spikes, causing them to climb beyond 600 ms. Our mechanism starts with a relatively large offset of about 400 ms but subsequently converges to smaller and more stable values than SNTP in this representative run. The smoother trajectory is associated with the adaptive threshold mechanism, which decreases K when the variation in the observed timestamp differences increases, making the detector more sensitive to possible desynchronization.

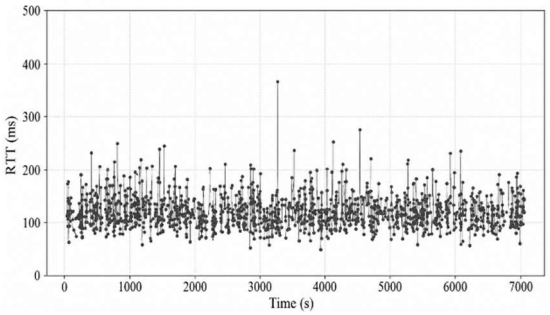

Figure 2(a): RTT Variation for CV = 0.4

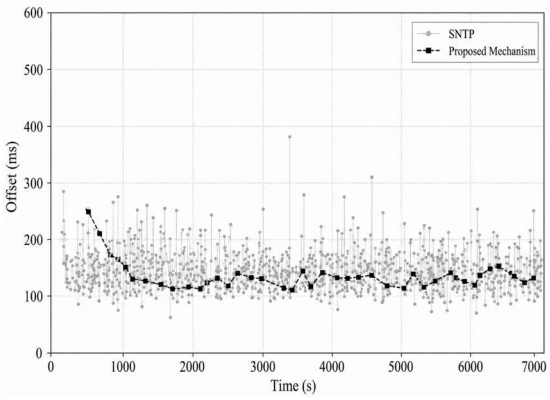

Figure 2(b): Clock Offset for CV = 0.4

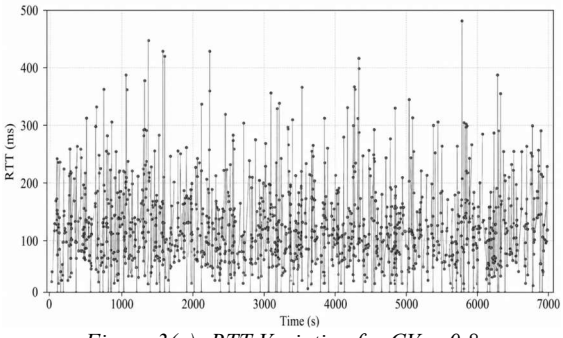

Figure 3(a): RTT Variation for CV = 0.8

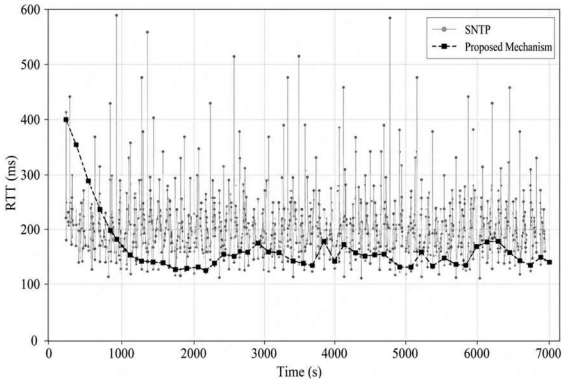

Figure 3(b): Clock Offset for CV = 0.8

The average offset across the entire simulation is a standard metric for direct comparison. Figure 4 presents these averages for the proposed mechanism and SNTP across a wider range of CV values with details listed in Table 2. Results are averaged over ten independent runs using different random seeds. The results show that both approaches provide broadly similar averages. This similarity is expected, since the initial transient phase of the proposed mechanism temporarily increases the average offset. This high offset is due to the need to collect W samples at the server side before starting the desynchronization detection. However, these full-run averages include the initial window-filling phase. As shown in Figures 1–3, the proposed mechanism provides lower or more stable offsets during the later part of the representative runs shown in these figures.

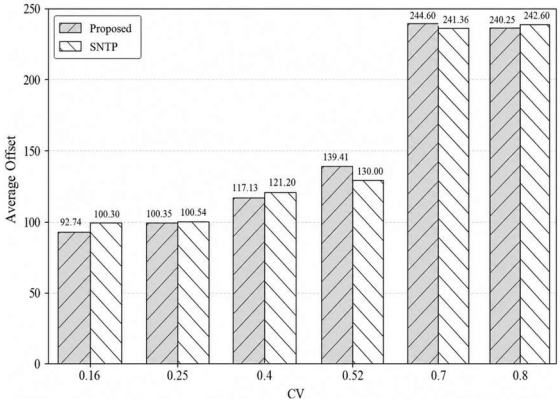

Figure 4: Average Clock Offset Across CV Values: Proposed Mechanism vs. SNTP. Bars Show Full-Run Averages; Error Bars Show 95% Confidence Intervals Across Ten Runs

4.1.2 Communication overhead and energy consumption

The communication overhead for the network conditions listed in Table 2 is shown in Figure 5. It shows that, in all evaluated scenarios, the proposed mechanism introduces considerably less overhead than SNTP. It reduces the number of synchronization messages received by the sensor from approximately 100 with SNTP to 2–4 adjustment notifications with the proposed mechanism, corresponding to a reduction of approximately 96–98%. The benefit arises because the server-side algorithm is event-driven and fires a notification only when a possible desynchronization is detected. The implemented SNTP baseline exchanges timing information periodically, including during periods when no correction is required, which increases communication overhead. By minimizing dedicated synchronization exchanges, the proposed mechanism reduces communication overhead in the evaluated MQTT-SN scenarios.

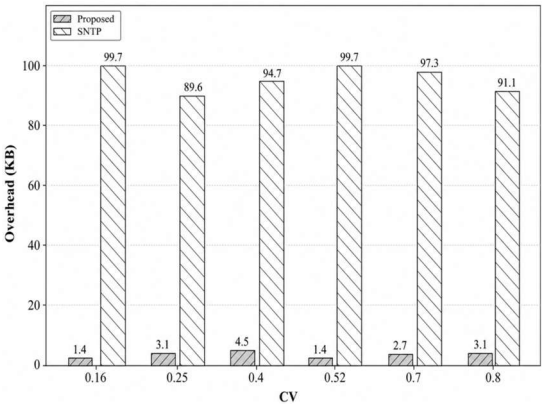

Figure 5: Number of Synchronization Messages Received by the Sensor: Proposed Mechanism vs. SNTP

Comparison of sensor energy consumption between the two approaches is illustrated in Figure 6. The proposed mechanism shows slightly lower simulated sensor energy consumption than SNTP across the evaluated scenarios. This difference is consistent with the mechanism design, which shifts synchronization detection to the server and reduces dedicated synchronization messages. This design leaves the sensors with only lightweight clock adjustments. The difference remains modest because the simulated energy metric depends on several factors beyond synchronization traffic. However, the results show that the proposed mechanism reduces communication overhead without increasing simulated sensor energy consumption under the evaluated conditions.

4.2 Comparison With SMQTT

To further evaluate the proposed mechanism, we compared it with SMQTT [25], a representative MQTT-based synchronization approach that, like

the proposed mechanism, relies on application messages carrying timestamp information. As the SMQTT source code was not publicly available, we reproduced similar network conditions by adjusting the delay model until the mean RTT, standard deviation of RTT, and resulting CV were close to the values reported in [25]. Although the exact packet-level behavior of SMQTT could not be replicated, matching the reported RTT statistics provides a basis for an indirect comparison under approximately similar delay conditions. Table 3 summarizes these conditions. The RTT and average offset values for SMQTT were taken from the results reported in [25], while the proposed-mechanism results were obtained from our OMNeT++ simulations.

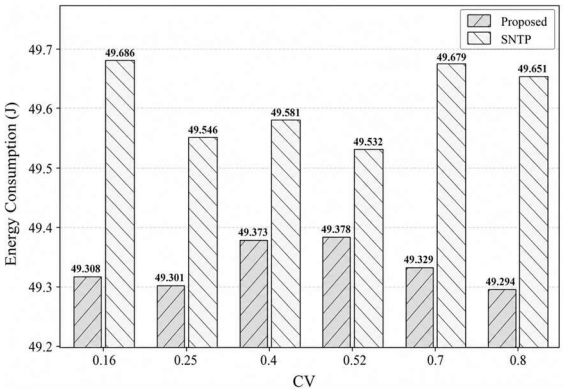

Figure 6: Simulated Sensor Energy Consumption: Proposed Mechanism vs. SNTP

Table 3: Approximately Matched Network Conditions and Average Clock Offsets for the Proposed Mechanism and SMQTT. SMQTT Values Are Reported Means From [25]

Method	CV	Condition	Mean RTT (ms)	Std. RTT (ms)
Proposed Mechanism	0.32	Good	83.7	27.1
SMQTT	0.32		83.23	26.85
Proposed Mechanism	0.49	Fair	95.07	47
SMQTT	0.49		95.19	47.29
Proposed Mechanism	0.91	Poor	93.69	85.27
SMQTT	0.94		93.66	88.52

Figure 7 depicts the average offsets of the proposed mechanism and the reported SMQTT results. Under the three approximately matched network conditions, the proposed mechanism produced lower average offsets than the reported SMQTT values. For example, in the poor network condition, the average offset decreased from 409.3 ms for SMQTT at CV = 0.94 to 312.8 ms for the proposed mechanism at CV = 0.91. The proposed mechanism showed average offsets that were 49% lower in the good condition, 31% lower in the fair condition, and 23% lower in the poor condition than the reported SMQTT values. The proposed-mechanism values were averaged over repeated simulation runs and were lower than the SMQTT means reported in [25] under the approximately matched conditions.

SMQTT does not report results for communication overhead or energy consumption. We therefore report the communication overhead and simulated sensor energy consumption of the proposed mechanism under the approximately matched network conditions, without presenting them as direct comparisons with SMQTT. Figure 8 shows that the synchronization traffic received by the sensor remains below 4 KB in all evaluated conditions. In the poor condition, the sensor received 50 adjustment notifications, corresponding to 3.66 KB of synchronization traffic. Since [25] does not report communication overhead using the same metric, a direct overhead comparison with SMQTT cannot be made.

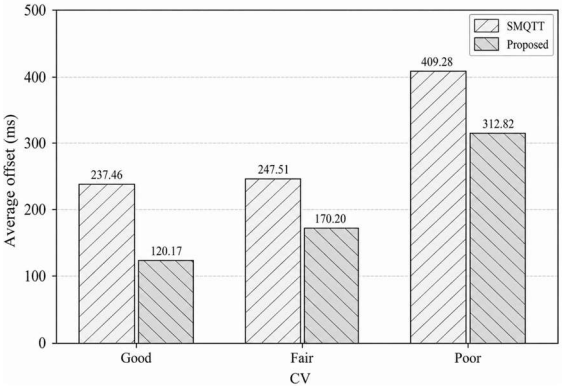

Figure 7: Average Clock Offset Under Approximately Matched Network Conditions. Error Bars Show 95% Confidence Intervals Across Ten Runs for the Proposed Mechanism; SMQTT Values Are Reported Means From [25]

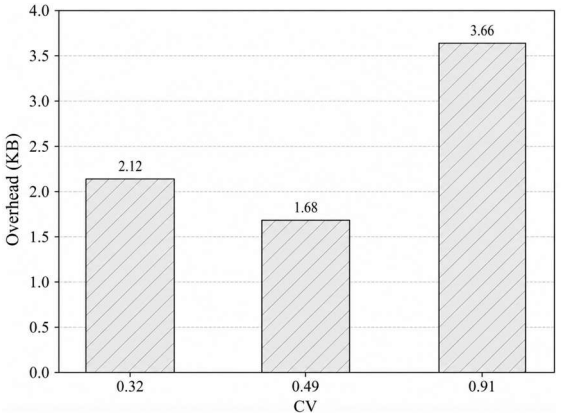

Figure 8: Communication Overhead of the Proposed Mechanism Under Approximately Matched SMQTT Network Conditions

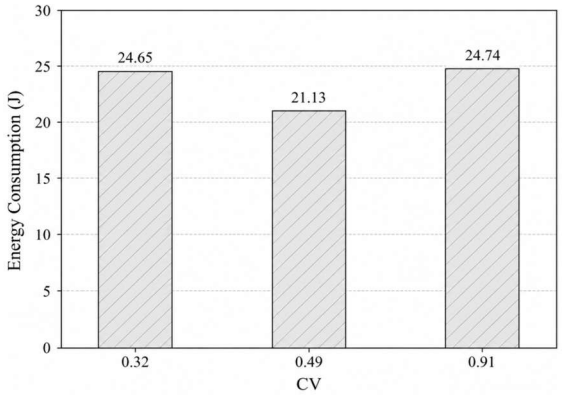

Figure 9: Simulated Sensor Energy Consumption of the Proposed Mechanism Under Approximately Matched SMQTT Network Conditions

4.3 Discussion

The simulation results show that the proposed mechanism provides broadly similar full-run clock-offset averages to SNTP and lower average offsets than the reported SMQTT results under approximately matched conditions. It also considerably reduces communication overhead relative to the implemented SNTP baseline without increasing simulated sensor energy consumption. Overall, these findings address the study objective by showing comparable synchronization accuracy with substantially lower dedicated synchronization traffic. These findings also support the study hypothesis under the evaluated conditions.

The adaptive threshold with parameter K supports the detection of possible desynchronization from the observed timestamp differences, without requiring per-sensor RTT estimation or regression. Because detection is performed on the server and notifications are event-driven, sensor-side processing remains limited. This avoids the sensor-side RTT calculation used in SMQTT [25]. The use of existing MQTT-SN traffic reduces the need for the periodic or scheduled synchronization-specific exchanges used by recent WSN approaches [19]–[21]. Results under the evaluated poor conditions (CV > 0.9) show that the mechanism continues to operate under high delay variability.

The evaluation was limited to a single-sensor simulation. Multi-sensor operation and gateway scalability were not evaluated. The comparison with SMQTT is also indirect because the SMQTT source code was not available. It therefore relies on published results under approximately matched delay conditions.

5. CONCLUSION

This paper addressed the time synchronization problem in resource-constrained IoT systems that deliver application traffic over MQTT-SN. The main contribution of this work is the combination of existing MQTT-SN traffic for timing information, server-side adaptive desynchronization detection, and event-driven clock adjustment. This design reduces the need for dedicated synchronization exchanges and sensor-side synchronization processing. The sensor's role is limited to timestamping data and applying clock adjustments when needed. Simulation results showed broadly similar full-run clock-offset averages to SNTP. The proposed mechanism reduced synchronization messages by approximately 96–98% without increasing simulated sensor energy consumption relative to SNTP under the evaluated conditions. It also produced lower average offsets than the reported SMQTT values under approximately matched conditions. The current evaluation is limited to a single-sensor simulation and an indirect comparison with SMQTT. Future work includes real-world multi-sensor deployments, scalability studies on gateway hardware, and adaptation of the mechanism to other application protocols such as CoAP.

REFERENCES

[1] L. Atzori, A. Iera, and G. Morabito, "The Internet of Things: A survey," Computer Networks, Vol. 54, No. 15, Oct. 2010, pp. 2787–2805, doi: 10.1016/j.comnet.2010.05.010.

[2] J. Gubbi, R. Buyya, S. Marusic, and M. Palaniswami, "Internet of Things (IoT): A vision, architectural elements, and future directions," Future Generation Computer Systems, Vol. 29, No. 7, Sep. 2013, pp. 1645–1660, doi: 10.1016/j.future.2013.01.010.

[3] D. Assylbek, A. Nadirkhanova, and D. Zorbas, "Energy-efficient Clock-Synchronization in IoT Using Reinforcement Learning," Proceedings of the 2024 20th International Conference on Distributed Computing in Smart Systems and the Internet of Things (DCOSS-IoT), Abu Dhabi, United Arab Emirates, 2024, pp. 244–248, doi: 10.1109/DCOSS-IoT61029.2024.00044.

[4] M. Xu, W. Xu, T. Han, and Z. Lin, "Energy-efficient time synchronization in wireless sensor networks via temperature-aware compensation," ACM Transactions on Sensor Networks, Vol. 12, No. 2, Apr. 2016, Article No. 12, doi: 10.1145/2876508.

[5] M. K. Maggs, S. G. O'Keefe, and D. V. Thiel, "Consensus Clock Synchronization for Wireless Sensor Networks," IEEE Sensors Journal, Vol. 12, No. 6, Jun. 2012, pp. 2269–2277, doi:

10.1109/JSEN.2011.2182045.
[6] J. M. Castillo-Secilla, J. M. Palomares, and J. Olivares, "Temperature aware methodology for time synchronization protocols in wireless sensor networks," Electronics Letters, Vol. 49, No. 7, Apr. 2013, pp. 506–508, doi: 10.1049/el.2012.3911.
[7] I.-K. Rhee, J. Lee, J. Kim, E. Serpedin, and Y.-C. Wu, "Clock synchronization in wireless sensor networks: An overview," Sensors, Vol. 9, No. 1, Jan. 2009, pp. 56–85, doi: 10.3390/s90100056.
[8] D. L. Mills, J. Martin, J. Burbank, and W. Kasch, "Network Time Protocol Version 4: Protocol and Algorithms Specification," RFC 5905, Jun. 2010.
[9] D. L. Mills, "Simple Network Time Protocol (SNTP) Version 4 for IPv4, IPv6 and OSI," RFC 4330, Jan. 2006.
[10] S. Ganeriwal, R. Kumar, and M. B. Srivastava, "Timing-sync protocol for sensor networks," Proceedings of the 1st International Conference on Embedded Networked Sensor Systems (SenSys), Los Angeles, CA, USA, Nov. 2003, pp. 138–149.
[11] M. Maróti, B. Kusy, G. Simon, and Á. Lédeczi, "The flooding time synchronization protocol," Proceedings of the 2nd International Conference on Embedded Networked Sensor Systems (SenSys '04), Baltimore, MD, USA, Nov. 2004, pp. 39–49.
[12] A. Banks and R. Gupta, "MQTT Version 3.1.1," OASIS Standard, Oct. 2014.
[13] A. Stanford-Clark and H. L. Truong, "MQTT For Sensor Networks (MQTT-SN) Protocol Specification Version 1.2," Nov. 14, 2013.
[14] Z. Shelby, K. Hartke, and C. Bormann, "The Constrained Application Protocol (CoAP)," RFC 7252, Jun. 2014.
[15] Y. Weng and Y. Zhang, "A survey of secure time synchronization," Applied Sciences, Vol. 13, No. 6, Mar. 2023, Article No. 3923, doi: 10.3390/app13063923.
[16] H. Yiğitler, B. Badihi, and R. Jäntti, "Overview of time synchronization for IoT deployments: Clock discipline algorithms and protocols," Sensors, Vol. 20, No. 20, Oct. 2020, Article No. 5928, doi: 10.3390/s20205928.
[17] L.-A. Phan, T. Kim, T. Kim, J. S. Lee, and J.-H. Ham, "Performance Analysis of Time Synchronization Protocols in Wireless Sensor Networks," Sensors, Vol. 19, No. 13, Jul. 2019, Article No. 3020, doi: 10.3390/s19133020.
[18] J. Elson, L. Girod, and D. Estrin, "Fine-grained network time synchronization using reference broadcasts," Proceedings of the 5th Symposium on Operating Systems Design and Implementation (OSDI), Dec. 2002, pp. 147–163, doi: 10.1145/844128.844143.
[19] Y. Zong, X. Dai, Z. Wei, M. Zou, W. Guo, and Z. Gao, "Robust Time Synchronisation for Industrial Internet of Things by H∞ Output Feedback Control," IEEE Internet of Things Journal, Vol. 10, No. 3, Feb. 2023, pp. 2021–2030, doi: 10.1109/JIOT.2022.3144199.
[20] Z. Jia, D. Cui, X. Dai, Z.-W. Liu, S. Liu, and T. Chai, "Low Overhead Minimum Variance Time Synchronization for Time-Sensitive Wireless Sensor Networks," IEEE Transactions on Automation Science and Engineering, Vol. 22, Feb. 2025, pp. 5349–5359, doi: 10.1109/TASE.2024.3419142.
[21] S. Liu, X. Dai, Y. Zong, Z. Jia, D. Zhou, and D. Cui, "Low-Overhead Time Synchronization for Energy-Constraint Industrial Internet of Things," Proceedings of the 2023 6th International Symposium on Autonomous Systems (ISAS), Nanjing, China, Jun. 23–25, 2023, pp. 1–6, doi: 10.1109/ISAS59543.2023.10164502.
[22] M. R. Palattella, N. Accettura, X. Vilajosana, T. Watteyne, L. A. Grieco, G. Boggia, and M. Dohler, "Standardized Protocol Stack for the Internet of (Important) Things," IEEE Communications Surveys & Tutorials, Vol. 15, No. 3, 2013, pp. 1389–1406, doi: 10.1109/SURV.2012.111412.00158.
[23] F. Palmese, E. Longo, A. E. C. Redondi, and M. Cesana, "CoAP vs. MQTT-SN: Comparison and performance evaluation in publish-subscribe environments," Proceedings of the 2021 IEEE 7th World Forum on Internet of Things (WF-IoT), New Orleans, LA, USA, 2021, pp. 153–158, doi: 10.1109/WF-IoT51360.2021.9595725.
[24] A. Shaout and B. Crispin, "Using the MQTT Protocol in Real Time for Synchronizing IoT Device State," International Arab Journal of Information Technology, Vol. 15, No. 3A, Special Issue 2018, pp. 515–521.
[25] S. B. Chauhan and R. N. Gore, "SMQTT: A Lightweight Clock Synchronization Algorithm for IoT Devices Using MQTT," Proceedings of the 2023 IEEE International Symposium on Precision Clock Synchronization for Measurement, Control, and Communication (ISPCS), London, United Kingdom, 2023, pp. 1–6, doi: 10.1109/ISPCS59528.2023.10296995.
[26] V. D. Khatade and A. A. Askhedkar, "Time

Synchronization for CoAP using NS2," Proceedings of the 2019 5th International Conference on Computing, Communication, Control and Automation (ICCUBEA), Pune, India, 2019, pp. 1–4, doi: 10.1109/ICCUBEA47591.2019.9129316.

[27] M. A. Hossain, A. M. Rahman, and M. Atiquzzaman, "COPA-Syn: A CoAP-based time synchronization protocol for IoT applications," IEEE Internet of Things Journal, Vol. 8, No. 12, Jun. 2021, pp. 9753–9763, doi: 10.1109/JIOT.2021.3066491.

[28] S. C. Son, N. W. Kim, B. T. Lee, C. H. Cho, and J. W. Chong, "A time synchronization technique for CoAP-based home automation systems," IEEE Transactions on Consumer Electronics, Vol. 62, No. 1, Feb. 2016, pp. 10–16, doi: 10.1109/TCE.2016.7448557.

[29] OMNeT++, "OMNeT++ Discrete Event Simulator," [Online]. Available: https://omnetpp.org/. Accessed: Oct. 1, 2025.

[30] T. Aydoğan Gümüş, "MQTT-SN Implementation for OMNeT++," GitHub repository, 2019. [Online]. Available: https://github.com/aydogangumus/mqtt-sn. Accessed: Oct. 1, 2025.