SRST: A Secure and Resilient Synchronization of Time for WSNs in IoT Applications

Amin Saiah[a], Chafika Benzaid[b], Mohamed Younis[c], Nadjib Badache[d]

[a] *University of Hassiba Ben-bouali Chlef, Algérie.*
Email: a.saiah@univ-chlef.dz
[b] *University of OULU, OULU, Finland.*
Email: chafika.benzaid@oulu.fi
[c] *University of Maryland Baltimore County, USA.*
Email: younis@umbc.edu
[d] *University of Sciences and Technology Houari Boumediene, Algérie.*
Email: badache@mail.cerist.dz

Abstract

Many applications of Wireless Sensor Networks (WSNs) in internet of things require accurate time synchronization for the successful realization. In a hostile environment, protecting time synchronization against faulty timestamps injection attacks is paramount. Malicious nodes (MNs) could broadcast faulty timestamps to decrease the accuracy of the whole network. The Receiver-Only (RO)-based synchronization methodology achieves high accuracy while reducing the number of timing messages compared to other methodologies. However, RO is vulnerable to node failure and supports only a single reference, which makes it inaccurate in the presence of MNs. To address these limitations, we propose SRST, a secure and resilient synchronization of time for WSNs. SRST employs a novel time synchronization model that extends RO to allow synchronizing sensor nodes to multiple mutually-synchronized references, which enhances robustness against node failure and malicious behavior. SRST optimizes convergence time by synchronizing the RO node with its synchronized 1-hop and 2-hop neighbors. RO node applies a new delay threshold-based detection technique to identify reference nodes as MNs through detection of faulty timestamps. SRST is topology-independent and can be applied to multi-tier, cluster-based and flat topologies. We validated SRST through simulations and prototype experiments, comparing its performance with several state-of-the-art protocols. The results demonstrate that SRST outperforms existing protocols in accuracy, achieving synchronization within less than 1 μs, and accelerates convergence time by a factor of 31.25 compared to the best-known protocol. SRST has also been shown to be more effective in mitigating faulty timestamp injection attacks, successfully detecting errors of 6 μs with a 100 % success rate and ensuring that the clocks of any two nodes do not deviate by more than 6 μs. Furthermore, the results indicate that SRST imposes little communication overhead.

Keywords: Wireless sensor networks, Internet of things, Time synchronization, Receiver-only synchronization, Faulty timestamps, Security.

1. Introduction

A Wireless Sensor Networks (WSNs) is a network of sensor nodes that can communicate wirelessly with each other and with the sink (or base station) over multiple hops, as illustrated in Fig. 1. Sensor nodes are small devices resource-constrained equipped with sensors to capture and collect data from physical or environmental conditions. Recent years have witnessed a growing interest in Internet of Things (IoT) applications, where a diverse set of devices are internetworked to perform sensing, processing and/or actuation tasks [1], [2], [3], [4]. Examples of these applications include remote healthcare monitoring, structural health monitoring, automated manufacturing, intelligent grid, smart transportation systems, precision agriculture and home/building control. In an IoT environment, the gateway (or IoT device) processes the data collected from the sensor nodes and send it to distant command centers (remote user) or Cloud/Fog servers for advanced analytics, storage, and decision-making. WSNs are a crucial part of IoT applications, providing the necessary infrastructure for real-time data collection, communication, and analysis. It allows data to be collected and transferred seamlessly from the real world to digital systems, facilitating real-time control and monitoring. Hence, precise synchronized time among sensor nodes is an essential requirement for ensuring accurate data correlation, power management, event detection and reporting, efficient communication, coordinated actions and real-time operations. Many IoT systems require a global time to operate correctly. For instance, in a smart power grid, all synchrophasor units must be synchronized in order to

assess the stability of the grid, accurately detect failures and overload, and avoid unwarranted adjustment in the power flow. In healthcare applications such as remote patient monitoring and body-mounted sensors, synchronized time guarantees accurate data collection, helping medical professionals to monitor health status, identify early warning signs, and provide timely interventions. In industrial automation systems, where sensors and actuators work together, imprecise timing can result in wrong action. The same generally applies for all cyber-physical systems involving distributed control. Overall, time synchronization is very crucial in many WSN-based IoT applications.

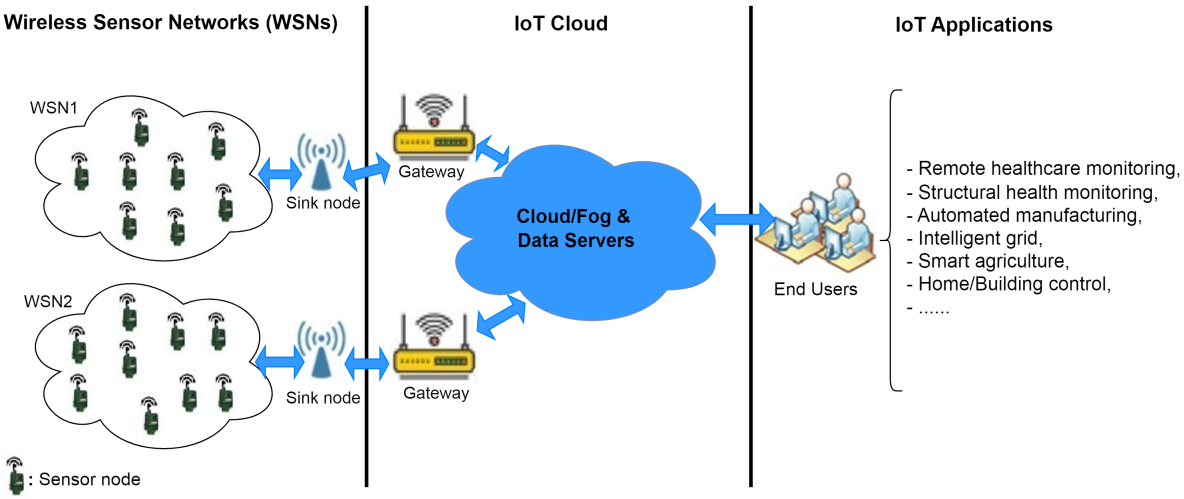


Figure 1: Architecture of WSNs for IoT applications.

Many centralized and distributed time synchronization approaches have been proposed for WSNs. However, the presence of malicious nodes (MNs) poses a major challenge. MNs can broadcast faulty time synchronization messages to disrupt network operation. Non-malicious nodes may use these faulty timestamps to synchronize their clocks, inadvertently propagating incorrect time information, leading to degraded time precision across the network. Centralized time synchronization can achieve fast convergence, but it depends on specific nodes (such as leaders) and hierarchical structure and, more importantly, can be disrupted if the reference node fails. Distributed time synchronization protocols avoid these shortcomings, yet they converge slowly as the communication overhead is quite high.

The popular basic methodologies used in time synchronization are Receiver-Receiver (RR) [5], [6], Sender-Receiver (SR) [7], [8] and Receiver-Only (RO) [9], [10]. Nonetheless, the RO methodology stands out highly efficient option for WSNs, achieving accurate time synchronization with significantly fewer messages. This reduces energy consumption, a critical factor for resource-constrained sensor nodes. However, synchronizing RO nodes to a single reference makes the synchronization process vulnerable to attacks launched by MNs, where a reference node could, in fact, be an MN sending faulty timestamps to RO nodes. Being unaware of the attack, RO nodes could use the faulty timestamps to synchronize their clocks and propagate the wrong time information to other sensor nodes. Hence, a malicious reference node could potentially disrupt the time synchronization of a large part of the network.

Designing a secure time synchronization protocol that achieves high-precision, fast convergence, robustness to node failure, and low communication overhead is a significant challenge. To address these design goals, we propose SRST, a secure and resilient synchronization of time protocol, which retains the advantages of both centralized and distributed synchronization while mitigating their limitations. To overcome the vulnerability of the RO methodology, SRST enables the synchronization of RO nodes with multiple mutually-synchronized references, which enhances robustness against node failure and malicious behavior. Unlike distributed approaches, which average clocks across all neighbors, the RO node in SRST synchronizes its clock with its synchronized 1-hop and 2-hop neighbors. This optimization accelerates convergence time, enabling SRST to achieve higher accuracy. Additionally, SRST does not rely on specific nodes; consequently, no additional overhead is required to select leader nodes for global time synchronization. This addresses the restrictive conditions in existing RO-based approaches, requiring RO nodes to lie within the common coverage region of two leaders. In summary, the paper makes the following contributions:

1. **Novel time synchronization model**: SRST introduces a novel time synchronization model that extends the RO methodology, enabling synchronization with multiple reference nodes. We derive Maximum Likelihood (ML) estimators for clock offset and clock skew and compute the Cramer-Rao Lower Bound (CRLB) to evaluate estimator performance, providing a lower bound for variance in unbiased estimators.

2. **Faulty timestamps detection**: SRST employs a delay threshold-based detection mechanism to mitigate faulty timestamps. RO node computes delay relative to reference node and determines if the timestamp is faulty or not according to a delay value. By receiving several timestamps from reference nodes, SRST can tolerate faulty ones.

3. **Validation and performance**: We validate SRST through simulation and experiments with real devices. The results show that SRST outperforms existing schemes in accuracy, communication overhead, and convergence speed. Furthermore, SRST is shown to be resilient to faulty timestamps and ensures that the attacker cannot cause a clock drift more than 6 μs without being detected.

The structure of the paper is as follows: Section 2 discusses the related research and points out gaps in the literature. Section 3 provides an overview of the RO methodology and the assumed system model. Section 4 describes our clock synchronization model in detail. In Section 5, we propose a delay threshold-based detection technique for secure time synchronization against faulty timestamps. Section 6 presents SRST protocol. Section 7 reports the performance of SRST. In Section 8, we summarize and discuss the main findings. Finally, Section 9 concludes the paper.

2. Related Work

In this section, we cover the related work in the literature and highlight the technical gap.

2.1. Time Synchronization Protocols

Time synchronization relies on estimating and adjusting two properties of a clock, namely offset and skew. *Offset* denotes the difference in time between two clocks. *Skew* refers to variations in the frequencies of two clocks, influenced by factors such as environmental changes. Many centralized and distributed time synchronization approaches have been proposed. Receiver-Receiver (RR), Sender-Receiver (SR), or Receiver-Only (RO) are the three basic methodologies used in time synchronization protocols. In the RR methodology [5], [6], receivers that receive the same beacon exchange their reception times to synchronize their clocks. The Reference Broadcast Synchronization (RBS) protocol [5] is an RR-based time synchronization. RBS aims to reduce or eliminate the synchronization errors resulting from send time and access time. However, the receivers expend substantial energy exchanging synchronization messages. Djenouri *et al.* [6] propose distributing the role of the beacon sender among all receivers, thereby addressing the failure issue inherent in RBS. Additionally, the reception times are included within the beacons, eliminating the need to exchange reception time between receivers. The shortcoming of this protocol is that the timing message contains a lot of timestamps.

In the SR methodology [7], [8], a receiver node synchronizes its clock using the timestamps from two-way messages exchanged with a sender node. The Timing-Sync Protocol for Sensor Networks (TPSN) [7] establishes initially a tree topology and then children nodes synchronize their clocks with parent node by performing SR methodology. TPSN employs timestamping at MAC-layer to limit sender-receiver sides non-determinism in message transmission. However, it ignores the clock skew. Moreover, if the parent node fails, then a procedure to determine a new parent is necessary, thus imposing significant cost and potentially long periods until resynchronization. In [8], the Flooding Time Synchronization Protocol (FTSP) is proposed. Initially, one node in the network is designated as the root of reference clock, in which all the network nodes will be synchronized with it. In this protocol, a receivers can be synchronized directly with sender by one broadcast message timestamped at MAC-layer. FTSP is very accurate than RBS and TPSN. However, a new procedure of root election is necessary when the root fails, thus adding more communication overhead.

In the RO methodology [9], [10], a set of RO nodes synchronize their clocks by receiving timing messages interchanged between two leader nodes. Such an RO methodology proves to be very effective since it can achieve accurate synchronized time with much fewer synchronization messages, and hence high energy efficiency than RR and SR based protocols [9]. In [9], the Pairwise Broadcast Synchronization (PBS) scheme is proposed to achieve synchronization with reduced energy consumption by minimizing the number of synchronization messages. Noh *et al.* [11] extend the PBS for synchronizing multicluster, where the entire network is organized into clusters by an algorithm which incurs energy consumption. PBS requires that the RO nodes are within the shared coverage area of leader nodes. The Synchronization through Piggybacked Reference (SPiRT) protocol [10] extend the RO methodology to reduce the number of pairwise messages by increasing the number of RO nodes. Liu *et al.* [12] propose a generalized RO-based synchronization model that enhances accuracy through a new method for estimating clock skew and offset. Existing RO-based schemes support only a single reference and rely on restrictive conditions,

namely: (i) the dependence on leaders, which makes the synchronization protocol vulnerable to their failure; and (ii) the RO nodes should lie within the common coverage region of two leaders.

Therefore, centralized time synchronization can achieve fast convergence, but it depends on specific nodes (such as leaders) and hierarchical topology and, more importantly, can be disrupted if the reference node fails. Distributed time synchronization is proposed to support node failure and does not require a specific node or hierarchical structure. Many distributed protocols, e.g., [13], [14], [15], [16], [17], [18], [19], [20], [21], [22], have been proposed, but they suffer from increased communication overhead, a slow convergence rate, and can not synchronize the network to a global time reference.

2.2. Security Protocols for Time Synchronization

The aforementioned time synchronization protocols cannot be applied correctly in hostile environments, where malicious nodes (MNs) can exist. Recently, many security approaches [23] for time synchronization have been proposed. The published methodologies for secure time synchronization can be summarized as they thwart attacks by using cryptographic mechanisms, preset threshold and/or outliers detection. In *Threshold-based detection*, Ganeriwal *et al.* [24] propose a secure single-hop (SPS) for SR methodology, enabling receiver nodes to detect delay attacks using a pre-defined delay threshold. SPBS [25], [26] also proposes a delay threshold to provide a secure mechanism for the RO methodology. SPBS avoids delaying timing messages by employing propagation delay thresholds d^*_{min} and d^*_{max}. In [27], the authors introduce a threshold filter algorithm designed to enhance the security of single-hop device-to-device and multi-hop time synchronization in industrial IoT environments. Qiu *et al.* [28] propose an offset threshold to defend against fake timestamps sent by MNs. In spanning tree topology, a node detects MN using a maximum time offset between its parent and grandparent. Yet, a sequence of multi-hop MNs cannot be detected. Jha *et al.* [29] analyze the security threat for consensus-based time synchronization algorithms and propose a generic algorithm which enables resilience to message manipulation attacks using an offset threshold.

In *Detecting outliers*, Sun *et al.* [25] propose a TinySeRSync protocol for secure SR methodology. In TinySeRSync the receiver can tolerate n malicious nodes if it selects $2n+1$ clock adjustments from different neighbor nodes. It uses a median to filter out outliers. Song *et al.* [30] have proposed two outlier-based detection techniques to identify the faulty clock offsets caused by delay attacks after collecting multiple clock offsets from neighbor nodes. However, both techniques rely on having a sufficiently large number of clock offsets available to filter out outliers. Jia *et al.* [31] employ a threshold to compare the relative clock skew between cluster heads and cluster members, discarding faulty timestamps produced by MNs. Phan *et al.* [32] present a novel method to enhance the security of distributed time synchronization by incorporating neighbor awareness. Each node can exclude the inappropriate neighbor for synchronization based on the status such as synchronized, new, or unsynchronized neighbor. Jeon *et al.* [33] employ the median of timestamps gathered from neighbors to exclude outliers when calculating the consensus value. The focus of Suresh *et al.* [34] is on detecting a Sybil attack on consensus-based time synchronization algorithms. A maximum relative clock skew is employed to detect and filter Sybil messages. However, neglecting the delay of the one-hop timing message could impact accuracy performance. Wang *et al.* [35] propose a novel mechanism to filter the faulty timestamps produced by Sybil attacks. It based on the verification of packet sequence numbers, sending timestamps, and receiving timestamps.

In *Cryptographic mechanisms*, Ganeriwal *et al.* [24] propose using an authentication tag to ensure the integrity of timing messages. Sun *et al.* [25] use μTESLA [36] to authenticate timing messages and symmetric cryptography for secure message broadcasting; μTESLA itself requires prior time synchronization. The authors in [37] and [38] propose to use blockchain-based technique against faulty timestamps sent by MNs. The Secure Pairwise Broadcast Time Synchronization (SPBS) protocol [39], [26] also uses public key-based authentication to ensure the timing messages integrity. However, public key cryptography is computationally heavy and would thus impose delays to the clock adjustment which decreases the accuracy of time synchronization.

2.3. Research Gap and Proposed Solution

Table 1 highlights the gap in the existing time synchronization research. Centralized time synchronization can achieve fast convergence, but it depends on specific nodes, imposes hierarchical structure and, more importantly, is susceptible to disruption if the reference node fails. Distributed time synchronization protocols avoid these shortcomings and are becoming popular nowadays; they converge slowly as the communication overhead is quite high. Recently published protocols, e.g., [16], [17], [18], [19], [20], [21], [22], have focused on reducing the convergence time of distributed time synchronization. However, they still need several rounds of synchronization to converge, especially for large-scale networks. Table 2, on the other hand, summarizes the shortcomings of existing security primitives for time synchronization. Cryptographic mechanisms delay authentication to ensure message integrity and authenticity which negatively affects the accuracy of time synchronization [26]. Employing

outlier-based detection requires a sufficiently large number of timestamps to filter out outliers. For threshold-based detection techniques, the attacker can introduce some errors to the synchronization process without being detected. Therefore, designing a secure time synchronization approach with high-precision, fast convergence, robustness to node failure, and low communication overhead is really a great challenge. Our proposed SRST protocol overcomes the aforementioned shortcomings. SRST is the first approach that keeps the advantages and mitigates the limitations of both centralized and distributed time synchronization. SRST extends the RO methodology to allow synchronizing RO nodes to multiple mutually-synchronized references, and thus it is robust to node failure and converges fast. Regarding the restrictive conditions in existing RO-based approaches, SRST does not depend on specific nodes, and consequently, no extra overhead is needed to select leader nodes (such as in [11] and [10]) in global time synchronization. Furthermore, SRST is topology-independent and can be applied to multi-tier, cluster-based, and flat topologies. SRST uses a new delay threshold-based detection technique to flag malicious nodes that inject faulty timestamps. As demonstrated in the remainder of the paper, an attacker cannot inject more than 6 μs, which is quite negligible in most WSN applications. In the following sections, we elaborate on the specifics of our approach.

Time sync. protocols	**Convergence speed**	**Leaders dependence**	**Robustness to node failure**	**topology dependence**
Centralized, e.g., [5], [6], [7], [8], [10], [9], [12]	Fast	Yes	No	Yes
Distributed, e.g., [13], [14], [15], [16], [17], [18], [19], [20], [21], [22]	Slow	No	Yes	No
SRST	Fast	No	Yes	No

Table 1: Classification and gap analysis of time synchronization protocols.

Security mechanisms	**Limitations**
Threshold-based detection, e.g., [26], [24], [27], [28], [29]	- The attacker can introduce some errors to the synchronization process without being detected.
Detecting outliers, e.g., [25], [30], [31], [32], [33], [34], [35]	- Assume that sufficient amount of timestamps is available to filter out faulty timestamps.
Cryptographic mechanisms, e.g., [24], [25], [37], [38]	- More storage, computing, and communication overhead. - Decrease the accuracy of synchronization by delaying authentication.

Table 2: Categorization and comparison of security mechanisms for time synchronization.

3. Background and System Model

This section covers important background and states noteworthy assumptions. For quick reference, Table 3 contains a summary of the important notations used throughout this paper.

Notation	Description
RO	Receiver-only node
H_j	jth synchronized neighbor of node RO
$F_{(j,k)}$	kth synchronized neighbor of node H_j
$C(t)$	Local clock at time t
δ	Clock offset
ω	Clock skew
σ	Variance
$d^{(H_j,F_{(j,k)})}$	Delay between nodes H_j and $F_{(j,k)}$
$d^{(H_j,RO)}$	Delay between nodes H_j and RO
$D^{(RO,F_{(j,k)})}$	$(d^{(H_j,F_{(j,k)})} - d^{(H_j,RO)})$
M_1	Synchronization message
M_2	Acknowledgment message
M_3	Forwarding message
$s_j^{(1)}$	Transmission time of M_1 at node H_j
s_j	Reception time of M_1 at node RO
$s_{j,k}$	Reception time of M_1 at node $F_{(j,k)}$
$s_{j,k}^{(1)}$	Transmission time of M_2 at node $F_{(j,k)}$
$s_{j,k}^{(2)}$	Reception time of M_2 at node H_j
$s_j^{(2)}$	Transmission time of M_3 at node H_j
$s_j^{(3)}$	Reception time of M_3 at node RO
S	Number of received timestamps $s_{j,k}$ from nodes $F_{(j,k)}$
DS	Number of $D^{(RO,F_{(j,k)})}$
$\delta^{(RO,F_{(j,k)})}$	Offset between clocks of nodes RO and $F_{(j,k)}$
$\omega^{(RO,F_{(j,k)})}$	Skew between clocks of nodes RO and $F_{(j,k)}$
f	Likelihood function
Δ	Error injected by the attacker
$s_{j,k}^*$	Faulty timestamps $s_{j,k}^* = s_{j,k} \pm \Delta$
$d^{(H_j,F_{(j,k)})}$	Delay between nodes H_j and $F_{(j,k)}$
$d^{(H_j,F_{(j,k)})^{*max}}$	upper-bound for $d^{(H_j,F_{(j,k)})}$
$d^{(H_j,F_{(j,k)})^{*min}}$	lower-bound for $d^{(H_j,F_{(j,k)})}$
$d^{(H_j,RO)^{*max}}$	Delay upper-bound for $d^{(H_j,RO)}$
$d^{(H_j,RO)^{*min}}$	Delay lower-bound for $d^{(H_j,RO)}$

Table 3: Summary of the used notations.

3.1. Overview of RO Methodology

According to the RO methodology, a node (e.g., RO in Fig. 2) can synchronize its clock by receiving the two-way timing messages exchanged between a two leader nodes, e.g., nodes H and F in Fig. 2. Thus, the RO methodology exchanges fewer timing messages, thereby consuming less energy compared to the SR and RR methodologies [9].

The basic idea (see Fig. 3) is that while node H employs the SR methodology with node F, neighbors can synchronize with node F by receiving timing messages exchanged between F and H. Node H synchronizes as follows: it broadcasts a synchronization message M_1 at its local clock $s_1^{(H)}$ and node F receives M_1 at its local clock $s_2^{(F)}$. Then, node F returns an acknowledgment message M_2 at its local clock $s_3^{(F)}$. The acknowledgment

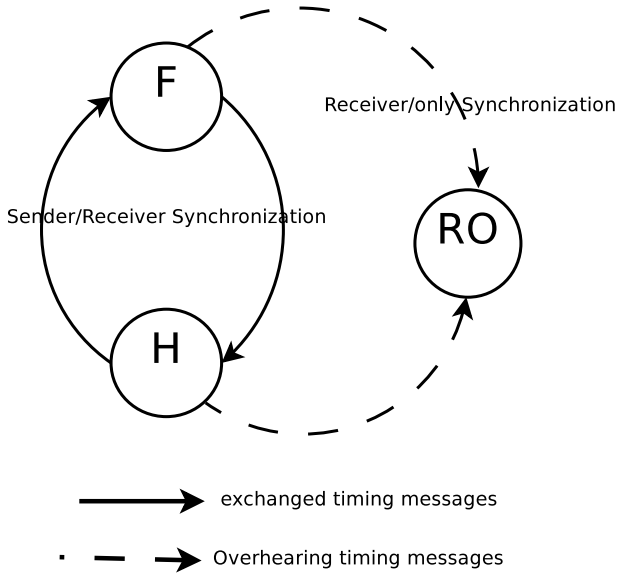


Figure 2: RO Synchronization principle.

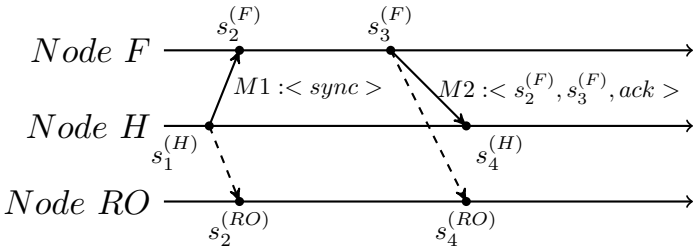


Figure 3: Clock synchronization model of RO methodology.

packet carries the timestamps $s_2^{(F)}$ and $s_3^{(F)}$. Upon receiving M_2, node H records its arrival time $s_4^{(H)}$. Using the timestamp quadruples $(s_1^{(H)}, s_2^{(F)}, s_3^{(F)}, s_4^{(H)})$, node H can compute clock difference with node F, $\delta^{(H,F)}$, as follows:

$$\delta^{(H,F)} = \frac{(s_2^{(F)} - s_1^{(H)}) - (s_4^{(H)} - s_3^{(F)})}{2} \tag{1}$$

Due to the wireless broadcast nature, a node RO within the shared coverage area of nodes H and F can receive two-way timing messages exchanged between them. Upon receiving the synchronization request M_1 and acknowledgment M_2, node RO records the timestamps $s_2^{(RO)}$ and $s_4^{(RO)}$ at which they were received. Since nodes F and RO are both neighbors of node H, they will receive M_1 nearly simultaneously. Thus, node RO can be synchronized directly from its own timestamp $s_2^{(RO)}$ and that of node F, i.e., $s_2^{(F)}$, received in the acknowledgment messages. Node RO does that without exchanging any messages with other nodes. The clock difference $\delta^{(RO,F)}$ between clocks of nodes RO and F is given by:

$$\delta^{(RO,F)} = s_2^{(F)} - s_2^{(RO)} \tag{2}$$

3.2. Clock Drift Model

Two clock models can be used to derive the ML estimator, namely: the clock offset model and the clock offset and skew model. The former estimates only offset and needs resynchronization to ensure long-term synchronization reliability, and thus several synchronization rounds are mandatory to maintain good accuracy. Meanwhile, the second model aims to estimate both properties of a clock to achieve sustained synchronization over time. It needs low timing messages and is therefore power-efficient. Both models are considered in this paper and are explained below.

3.2.1. Clock offset model

Assuming that there is no skew, the local clock $C(t)$ at real time t is given by [9]:

$$C(t) = t + \delta \tag{3}$$

where δ is the offset. Thus, the relationship time between nodes RO and F is given by [9]:

$$s_2^{(F)} = s_2^{(RO)} + \delta^{(RO,F)} + D^{(RO,F)} \tag{4}$$

where $\delta^{(RO,F)}$ represents the relative offset between clocks of nodes RO and F. A delay between sender and receiver is composed of a *fixed* (deterministic) part denoted by fd and a *variable* (random) part denoted by vd. Let $d^{(H,F)}$ and $d^{(H,RO)}$ indicate the time delays of messages from node H to nodes F and RO, respectively. Thus, $d^{(H,F)} = fd^{(H,F)} + vd^{(H,F)}$ and $d^{(H,RO)} = fd^{(H,RO)} + vd^{(H,RO)}$. The fixed parts are presumed to be identical, while the variable parts are modeled as normally distributed random variables with mean μ and variance σ_0^2. It follows that $d^{(H,RO)} - d^{(H,F)} = vd^{(H,RO)} - vd^{(H,F)}$. Thus, $D^{(RO,F)}$ equals $vd^{(H,RO)} - vd^{(H,F)}$, i.e., $D^{(RO,F)}$ is the difference between two normal distributions characterized by identical means and variances. Therefore, $D^{(RO,F)}$ is a random variable with $\mu = 0$ and $\sigma^2 = 2\sigma_0^2$; i.e., $D^{(RO,F)} \sim \mathcal{N}(0, \sigma^2)$.

3.2.2. Clock offset and skew model

Considering both properties of clocks, the local clock $C(t)$ at real time t is given by [9]:

$$C(t) = \omega.t + \delta \tag{5}$$

where ω denotes the skew and δ represents the offset. Thus, The relationship between the clocks of nodes RO and F is expressed as [9]:

$$s_2^{(F)} = \omega^{(RO,F)}.s_2^{(RO)} + \delta^{(RO,F)} + D^{(RO,F)} \tag{6}$$

where $\omega^{(RO,F)}$ refers to the relative skew, and $\delta^{(RO,F)}$ denotes the relative offset between clocks of nodes RO and F. $D^{(RO,F)}$ is as defined in the clock offset model, discussed above.

4. Model of Time Synchronization

This section presents a novel time synchronization model that extends the RO methodology to allow synchronizing RO nodes to multiple mutually-synchronized references. Both clock models are derived and analyzed.

4.1. Problem Statement and Main Idea

As described in the RO methodology, when node H synchronizes with reference node F through two-way message exchanges, RO nodes located in the overlapping region of nodes H and F can achieve synchronization by overhearing the synchronization and acknowledgment messages. This methodology improves performance by reducing the number of timing messages, thereby lowering energy consumption. However, the RO node must be a neighbor of both nodes H and F. More importantly, synchronizing to a single reference node F makes the process vulnerable to failure or malicious behavior, where the reference node could send faulty timestamps to disrupt clock synchronization.

The proposed time synchronization model addresses these limitations, enabling RO nodes to synchronize with multiple reference nodes. This model improves clock accuracy and enhances system resilience against node failure and malicious behavior. The main idea is to utilize all neighboring nodes H of the RO node, including those whose reference nodes are outside the RO node's communication range. The timestamps sent by the reference nodes are forwarded to the RO node by nodes H through an additional timing message. This method allows the RO node to synchronize with multiple reference nodes. Fig. 4 illustrates the extension of RO methodology to achieve this goal. Let $j \in \{1, \ldots, J\}$ denote the jth neighbor node H_j of node RO. Let $F_{(j,k)}$ with $k \in \{1, \ldots, K_j\}$ represent the kth neighbor of node H_j. Node RO is an unsynchronized neighbor of node H_j, and node $F_{(j,k)}$ is a synchronized neighbor of node H_j. Therefore, node RO can synchronize its clock to S synchronized reference nodes $F_{(j,k)}$ via the intermediate nodes H_j, where:

$$S = \sum_{j=1}^{J} K_j \tag{7}$$

Unlike other RO-based approaches such as PBS, $F_{(j,k)}$ and RO do not need to be direct neighbors. The restrictive requirement that RO nodes be neighbors of both $F_{(j,k)}$ and H_j is thus removed.

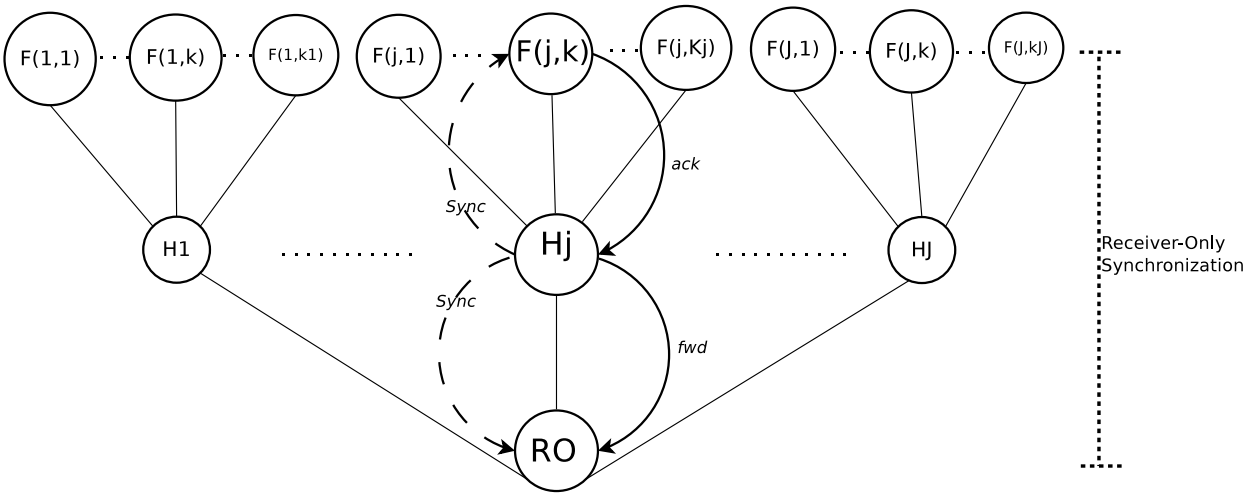


Figure 4: Extension of RO methodology.

4.2. Detailed Description

Fig. 5 provides a detailed description of the proposed clock synchronization model. The synchronization process for node *RO* proceeds as follows:

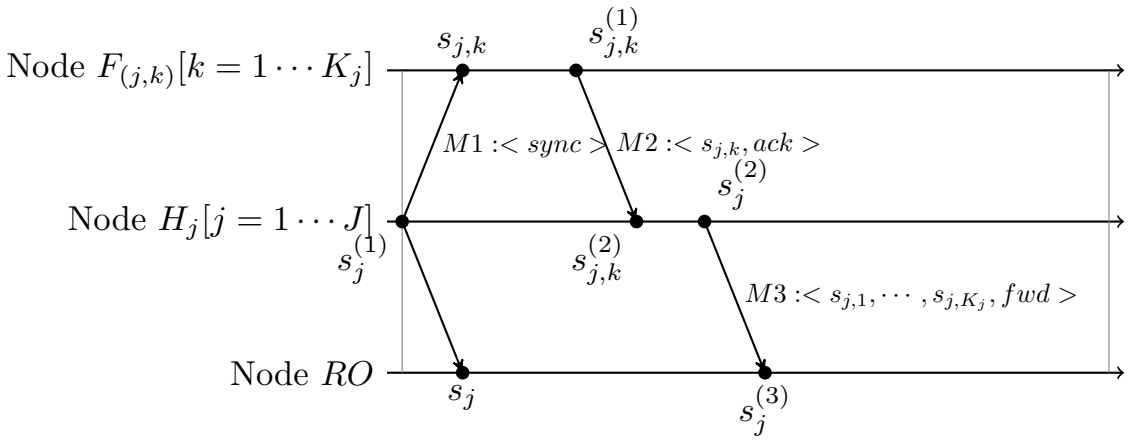


Figure 5: Clock synchronization model.

1. Each node H_j broadcasts a synchronization message M_1 at time $s_j^{(1)}$.
2. Nodes RO and $F_{(j,k)}$ receive M_1 at their local time s_j and $s_{j,k}$, respectively.
3. At time $s_{j,k}^{(1)}$, node $F_{(j,k)}$ sends back an acknowledgment message M_2, containing M_1's arrival time $s_{j,k}$.
4. Node H_j receives M_2 at $s_{j,k}^{(2)}$.
5. After node H_j collects timestamps from all nodes $F_{(j,k)}$, it broadcasts a forwarding message M_3 at time $s_j^{(2)}$, which includes the list of timestamps $\{s_{j,1}, ..., s_{j,K_j}\}$.
6. At $s_j^{(3)}$ node RO receives the timestamps from each node H_j.

After collecting a list of timestamps from each node H_j, node RO compiles Table 4. The first column represents the list of timestamps s_j corresponding to nodes H_j. Each row of timestamps s_j represents the list of timestamps $s_{j,k}$ received from nodes $F_{(j,k)}$. To deal with node failure, nodes RO, H_j and $F_{(j,k)}$ use timeouts for receiving M_3 (the list of timestamps), M_2 (the individual timestamp) and M_1 (the synchronization message), respectively.

s_1	$s_{1,1}$	$\cdots$	$s_{1,k}$	$\cdots$	s_{1,K_1}
$\vdots$	$\vdots$		$\vdots$		$\vdots$
$\mathbf{s_j}$	$s_{j,1}$	$\cdots$	$\mathbf{s_{j,k}}$	$\cdots$	s_{j,K_j}
$\vdots$	$\vdots$		$\vdots$		$\vdots$
s_J	$s_{J,1}$	$\cdots$	$s_{J,k}$	$\cdots$	s_{J,K_J}

Table 4: The list of timestamps obtained by node RO.

Thus, node RO can compute the clock offset and skew by applying RO methodology using the timestamps shown in Table 4. In the next sub-section, both clock models are derived and analyzed.

As discussed above, several nodes H_j can send multiple M_1 messages to node $F_{(j,k)}$, which sends back the receive time for each M_1 message. To reduce medium access collisions and network congestion caused by sending back the arrival times of each M_1 message separately, node $F_{(j,k)}$ concatenates the recorded receive times with the corresponding identities of H_j into a only one packet and broadcasts this combined packet. Node H_j then extracts from the received packet the time at which its own M1 message was received.

4.3. Clock Estimators and Analysis

In what follows, the ML estimators and the corresponding CRLB are derived for both clock models.

4.3.1. Clock offset Model

By applying equation (4) to our time synchronization model, the time of nodes RO and $F_{(j,k)}$ is related as follows:

$$D^{(RO,F_{(j,k)})} = s_{j,k} - s_j - \delta^{(RO,F_{(j,k)})} \tag{8}$$

where $\delta^{(RO,F_{(j,k)})}$ is the relative offset between clocks of nodes RO and $F_{(j,k)}$, and $D^{(RO,F_{(j,k)})}$ is as defined as explained in Section 3.2.1. Let S denotes the number of received timestamps. The S values of $D^{(RO,F_{(j,k)})}$, denoted as DS, are presented by:

$$DS = \{\underbrace{D^{(RO,F_{(1,1)})}, \ldots, D^{(RO,F_{(j,k)})}, \ldots, D^{(RO,F_{(J,K_J)})}}_{\mathrm{S}}\}$$

The likelihood function for ($\delta^{(RO,F_{(j,k)})}$,σ^2), based on the observations $\{s_j\}_{1\le j\le J}$ and $\{s_{j,k}\}_{1\le j\le J}^{1\le k\le K_j}$, is given by:

$$\begin{aligned} f\big(\delta^{(RO,F_{(j,k)})}, \sigma^2 \mid DS\big) &= \prod_{j=1}^{J}\Big(\prod_{k=1}^{K_j}\frac{1}{\sqrt{2\pi\sigma^2}} e^{\frac{-1}{2\sigma^2}(D^{(RO,F_{(j,k)})})^2}\Big) \\ &= \Big(\frac{1}{\sqrt{2\pi\sigma^2}}\Big)^S e^{\frac{-1}{2\sigma^2}\sum\limits_{j=1}^{J}\Big(\sum\limits_{k=1}^{K_j}(s_{j,k}-s_j-\delta^{(RO,F_{(j,k)})})^2\Big)} \end{aligned} \tag{9}$$

Thus, the ML estimator of $\delta^{(RO,F_{(j,k)})}$ is derived by taking the derivative of the log-likelihood function with regard to $\delta^{(RO,F_{(j,k)})}$, resulting in:

$$\delta^{(RO,\hat{F}_{(j,k)})} = argmax(\ln f(\delta^{(RO,F_{(j,k)})} \mid DS\big) = \frac{1}{S}\times\sum_{j=1}^{J}\Big(\sum_{k=1}^{K_j}\big(s_{j,k}-s_j\big)\Big) \tag{10}$$

To evaluate the performance of the derived ML estimator for $\delta^{(RO,F_{(j,k)})}$, we calculate the associated Cramer-Rao Lower Bound (CRLB). The CRLB represents the theoretical minimum variance achievable by any unbiased estimator. The CRLB for the vector parameter $\theta = [\delta^{(RO,F_{(j,k)})}]^T$ is computed from the inverse of the 1×1 Fisher Information matrix, $I^{-1}(\theta)$. This follows from the bound $Var(\hat{\theta}) \ge [I^{-1}(\theta)]$. The CRLB for the offset $\delta^{(RO,F_{(j,k)})}$ is given by:

$$Var(\hat{\delta}^{(RO,F_{(j,k)})}) \ge (I^{-1}(\delta^{(RO,F_{(j,k)})})) = \frac{\sigma^2}{S} \tag{11}$$

The mean square error (MSE) performance of the proposed estimator of $\delta^{(RO,F_{(j,k)})}$ has been numerically evaluated and compared with the corresponding CRLB. The simulation results comparing the MSE of $\delta^{(RO,F_{(j,k)})}$ with

its corresponding CRLB are presented in Fig. 6. Each data point in the plots represents the mean obtained from 10^4 independent simulations. With an increasing number of timestamps S, the MSE of $\delta^{(RO,F_{(j,k)})}$ quickly converges to the CRLB.

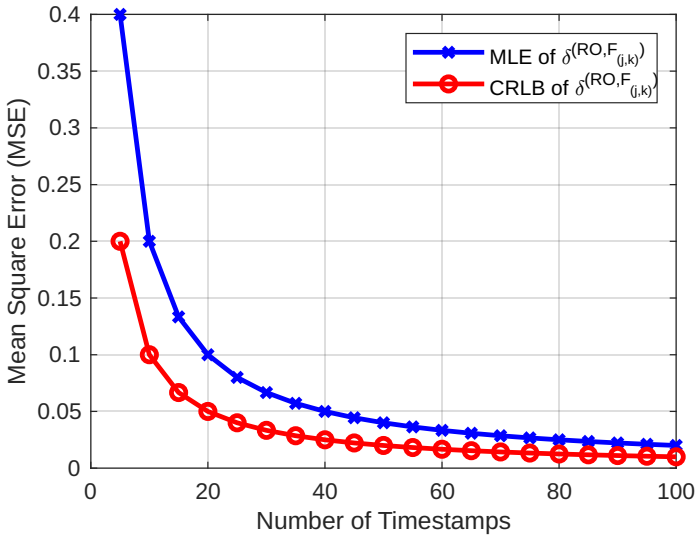


Figure 6: MSE performance of offset-only estimation vs. the number of timestamps.

4.3.2. Clock offset and skew Model

Applying equation (6) to our time synchronization model, the relationship between the clocks of nodes RO and $F_{(j,k)}$ is expressed as:

$$D^{(RO,F_{(j,k)})} = s_{j,k} - \omega^{(RO,F_{(j,k)})}.s_j - \delta^{(RO,F_{(j,k)})} \tag{12}$$

where $\omega^{(RO,F_{(j,k)})}$ refers to the relative skew, and $\delta^{(RO,F_{(j,k)})}$ denotes the relative offset between clocks of nodes RO and $F_{(j,k)}$. $D^{(RO,F_{(j,k)})}$ is as defined in Section 3.2.1. The likelihood function for ($\delta^{(RO,F_{(j,k)})}$,$\omega^{(RO,F_{(j,k)})}$,σ^2), based on the observations $\{s_j\}_{1\leq j\leq J}$ and $\{s_{j,k}\}_{1\leq j\leq J}^{1\leq k\leq K_j}$, is given by:

$$\begin{aligned} f\left(\delta^{(RO,F_{(j,k)})}, \omega^{(RO,F_{(j,k)})}, \sigma^2 \mid DS\right) &= \prod_{j=1}^{J}\left(\prod_{k=1}^{K_j}\frac{1}{\sqrt{2\pi\sigma^2}} e^{\frac{-1}{2\sigma^2}(D^{(RO,F_{(j,k)})})^2}\right) \\ &= \left(\frac{1}{\sqrt{2\pi\sigma^2}}\right)^S e^{\frac{-1}{2\sigma^2}\sum\limits_{j=1}^{J}\left(\sum\limits_{k=1}^{K_j}(s_{j,k}-\omega^{(RO,F_{(j,k)})}.s_j-\delta^{(RO,F_{(j,k)})})^2\right)} \end{aligned} \tag{13}$$

Hence, the ML estimator of $\omega^{(RO,F_{(j,k)})}$ and $\delta^{(RO,F_{(j,k)})}$ is derived by taking the derivative of the log-likelihood function with regard to $\omega^{(RO,F_{(j,k)})}$ and $\delta^{(RO,F_{(j,k)})}$, resulting in:

$$\hat{\omega}^{(RO,F_{(j,k)})} = \frac{\sum\limits_{j=1}^{J}\left(\sum\limits_{k=1}^{K_j} s_{j,k}\right)\times\sum\limits_{j=1}^{J}(\sum\limits_{k=1}^{K_j} s_j) - S\sum\limits_{j=1}^{J}\left(\sum\limits_{k=1}^{K_j} s_{j,k}.s_j\right)}{\left(\sum\limits_{j=1}^{J}(\sum\limits_{k=1}^{K_j} s_j)\right)^2 - S\sum\limits_{j=1}^{J}\left(\sum\limits_{k=1}^{K_j}(s_j)^2\right)} \tag{14}$$

$$\hat{\delta}^{(RO,F_{(j,k)})} = \frac{1}{S}\left[\sum_{j=1}^{J}(\sum_{k=1}^{K_j} s_{j,k}) - \hat{\omega}^{(RO,F_{(j,k)})}\times\sum_{j=1}^{J}(\sum_{k=1}^{K_j} s_j)\right] \tag{15}$$

The CRLB for the vector parameter $\theta = [\delta^{(RO,F_{(j,k)})}, \omega^{(RO,F_{(j,k)})}]^T$ is computed from the inverse of the 2×2 Fisher Information matrix, $I^{-1}(\theta)$. This follows from the bound $Var(\hat{\theta}_i) \geq [I^{-1}(\theta)]_{i,i}$. The result is:

$$Var(\hat{\delta}^{(RO,F_{(j,k)})}) \geq (I^{-1}(\theta))_{1,1} = \frac{\sigma^2 \sum_{j=1}^{J} \left(\sum_{k=1}^{K_j} (s_j)^2 \right)}{S \sum_{j=1}^{J} \left(\sum_{k=1}^{K_j} (s_j)^2 \right) - \left(\sum_{j=1}^{J} \left(\sum_{k=1}^{K_j} s_j \right) \right)^2} \tag{16}$$

$$Var(\hat{\omega}^{(RO,F_{(j,k)})}) \geq (I^{-1}(\theta))_{2,2} = \frac{S\sigma^2}{S \sum_{j=1}^{J} \left(\sum_{k=1}^{K_j} (s_j)^2 \right) - \left(\sum_{j=1}^{J} \left(\sum_{k=1}^{K_j} s_j \right) \right)^2} \tag{17}$$

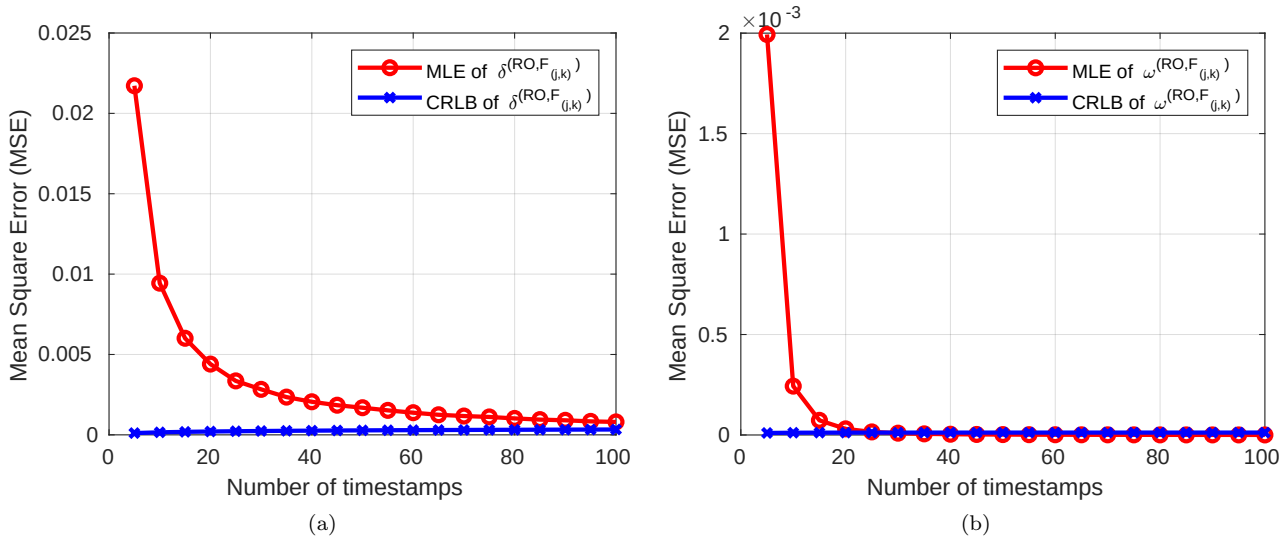


Figure 7: MSE performance of skew $\omega^{(RO,F_{(j,k)})}$ and offset $\delta^{(RO,F_{(j,k)})}$ estimation vs. the number of timestamps.

Following an evaluation scenario similar to the offset model, the MSE performance of $\omega^{(RO,F_{(j,k)})}$ and $\delta^{(RO,F_{(j,k)})}$ has been numerically evaluated and compared with the corresponding CRLB. The simulation results comparing the MSE of the obtained estimators and the corresponding CRLB are presented in Fig. 7. With an increasing number of timestamps S, the MSE of $\omega^{(RO,F_{(j,k)})}$ and $\delta^{(RO,F_{(j,k)})}$ quickly converges to the CRLB.

4.4. Communication and Computational Complexity

The complexity of the proposed time synchronization model can be analyzed in terms of communication and computational overhead. Each node H_j exchanges two timing messages: the synchronization message M_1 and the forwarding message M_3. Each reference node $F_{(j,k)}$ sends an acknowledgment message M_2 to node H_j. For synchronization with a single reference node, this requires two messages per H_j and one message per $F_{(j,k)}$. Therefore, the total number of timing messages for an RO node to synchronize with S reference nodes via J neighbors H_j is $O(J + S)$. Thus, the communication complexity depends on the number of neighbors H_j and the number of reference nodes they can hear from.

For computational complexity, each node H_j collects K_j timestamps from its reference nodes, resulting in $O(K_j)$ operations per H_j. The RO node collects S timestamps from its neighbor nodes H_j and calculates clock skew and offset using these S timestamps from reference nodes $F_{(j,k)}$. The runtime complexity of these calculations is linear, i.e., $O(S)$. Hence, the computational complexity at the RO node depends on the number S of reference nodes.

Overall, the complexity is linear for both communication and computation. This makes the proposed time synchronization model significantly more efficient than consensus-based protocols or protocols with multi-round requirements, where the communication complexity often around $O(n^2)$ or higher. As a result, the model is well-suited for resource-constrained environments.

In hostile environments, the network could be subject to attacks. Particularly, malicious nodes (MNs) may try to inject faulty timestamps to degrade the synchronization accuracy. In the next section, we will see how the RO nodes can prevent faulty timestamps injection attacks in the time synchronization process.

5. Detection of Faulty Timestamps Injection Attacks

5.1. Attack Model

We assume that the attacker (i.e., malicious node) has the ability to inject faulty timestamps in the time synchronization process. Faulty timestamps can be modeled as adding a negative value $-\Delta$ or positive value $+\Delta$ to the correct timestamps. Nodes $F_{(j,k)}$ and/or H_j can be MNs and send faulty timestamps $s^*_{j,k}$, which are given by:

$$s^*_{j,k} = s_{j,k} + \Delta \tag{18}$$

or

$$s^*_{j,k} = s_{j,k} - \Delta \tag{19}$$

Consequently, nodes RO would calculate incorrect time difference $\delta^{(RO,F_{(j,k)})^*}$ as follows:

$$\begin{aligned} \delta^{(RO,F_{(j,k)})^*} &= (s^*_{j,k} - s_j) \\ &= (s_{j,k} + \Delta - s_j) \\ &= (s_{j,k} - s_j) + \Delta \\ &= \delta^{(RO,F_{(j,k)})} + \Delta \end{aligned} \tag{20}$$

or

$$\begin{aligned} \delta^{(RO,F_{(j,k)})^*} &= (s^*_{j,k} - s_j) \\ &= (s_{j,k} - \Delta - s_j) \\ &= (s_{j,k} - s_j) - \Delta \\ &= \delta^{(RO,F_{(j,k)})} - \Delta \end{aligned} \tag{21}$$

Below, the delay thresholds and the corresponding faulty timestamps detection technique are presented.

5.2. Delay Thresholds

In our time synchronization model, nodes RO receive S timestamps $\{s_{j,k}\}_{1\le j\le J}^{1\le k\le K_j}$, as enumerated in Table 4, from several reference nodes $F_{(j,k)}$.To prevent faulty timestamps injection attacks from MNs in the time synchronization process, we use delay thresholds to detect faulty timestamps which will not be used for updating the clocks of RO nodes.

In Fig. 5, let $s_j^{(2)}$ represents the transmission time of the forwarding message from node H_j to node RO, while $s_j^{(3)}$ denotes its arrival time at node RO. The value $s_j^{(3)}$ can be represented by:

$$s_j^{(3)} = s_j^{(2)} + \delta_i^{(H_j,RO)} + d^{(H_j,RO)} \tag{22}$$

where $\delta_i^{(H_j,RO)}$ represents the relative offset between clocks of nodes H_j and RO. $d^{(H_j,RO)}$ denotes the transmission delay of the forwarding message from node H_j to node RO. From equation (22), we find:

$$\delta_i^{(H_j,RO)} = s_j^{(3)} - s_j^{(2)} - d^{(H_j,RO)} \tag{23}$$

Following the triangle law, as shown in equation (24) [40], if the nodes RO, H_j and $F_{(j,k)}$ honestly compute their time offsets, then:

$$\delta_i^{(RO,H_j)} + \delta_i^{(H_j,F_{(j,k)})} + \delta_i^{(F_{(j,k)},RO)} = 0 \tag{24}$$

where $\delta_i^{(RO,H_j)}$, $\delta_i^{(H_j,F_{(j,k)})}$, and $\delta_i^{(F_{(j,k)},RO)}$ are, respectively, the relative offsets between the clocks of nodes RO and H_j, H_j and $F_{(j,k)}$, and $F_{(j,k)}$ and RO. For the SR methodology, the delay $d^{(H_j,F_{(j,k)})}$ and the offset $\delta_i^{(H_j,F_{(j,k)})}$ between nodes H_j and $F_{(j,k)}$ can be given by [7]:

$$d^{(H_j,F_{(j,k)})} = \frac{(s_{j,k} - s_j^{(1)}) + (s_{j,k}^{(2)} - s_{j,k}^{(1)})}{2} \tag{25}$$

$$\delta_i^{(H_j,F_{(j,k)})} = \frac{(s_{j,k} - s_j^{(1)}) - (s_{j,k}^{(2)} - s_{j,k}^{(1)})}{2} \tag{26}$$

where $s_j^{(1)}$ denotes the time at which the synchronization message is transmitted. $s_{j,k}^{(1)}$ represents the transmission time of the acknowledgment message from node $F_{(j,k)}$ to node H_j, while $s_{j,k}^{(2)}$ denotes its arrival time at node H_j, as shown in Fig. 5.From equations (23), (24) and (26), we find:

$$d^{(H_j,RO)} = (s_j^{(3)} - s_j^{(2)}) + (s_{j,k} - s_j) - \frac{(s_{j,k} - s_j^{(1)}) - (s_{j,k}^{(2)} - s_{j,k}^{(1)})}{2} \tag{27}$$

The propagation delay follows a Gaussian distribution (as indicated in [5], [26] and [24]). With a 99.97% confidence, it is bounded by $d_{avg} - 3\sigma$ as the minimum delay and $d_{avg} + 3\sigma$ as the maximum delay, where d_{avg} represents the mean delay and σ denotes the standard deviation. Consequently, the minimum and maximum delays for $d^{(H_j,F_{(j,k)})}$ can be set as follows:

$$d^{(H_j,F_{(j,k)})^{*min}} = d^{(H_j,F_{(j,k)})^{avg}} - 3\sigma \tag{28}$$

$$d^{(H_j,F_{(j,k)})^{*max}} = d^{(H_j,F_{(j,k)})^{avg}} + 3\sigma \tag{29}$$

The minimum and maximum delays for $d^{(H_j,RO)}$ can, respectively, be set to:

$$d^{(H_j,RO)^{*min}} = d^{(H_j,RO)^{avg}} - 3\sigma \tag{30}$$

$$d^{(H_j,RO)^{*max}} = d^{(H_j,RO)^{avg}} + 3\sigma \tag{31}$$

Two types of delay thresholds, specifically, ($d^{(H_j,F_{(j,k)})^{*min}}$,$d^{(H_j,RO)^{*min}}$), and ($d^{(H_j,F_{(j,k)})^{*max}}$, $d^{(H_j,F_{(j,k)})^{*max}}$) are used for verify the correctness of $d^{(H_j,F_{(j,k)})}$ and $d^{(H_j,RO)}$. These delay thresholds can be estimated before network deployment with great precision [24, 26]. If the calculated delays $d^{(H_j,F_{(j,k)})}$ or $d^{(H_j,RO)}$ fall outside the range between the minimum and maximum delays, the corresponding timestamp is considered faulty.

Nodes *RO* can detect faulty timestamps $s_{j,k}^*$ using $d^{(H_j,F_{(j,k)})}$ or $d^{(H_j,RO)}$. However, the MN can be smart by modifying other timestamps to eliminate the introduced error Δ in delay calculation. To this end, two delays are used, if the errorΔ is not detected by $d^{(H_j,F_{(j,k)})}$, it will be detected by $d^{(H_j,RO)}$. Two cases are possible:

Case 1: The MN can modify the timestamps $s_{j,k}$ and $s_{j,k}^{(1)}$ at the same time: $s_{j,k}^* = s_{j,k} + \Delta$ and $s_{j,k}^{(1)*} = s_{j,k}^{(1)} + \Delta$. Node *RO* cannot detect such an error using $d^{(H_j,F_{(j,k)})}$, yet it will be able to detect it using $d^{(H_j,RO)}$.

Case 2: The MN can increase $s_{j,k}$ and $s_j^{(2)}$ by error Δ, and decrease $s_{j,k}^{(1)}$ by error Δ at the same time (i.e., $s_{j,k}^* = s_{j,k} + \Delta$, $s_j^{(2)*} = s_j^{(2)} + \Delta$ and $s_{j,k}^{(1)*} = s_{j,k}^{(1)} - \Delta$). Thus, node *RO* will be able to detect it using $d^{(H_j,F_{(j,k)})}$, rather than $d^{(H_j,RO)}$.

Therefore, both $d^{(H_j,F_{(j,k)})}$ and $d^{(H_j,RO)}$ are used to allow RO nodes to detect faulty timestamps injection attacks. The RO nodes do not use faulty timestamps for updating their clocks.

6. SRST Protocol

This section introduces the SRST protocol, which integrates the proposed time synchronization model with the faulty timestamp detection technique. Fig. 8 presents a sequence diagram of the SRST protocol.

6.1. Process of SRST

The SRST process consists of several key steps, which are detailed below:

Step 1: A single node is designated as the primary time reference, which can be synchronized with a global source such as a UTC server and broadcasts the correct timestamps to the network.

Step 2: During every synchronization round, the primary reference node transmits an *Init* message. Nodes designated as H_j are the receivers of this broadcast. The nodes *RO* synchronize their clocks to nodes $F_{(j,k)}$ using the intermediate nodes H_j.

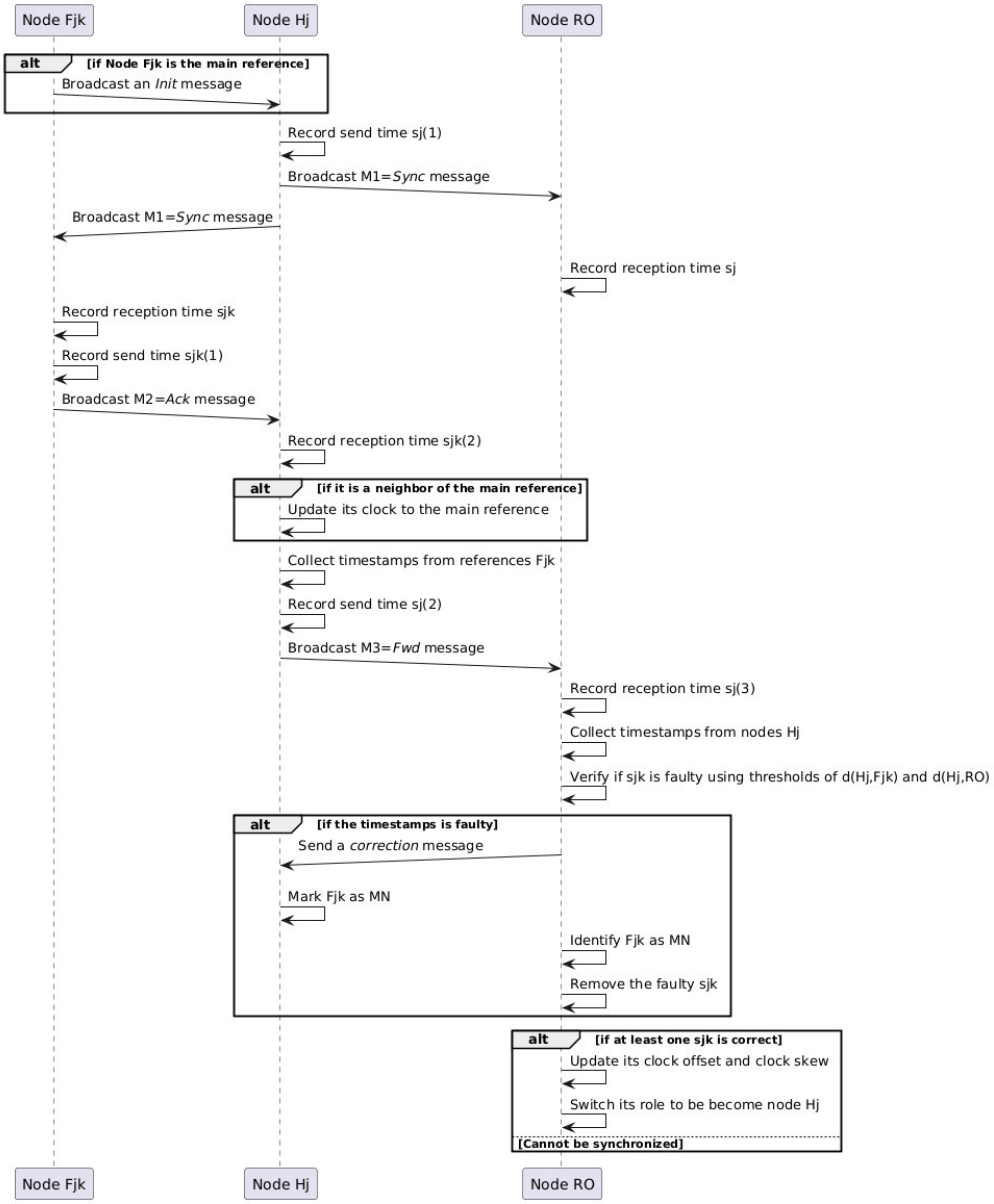


Figure 8: Sequence diagram description of SRST.

Step 3: Node H_j broadcasts a synchronization message M_1 at time $s_j^{(1)}$. M_1 contains the H_j's identifier (ID_{H_j}); specifically, $M1 = (ID_{H_j}, Sync)$.

Step 4: Upon receiving a synchronization message M_1, the following cases are explored:

Case 1: If the receiver is a reference node $F_{(j,k)}$ of H_j then it records M_1's arrival time $s_{j,k}$. Node $F_{(j,k)}$ transmits an acknowledgment message M_2 back to node H_j at time $s_{j,k}^{(1)}$. The message contains the $F_{(j,k)}$'s identifier ($ID_{F_{(j,k)}}$), the H_j's identifier ID_{H_j} received in M_1, the M_1's arrival time $s_{j,k}$ and the timestamps $s_{j,k}^{(1)}$; specifically, $M2 = (ID_{F_{(j,k)}}, ID_{H_j}, s_{j,k}, s_{j,k}^{(1)}, Ack)$.

Case 2: If the receiver is a node RO of H_j then it records M_1's arrival time s_j.

Step 5: When the acknowledgment message M_2 is received by the corresponding H_j at time $s_{j,k}^{(2)}$, the included timestamps in M_2 are recorded. If H_j is a neighbor of the primary reference then it synchronizes its clock using the offset estimator given in equation (26). After node H_j collects all timestamps from nodes $F_{(j,k)}$, it broadcasts a forwarding message M_3 to nodes RO at time $s_j^{(2)}$. M_3 contains the H_j's identifier (ID_{H_j}), $s_j^{(1)}$, the timestamps

$List < s_{j,k} >$, $List < s_{j,k}^{(1)} >$, $List < s_{j,k}^{(2)} >$ and $s_j^{(2)}$; specifically, $M3 = (ID_{H_j}, s_j^{(1)}, List < s_{j,k} >, List < s_{j,k}^{(1)} >, List < s_{j,k}^{(2)} >, s_j^{(2)}, Fwd)$.

Step 6: When the forwarding message M_3 is received by node RO at time $s_j^{(3)}$ from each node H_j, node RO verifies the correctness of the received timestamps before synchronizing its clock. For each timestamp $s_{j,k}$ from Table 4, node RO calculates $d^{(H_j,F_{(j,k)})}$ and $d^{(H_j,RO)}$ given in equations (25) and (27), respectively.

if $\left(d^{(H_j,F_{(j,k)})} < d^{(H_j,F_{(j,k)})^{*min}}\right)$ or $\left(d^{(H_j,F_{(j,k)})} > d^{(H_j,F_{(j,k)})^{*max}}\right)$ or $\left(d^{(H_j,RO)} < d^{(H_j,RO)^{*min}}\right)$ or $\left(d^{(H_j,RO)} > d^{(H_j,RO)^{*max}}\right)$ **then:**

- Remove the faulty $s_{j,k}$ from the Table 4, because it is faulty;

- Add $(ID_{F_{(j,k)}}, ID_{H_j})$ to the list of MNs, where $ID_{F_{(j,k)}}$ is the identifier of the malicious reference node $F_{(j,k)}$, and ID_{H_j} is the identifier of the intermediate node H_j;

- Send a correction message to node H_j, the message contains:
$(ID_B, ID_{H_j}, ID_{F_{(j,k)}}, Corr)$.

If Table 4 is empty, node RO cannot be synchronized. This can happen when node RO has only one reference node which happens to be MN, or all its reference nodes are MNs. Otherwise, node RO adjusts its clock using skew and offset estimators given in equations (14) and (15). Subsequently, node RO switches its role to be become node H_j to synchronize unsynchronized neighbor nodes. This process is iterated until synchronization is achieved across all network nodes.

Step 7: Upon receiving the correction message, H_j marks the intermediate node with $ID_{F_{(j,k)}}$, contained in M_3, as an MN instead of the reference node. Note that when H_j was an RO node, $F_{(j,k)}$ was an intermediate node of H_j.

Now, we discuss how the SRST protocol prevents attacks from MNs. Node $F_{(j,k)}$ and/or node H_j can be MNs and send faulty timestamps. Node RO can detect these faulty timestamps using the delay thresholds. The faulty timestamps will not be used in the offset calculation process, as shown in Step 6. However, it cannot determine who sends the faulty timestamps $F_{(j,k)}$ or H_j, because the timestamps are received from the intermediate node H_j. Initially, node RO identifies $F_{(j,k)}$ as MN. We have three cases: $F_{(j,k)}$ is MN, H_j is MN, or both $F_{(j,k)}$ and H_j are MNs.

Case 1: When $F_{(j,k)}$ is MN, node RO correctly identifies the MN.

Case 2: If H_j is an MN then node RO cannot correctly identify the MN. But when the correction message is received, node RO will identify node H_j as MN instead of the reference node $F_{(j,k)}$, as shown in Step 7.

Case 3: If both $F_{(j,k)}$ and H_j are MNs, node RO correctly identifies node H_j as MN when receiving the correction message. $F_{(j,k)}$ will be identified as MN by a node RO with another non-malicious node H_j. Otherwise, $F_{(j,k)}$ as well as the malicious nodes H_j are isolated from the time synchronization process.

In summary, SRST is secure, stays resilient to faulty timestamps and guarantees the correct time synchronization as long as node RO has at least one non-malicious reference node. It can tolerate up to $(S-1)$ faulty timestamps, where S is the number of nodes $F_{(j,k)}$. In the case when node RO has only one reference node $F_{(j,k)}$ which is MN, it cannot synchronize its clock. Nonetheless this case is rare because node RO synchronizes its clock to reference nodes within two hops, as shown in the time synchronization model. This guarantees that node RO owns at least two reference nodes.

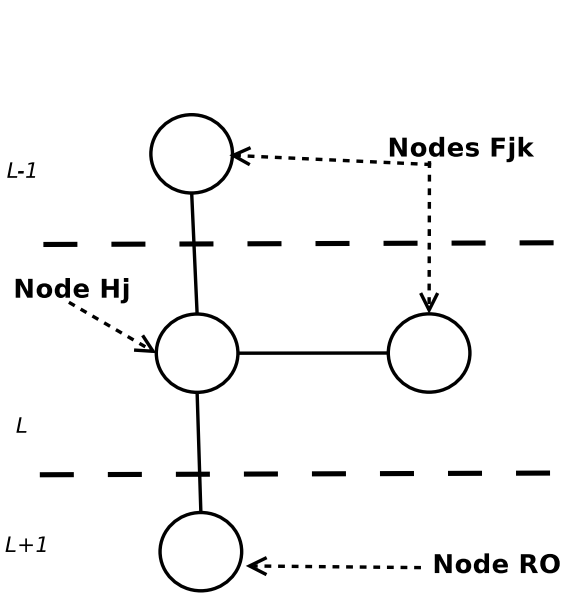
6.2. Multi-hop Support

In Section 6.1, we have seen that the primary reference node periodically starts the SRST protocol to synchronize the network nodes. The neighbor nodes of the primary reference node are nodes H_j. The nodes RO synchronize their clocks to nodes $F_{(j,k)}$ using the intermediate nodes H_j. Subsequently, nodes RO change role to act as nodes H_j to continue the multi-hop synchronization. This iterative process continues until all network nodes achieve synchronization. This section explains how we define the corresponding nodes RO and $F_{(j,k)}$ of the node H_j. Generally, an approach of time synchronization relies on a specific topology: multi-tier, cluster or flat. SRST is topology-independent and can be applied to any of these topologies.

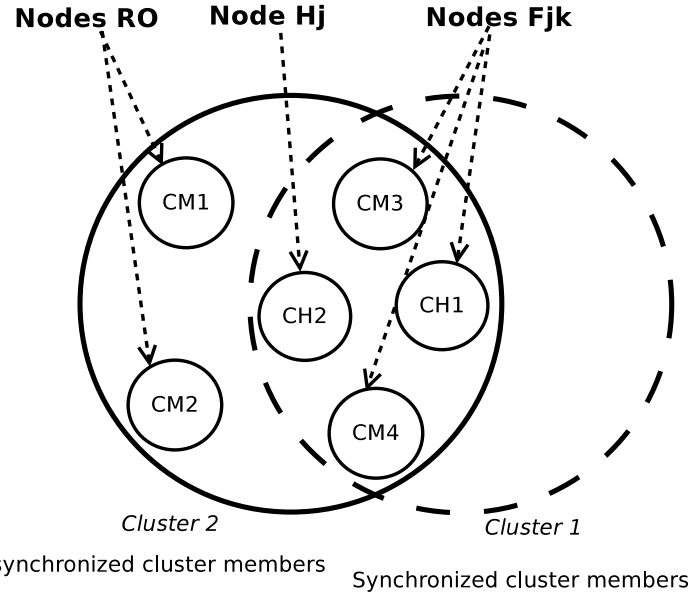
• *Multi-tier topology.* SRST is applicable to networks with a hierarchical structure, where each node is assigned a level based on its hop count from the primary reference node (as in [7]). The nodes of level L as nodes H_j apply SRST to synchronize the nodes of level $L+1$ to the nodes of levels L and $L-1$. Nodes RO are the neighbors of node H_j in level $L+1$. Nodes $F_{(j,k)}$ are the synchronized neighbors of node H_j in levels L and $L-1$ (See Fig. 9a).

• *Cluster-based topology.* SRST is also applicable to clustered networks, where each cluster is managed by a cluster-head capable of direct communication with all cluster members. Unsynchronized cluster members (nodes RO) would leverage the exchange of synchronization messages between H_j and other cluster members (nodes $F_{(j,k)}$). Fig. 9b depicts a scenario with two clusters, where $CH1$ leads cluster 1 and $CH2$ leads cluster 2. Cluster 1 is a synchronized cluster and cluster 2 an unsynchronized cluster. Nodes $CM3$, $CM4$ and $CH2$ are synchronized cluster members of cluster 1. Applying SRST, cluster-head $CH2$ synchronizes its unsynchronized cluster members $CM1$ and $CM2$ to synchronized cluster members $CM3$, $CM4$ and $CH2$.

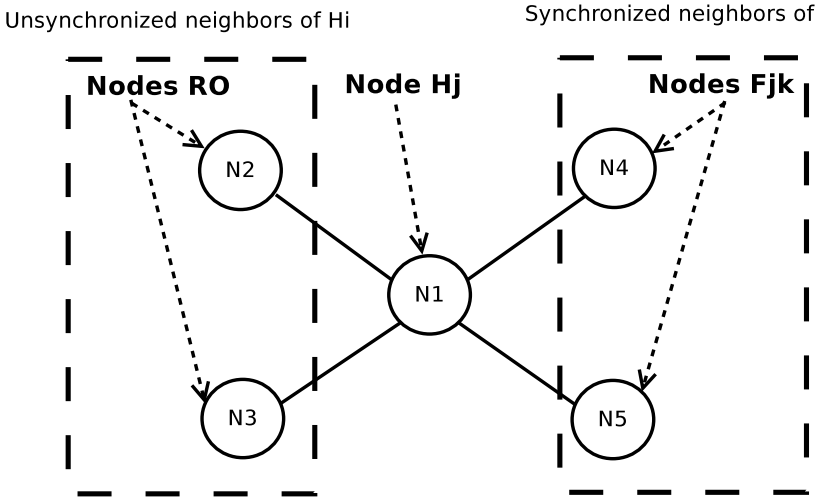
• *Flat topology.* A flat network topology does not have hierarchical levels or clusters. Using SRST, synchronized nodes H_j facilitate the synchronization of their unsynchronized neighbors (nodes RO) with their synchronized neighbors (nodes $F_{(j,k)}$). Fig. 9c illustrates an example, where node $N1$ applies SRST to synchronize its neighbors $N2$ and $N3$ with $N4$ and $N5$.



(a) Multi-tier topology.



(b) Cluster-based topology.



(c) Flat topology.

Figure 9: Multi-hop Support for SRST.

7. Implementation and Performance Evaluation

To validate the performance of SRST, we have used simulation and experiments with real devices. Specifically, we have employed the Avrora simulator, a widely recognized tool for simulating sensor networks. The key advantages of this simulator include its ability to provide cycle-accurate simulation of clock behavior and accurate emulation of low-power microcontrollers, which enables precise measurement of synchronization errors, and accurately models timing delays and drifts between nodes. Each node in the simulation has been represented as a resource-constrained device with low-power microcontrollers. Nodes communicate using the IEEE 802.15.4 communication protocol. The performance of SRST is compared with that of STSP [28], PBS [9], SPiRT [10], ATS [13], TPSN [7] and FTSP [8]. Notably, STSP is a secure protocol designed to mitigate faulty timestamps. SRST protocol and the other baseline protocols have been implemented on the nodes using the TinyOS 2.1.2 operating system, and the simulation has been conducted for 300 seconds. Each node provides its synchronized time after a synchronization round of 3 seconds. The synchronized time among nodes is used to compute the synchronization error and assess the accuracy. The MAC-layer timestamping is applied in all protocols.

7.1. Theoretical Communication Overhead Analysis

This section compares SRST with PBS [9] and SPiRT [10] protocols, analyzing the number of timing messages needed to compute clock skew and offset. In Fig. 4, we use J, M, S, and N to denote the number of nodes H_j, RO, $F_{(j,k)}$, and the rounds number, respectively.

Let N_{PBS} be the number of synchronization messages required by PBS. Every node H_j must apply the SR methodology with reference node $F_{(j,k)}$ for N synchronization rounds. Next, H_j becomes a reference node to its neighboring nodes RO, each one of these B nodes applying an SR methodology to synchronize with H_j. Thus,

$$N_{PBS} = (2 * J + 2 * M) * N = 2 * (J + M) * N \tag{32}$$

Let N_{SPiRT} be the number of synchronization messages required by SPiRT. Every node H_j must apply the SR methodology with reference node $F_{(j,k)}$ for N rounds to synchronize their nodes RO. Thus,

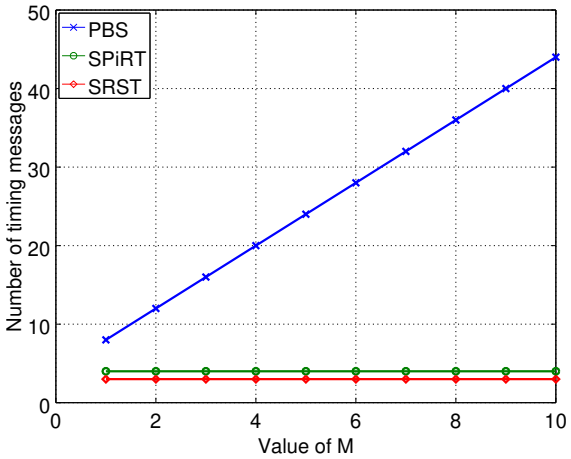
$$N_{SPiRT} = 2 * J * N \tag{33}$$

Let N_{SRST} be the number of synchronization messages required by SRST. Every node H_j sends a synchronization message and their reference nodes $F_{(j,k)}$ respond back with the timestamp. H_j then sends the list of timestamps to their corresponding nodes RO. Thus,

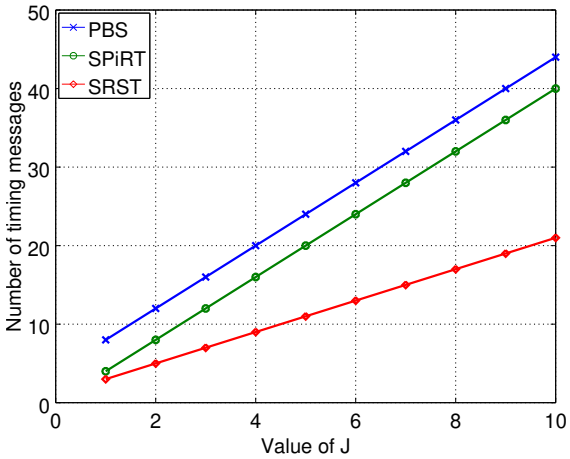
$$N_{SRST} = S + 2 * J \tag{34}$$

Fig. 10 shows the number of timing messages as M, J and S vary, with N fixed at 2 (the minimum for clock skew calculation). The message complexity of PBS increases proportionally with N and $J + M$. For SPiRT, the number of messages grows proportionally with N and J. However, the number of messages in SRST doesn't scale directly with either N or M. During each synchronization round in SRST, a node RO can derive joint skew/offset estimators using timing messages from S reference nodes, as detailed in Section 4.3. The results demonstrate that SRST's performance is superior to PBS and SPiRT in terms of communication efficiency.

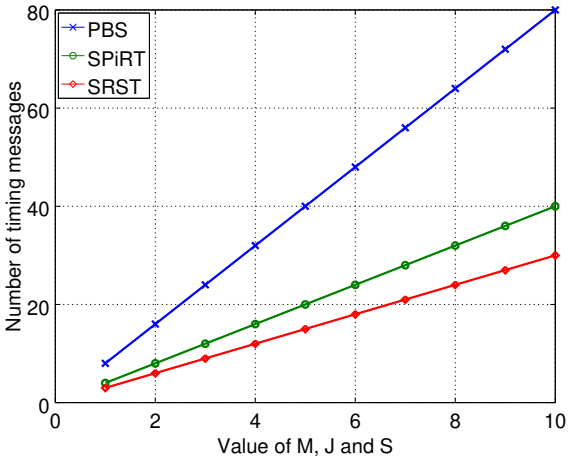
Moreover, RO-based synchronization protocols like PBS and SPiRT have a key requirement: the RO node must be within the shared coverage area of two leader nodes employing SR methodology. Achieving network-wide synchronization necessitates leader selection algorithms, such as the groupwise pair selection algorithm (GPA) used in PBS's multi-cluster extension. The formation of clusters in RO-based multi-hop time synchronization protocols increases the messaging overhead, and consequently consumes more energy. However, no extra overhead is needed to select reference nodes, intermediate nodes and RO nodes in SRST, as shown in Section 6.2.



(a) M, when $J = S = 1$



(b) J, when $M = S = 1$



(c) $M = J = S$

Figure 10: Number of timing messages vs M, J and S.

7.2. Synchronization Accuracy

To evaluate the synchronization error of SRST, we have used five nodes (real devices) as depicted in Fig. 11. We designate Node 1 as the primary time reference. Firstly, Node 2 and Node 3 synchronize their clocks to Node 1. Thus, the three Nodes become reference nodes to Node 4. The latter synchronizes its clock to Nodes 1, 2 and 3 by applying SRST. This process runs for several synchronization rounds. Following each round, node 5 broadcasts a beacon to all nodes, which is used to calculate the synchronization error. This error is determined by comparing the beacon reception times across different nodes. We have also conducted simulation experiments using Avrora. Fig. 11 presents the experimental and simulation results of the minimum, average and maximum synchronization error of SRST. From both experimental and simulation results, SRST is able to synchronize a node with an average accuracy of approximately 1 μs. In the best-case scenario, nodes were perfectly synchronized, while the worst-case synchronization error was approximately 4 μs.

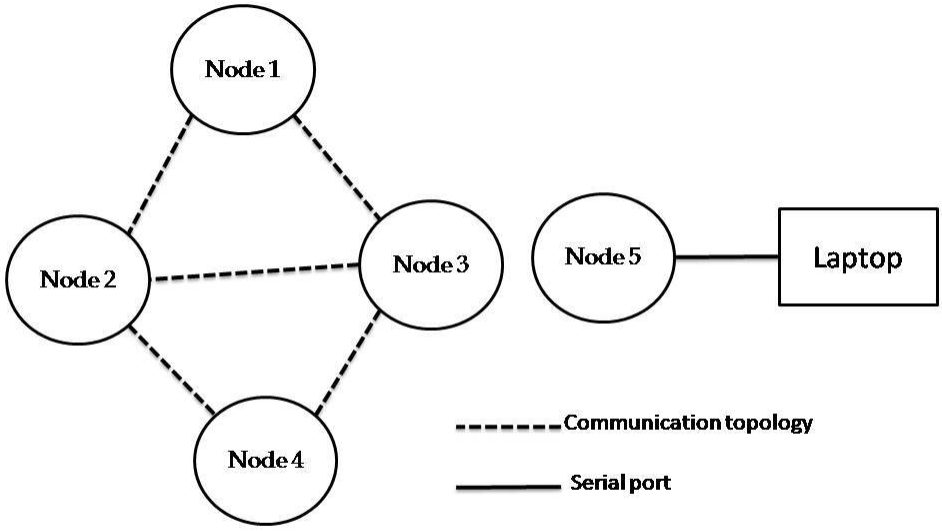
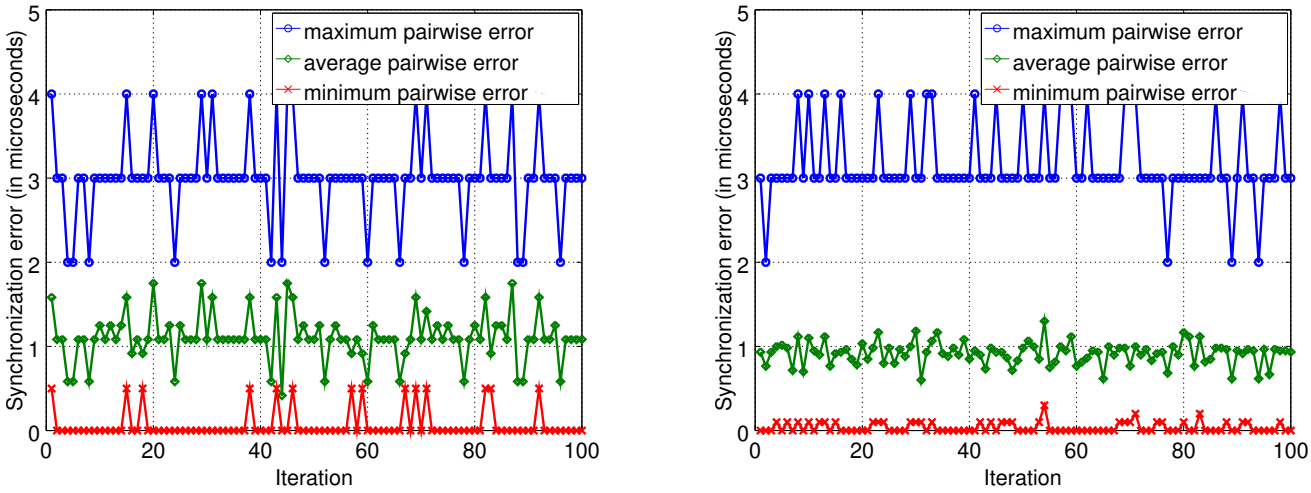


Figure 11: Experimental setup for evaluating SRST's synchronization accuracy.



(a) Experimental result.

(b) Simulation result.

Figure 12: Comparison of Experimental and Simulation results: Precision vs time.

We have also compared SRST with STSP [28], SPiRT [10], PBS [9], TPSN [7] and FTSP [8] in terms of accuracy. The mentioned protocols have been simulated using randomly placed nodes over 5×5 grid-based topologies as depicted in Fig. 13.

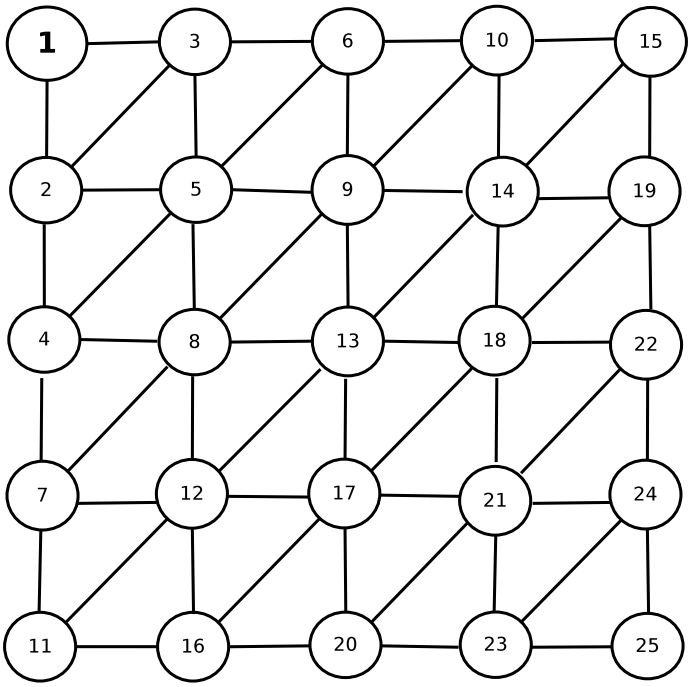


Figure 13: Topology used in the simulation.

In this topology, we have 9 levels where node 1 is the primary reference at level 0, nodes 2 and 3 at level 1, nodes 3, 4 and 5 at level 2, and so on. Some RO nodes can synchronize their clocks up to seven reference nodes to show the advantage of using multiple synchronized references compared to other protocols. For example, node 13 synchronizes its clock to nodes 4, 5, 6, 7, 8, 9 and 10. The nodes synchronize their clocks using each implemented protocol for several synchronization rounds. As discussed above, after each round, we use another node to compute the synchronization error. The simulation results have been obtained with a 99% confidence interval. Fig. 14 depicts the average synchronization error obtained from our simulations. From this figure, it can be seen that SRST can achieve a lower average synchronization error of approximately 0.75 μs. Thus, SRST clearly outperforms STSP, SPiRT, PBS, TPSN and FTSP in terms of precision.

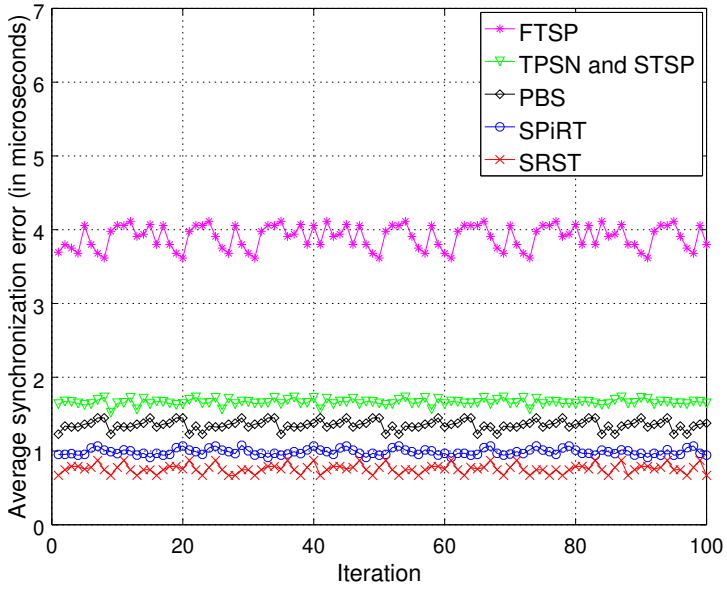


Figure 14: Simulation result: SRST vs STSP vs SPiRT vs PBS vs TPSN vs FTSP in terms of synchronization precision.

Convergence Rate: It indicates how many iterations are needed to reach a common global time by all clocks of network nodes. Faster convergence, achieved through fewer iterations, which leads to reducing inter-node communication and consequently decreasing energy consumption. To compare the convergence time of SRST with another distributed time synchronization, we have chosen ATS. Fig. 15 reports the convergence rate for SRST and ATS. The outcomes show that SRST achieves fast convergence. It converges towards less than 1 μs in one synchronization round by exchanging 68 synchronization messages. In the meantime, ATS was able to achieve this level of accuracy after 85 rounds by exchanging 25 synchronization messages per round, for a total of 2125 synchronization messages. This demonstrates that SRST accelerates convergence time by a factor of 31.25 compared to the ATS protocol. Both SRST and ATS utilize a consensus mechanism, where each node adjusts its clock based on a value derived from its neighbors' clocks. The offset between the consensus values of two neighboring nodes depends on the offset of their neighbors' clocks. As a result, when the offset between the clocks of two neighbors increases, the synchronization error grows as well. However, SRST achieves low synchronization error from the first synchronization round because the offset between the clocks of reference nodes is minimal.

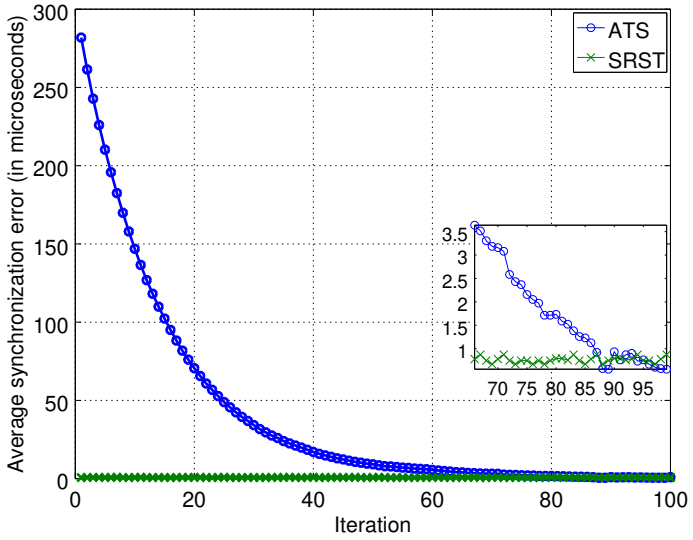


Figure 15: Comparison of SRST and ATS on convergence time.

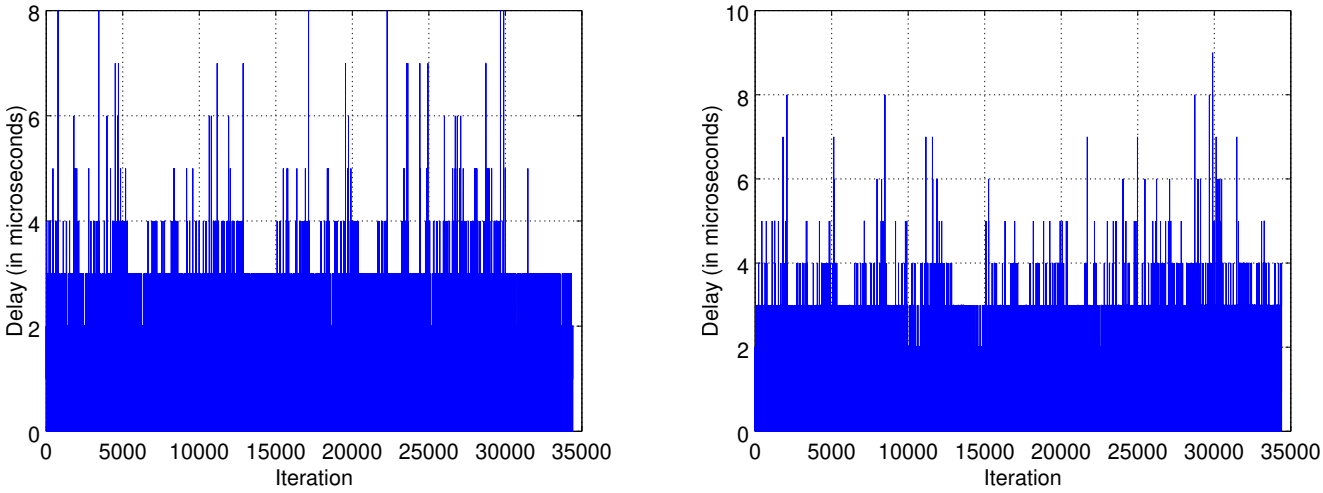
7.3. Security Performance

7.3.1. Delay Thresholds Estimation

The impact of a faulty timestamp on synchronization depends on the minimum and maximum delay thresholds. To make an accurate choice, we analyze $d^{(H_j,F_{(j,k)})}$ and $d^{(H_j,RO)}$ in the absence of faulty timestamps. We ran the SRST on the topology illustrated in Fig. 13 to compute the delay thresholds. For each node, we collected several readings of $d^{(H_j,F_{(j,k)})}$ and $d^{(H_j,RO)}$. Overall, we got 35000 delay measurements for all nodes. Fig. 16a and Fig. 16b illustrate the measured delay for each node across all runs. The statistical summary of these measurements is presented in Table 5. By knowing the delay thresholds $[d^{(H_j,F_{(j,k)})^{*max}}, d^{(H_j,RO)^{*max}}]$ and $[d^{(H_j,F_{(j,k)})^{*min}}, d^{(H_j,RO)^{*min}}]$, respectively, SRST is able to detect faulty timestamps by checking if the current delays $d^{(H_j,F_{(j,k)})}$ and $d^{(H_j,RO)}$ fall in the interval $[d^{(H_j,F_{(j,k)})^{*min}}, d^{(H_j,F_{(j,k)})^{*max}}]$ and $[d^{(H_j,RO)^{*min}}, d^{(H_j,RO)^{*max}}]$, respectively. From the obtained results, we find that: $d^{(H_j,F_{(j,k)})^{*min}}$=0.015 μs, $d^{(H_j,F_{(j,k)})^{*max}}$=5.5275 μs, $d^{(H_j,RO)^{*min}}$=0.062 μs and $d^{(H_j,RO)^{*max}}$=5.702 μs.

	$d^{(H_j,F_{(j,k)})}$	$d^{(H_j,RO)}$
Maximum(μs)(d^{max})	8	9
Average(μs)(d^{min})	2.7645	2.882
Standard deviation(σ)	0.921	0.94
minimum threshold(μs)(d^{*min})	0.015	0.062
maximum threshold(μs)(d^{*max})	5.5275	5.702

Table 5: Statistics of delays $d^{(H_j,F_{(j,k)})}$ and $d^{(H_j,RO)}$.



(a) Delay $d^{(H_j,F_{(j,k)})}$

(b) Delay $d^{(H_j,RO)}$

Figure 16: Delay measurements.

Maximum impact of MNs: The maximum delay factor, Δ, that goes undetected will be when $d^{(H_j,F_{(j,k)})} = d^{(H_j,F_{(j,k)})^{*min}}$ (or $d^{(H_j,F_{(j,k)})} = d^{(H_j,F_{(j,k)})^{*max}}$) and $d^{(H_j,RO)} = d^{(H_j,RO)^{*min}}$ (or $d^{(H_j,RO)} = d^{(H_j,RO)^{*max}}$). In this scenario, the MN can introduce a maximum error of $\Delta = 12\sigma$=11.052 μs for $d^{(H_j,F_{(j,k)})}$ and $\Delta = 12\sigma$=11.28 μs for $d^{(H_j,RO)}$. The RO node calculates $d^{(H_j,RO)}$ and $d^{(H_j,RO)}$ as follows:

$$d^{(H_j,F_{(j,k)})} = d^{(H_j,F_{(j,k)})^{*max}} - \frac{\Delta}{2} = d^{(H_j,F_{(j,k)})^{avg}} + 3\sigma - \frac{12\sigma}{2} \geq d^{(H_j,F_{(j,k)})^{*min}} \tag{35}$$

$$d^{(H_j,F_{(j,k)})} = d^{(H_j,F_{(j,k)})^{*min}} + \frac{\Delta}{2} = d^{(H_j,F_{(j,k)})^{avg}} - 3\sigma + \frac{12\sigma}{2} \leq d^{(H_j,F_{(j,k)})^{*max}} \tag{36}$$

And:

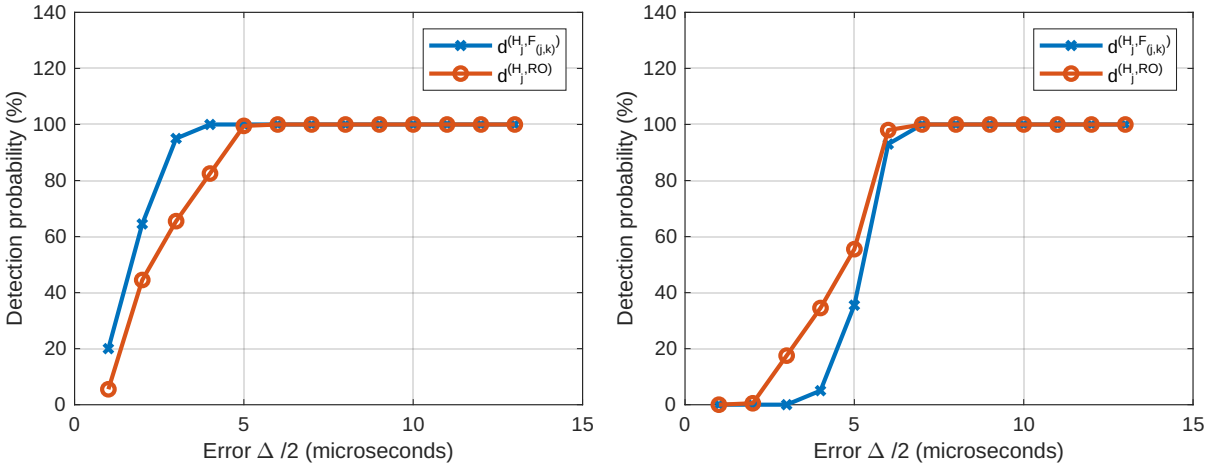
$$d^{(H_j,RO)} = d^{(H_j,RO)^{*max}} - \frac{\Delta}{2} = d^{(H_j,RO)^{avg}} + 3\sigma - \frac{12\sigma}{2} \geq d^{(H_j,RO)^{*min}} \tag{37}$$

$$d^{(H_j,RO)} = d^{(H_j,RO)^{*min}} + \frac{\Delta}{2} = d^{(H_j,RO)^{avg}} - 3\sigma + \frac{12\sigma}{2} \leq d^{(H_j,RO)^{*max}} \tag{38}$$

The calculated offset will deviate by $|\frac{\Delta}{2}|$, resulting in a maximum synchronization error of $|6\sigma|$. For $d^{(H_j,F_{(j,k)})}$, this corresponds to 5.526 μs, and for $d^{(H_j,RO)}$, it is 5.64 μs. Thus, the maximum impact of MNs on the synchronization process remains bounded within these error margins.

Detection rate of error Δ: To analyze the robustness to error Δ injected by MNs, we use four nodes; one of them is designated as H_j which initiates the SRST protocol. Two other nodes play the role of $F_{(j,k)}$ and RO, respectively. The fourth node is used for computing the synchronization error between the three nodes. To introduce an error Δ, the node RO was programmed to intentionally adds $\frac{\Delta}{2}$ to the computed delay before synchronizing its clock. The minimum and maximum delay thresholds for both $d^{(H_j,F_{(j,k)})}$ and $d^{(H_j,RO)}$ are fixed at 0 μs and 6 μs, respectively. We ran SRST for 200 synchronization rounds.

Fig. 17a and Fig. 17b plot the detection probability of negative error $-\Delta$ and positive error $+\Delta$, respectively, when using threshold-based detection of $d^{(H_j,F_{(j,k)})}$ and $d^{(H_j,RO)}$. Table 6 summarizes the statistics of synchronization error in faulty timestamps settings for multiple values of error Δ. The synchronization error has been computed relative to node $F_{(j,k)}$. The results demonstrate that MNs cannot degrade the synchronization accuracy between nodes $F_{(j,k)}$ and RO by more than 6 μs. The last column of Table 6 presents the detection rate corresponding to each error Δ. Notably, SRST achieves a 100 % success rate in detecting errors of 6 μs, ensuring robustness against malicious attacks.



(a) Negative error ($-\Delta$). (b) Positive error ($+\Delta$).

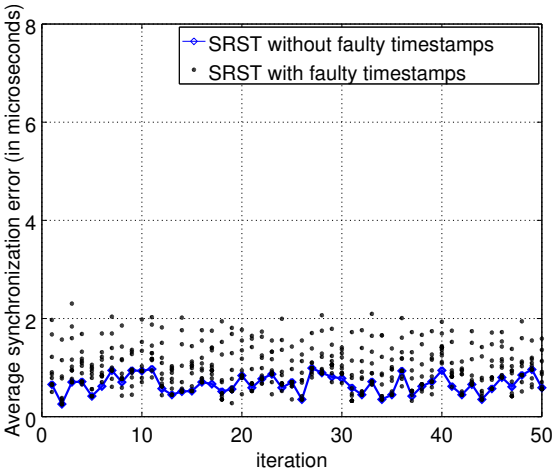
Figure 17: Detection probability of error Δ when using delay thresholds of $d^{(H_j,F_{(j,k)})}$ and $d^{(H_j,RO)}$.

Introduced error	Minimum error(μs)		Average error(μs)		Maximum error(μs)		Detection probability (%)	
	$d^{(H_j,F_{(j,k)})}$	$d^{(H_j,RO)}$	$d^{(H_j,F_{(j,k)})}$	$d^{(H_j,RO)}$	$d^{(H_j,F_{(j,k)})}$	$d^{(H_j,RO)}$	$d^{(H_j,F_{(j,k)})}$	$d^{(H_j,RO)}$
$\frac{\Delta}{2} = +1\mu s$	0	0	0.95	0.95	2	2	0	0
$\frac{\Delta}{2} = -1\mu s$	0	0	1.01	0.95	2	2	20	5.5
$\frac{\Delta}{2} = +2\mu s$	1	1	2.07	2.07	3	3	0	0.5
$\frac{\Delta}{2} = -2\mu s$	1	1	2.21	2.14	3	3	64.5	44.5
$\frac{\Delta}{2} = +3\mu s$	2	2	3.07	3.02	4	4	0	17.5
$\frac{\Delta}{2} = -3\mu s$	2	2	3.1	3.02	4	4	95	65.5
$\frac{\Delta}{2} = +4\mu s$	3	3	4.06	4.09	5	5	5	34.5
$\frac{\Delta}{2} = -4\mu s$	/	3	/	4.28	/	5	100	82.5
$\frac{\Delta}{2} = +5\mu s$	4	4	4.99	4.97	6	6	35.5	55.5
$\frac{\Delta}{2} = -5\mu s$	/	4	/	4	/	4	100	99.5
$\frac{\Delta}{2} = +6\mu s$	5	5	5.82	5.92	7	7	93	98.5
$\frac{\Delta}{2} = -6\mu s$	/	/	/	/	/	/	100	100

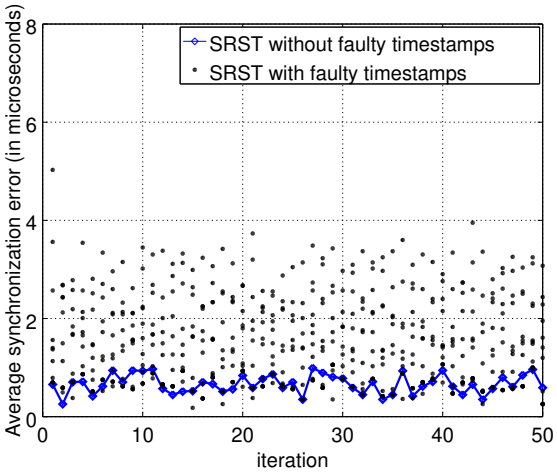
Table 6: Statistics of synchronization error in faulty timestamps settings.

7.3.2. Resilience to Faulty Timestamps

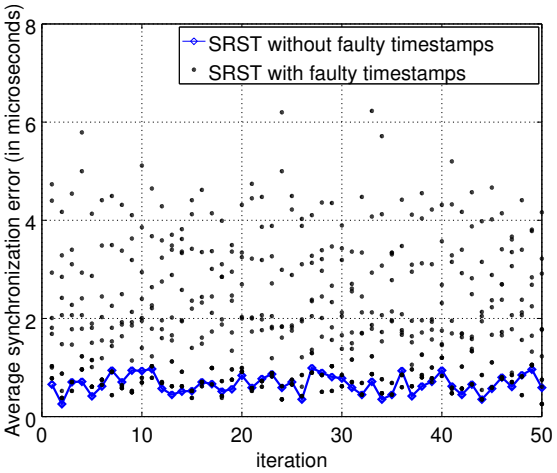
SRST demonstrates resilience to faulty timestamps by leveraging multiple timestamps received by an RO node from reference nodes. This ensures that the RO node is able to synchronize its clock as long as it can receive one correct timestamp. To evaluate the impact of faulty timestamps on synchronization accuracy, we conducted an experiment using the topology illustrated in Fig. 13, running the SRST protocol in the presence and absence of faulty timestamps. MNs were strategically selected to ensure that every node received at least one correct timestamp. These MNs were programmed to deliberately modify timestamps by adding $\pm\Delta$ to the timestamp $s_{j,k}$ before transmitting it. The introduced error Δ is set to values of 2 μs, 4 μs, 6 μs, 8 μs, 10 μs and 12 μs. As previously discussed, the delay and offset are affected by $\frac{\Delta}{2}$ and the MN cannot introduce an error exceeding $\frac{\Delta}{2} = 6$ μs. Consequently, an error of 12 μs will be fully detected with a 100 % success rate using the delay thresholds for $d^{(H_j,F_{(j,k)})}$ and $d^{(H_j,RO)}$. We applied the SRST protocol in the presence of faulty timestamps, i.e., $s^*_{j,k} = s_{j,k} \pm \Delta$, across 50 synchronization rounds. After each round, the synchronization error was computed. Fig. 18a–18d compare the average synchronization error for SRST with and without faulty timestamps as the number of MNs increases. Without faulty timestamps, the error remains consistently low, below 1 μs. In the presence of faulty timestamps, the error is slightly higher but does not exceed 6.2 μs. Fig. 18e summarizes the results, showing the maximum average synchronization error as the percentage of MNs ranges from 2% to 32%. The synchronization error gradually increases but stabilizes around 6.2 μs. It is observed that a faulty timestamp has only a slight impact of 5.5 μs on synchronization accuracy. This highlights the effectiveness of the SRST protocol and demonstrates its resilience against faulty timestamps.



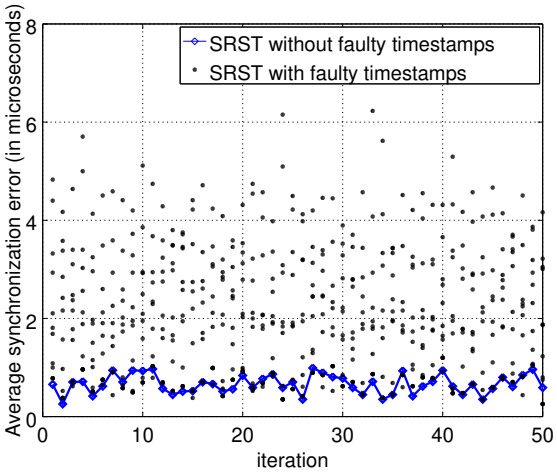
(a) One MNs.



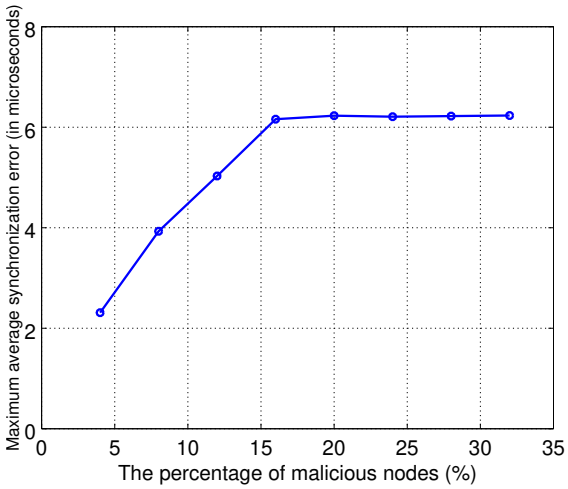
(b) Three MNs.



(c) Five MNs.



(d) Seven MNs.



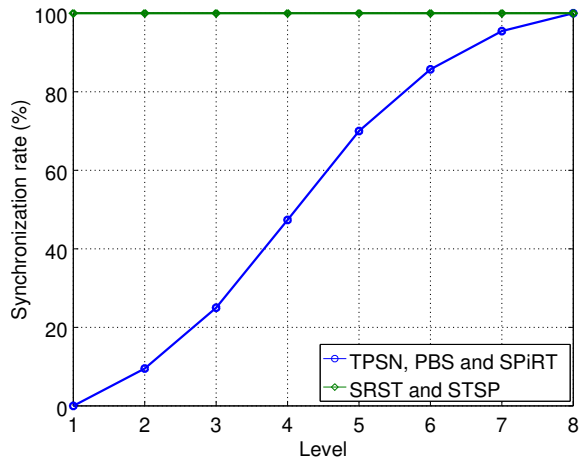
(e) Maximum average synchronization error vs rate of MNs.

Figure 18: Synchronization error when malicious nodes can exist.

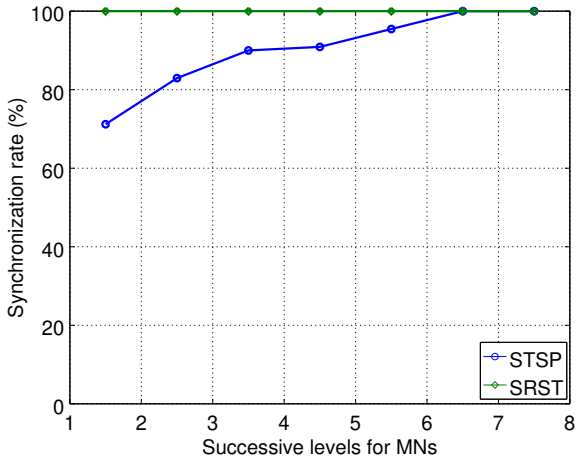
7.3.3. Synchronization Rate

We define it as the proportion of nodes that have achieved synchronization. We compare the synchronization rate of SRST, STSP, SPiRT, PBS, TPSN and FTSP protocols under varying MN counts. STSP has been proposed recently to secure SR methodology against faulty timestamps. In STSP, a node detects the faulty timestamp using a maximum time offset between its parent node and grandparent. In this experiment, the maximum time offset is computed by running STSP in the absence of faulty timestamps and is found to be equal to 12 μs. We ran SRST, STSP, SPiRT, PBS, TPSN and FTSP for the network topology in Fig. 13 where some of the nodes are MNs. In this topology, we have 8 levels where node 1 is the primary reference at level 0, nodes 2 and 3 at level 1, nodes 3, 4 and 5 at level 2, and so on. MNs are nodes that were programmed to intentionally add error $\Delta = 100$ μs to the correct timestamp before sending it out. The introduced error of 100 μs could be detected with 100% in both SRST and STSP.

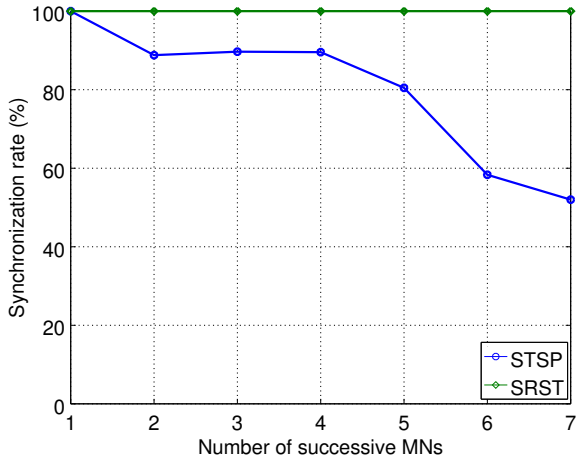
Fig. 19 shows the synchronization rate results obtained when one or multiple MNs are present. As seen in Fig. 19a, TPSN, PBS, and SPiRT are unable to synchronize nodes located beyond the level of the MNs. However, SRST and STSP successfully synchronize all nodes to the primary reference even when all nodes at a specific level are MNs. In Fig. 19b, when MNs exist in two consecutive levels, i.e., [1,2], [2,3],...,[7,8], STSP cannot synchronize the neighbors of MNs, whereas SRST can synchronize all nodes to the primary reference. As the number of successive MNs increases to 5 and 7 (Fig. 19c), the performance of STSP diminishes dramatically; yet SRST remarkably sustains its robustness and continues to synchronize all nodes. In conclusion, SRST is more effective than STSP in mitigating the impact of faulty timestamps injection attacks.



(a) level's nodes are MNs.



(b) two successive MNs.



(c) several successive MNs.

Figure 19: Synchronization rate when malicious nodes can exist.

7.4. Synchronization Latency

In this section, we compare the latency of SRST, SPiRT, PBS and TPSN protocols. We define the latency as the time required to accomplish one synchronization round in the network. We have chosen a topology with several hops, in which each node can synchronizes its clock by SRST with two or three reference nodes. Fig. 20 illustrates the latency when the hops number varies from 2 to 18. The latency grows with the increase in hop number. The latency of SRST, PBS and TPSN is almost the same; SRST takes slightly longer time. While SPiRT takes a lot of time as it applies at least two synchronization rounds to synchronize a node. Despite this slight increase, SRST is providing better accuracy and improves the security of time synchronization.

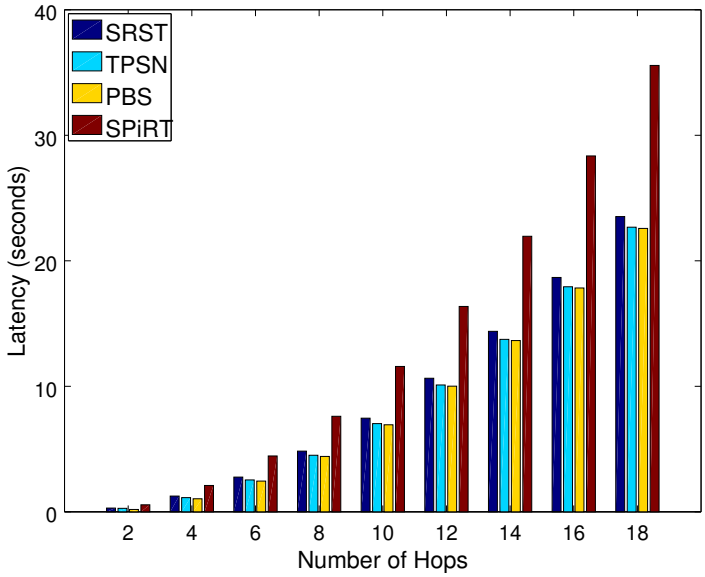


Figure 20: Latency with hop number.

8. Results and Discussion

To summarize, SRST offers a new, secure approach to time synchronization, addressing the limitations of existing methods. SRST is robust to node failure and malicious behavior, converges quickly, does not rely on specific nodes, and is topology-independent.

• *Communication overhead.* Communication over the network, particularly transmission, is one of the most energy-intensive operations. In SRST, RO nodes achieve synchronization by only overhearing timing messages. The number of timing messages required for SRST to compute clock skew and offset is analyzed and compared to those of the PBS and SPiRT approaches. In PBS and SPiRT, the RO node synchronizes to a single reference node, requiring several timing messages exchanges to compute clock skew. Moreover, extending to multi-hop time synchronization requires the formation of clusters to select leader nodes, which further increases the communication overhead. As a result, the message complexity for PBS and SPiRT increases in proportion to the number of synchronization rounds and the additional message exchanges needed for cluster formation. In contrast, the RO node in SRST can derive the joint skew/offset estimators in a synchronization round using timing messages from reference nodes, with no extra overhead required for multi-hop extension. SRST offers significant communication efficiency advantages over PBS and SPiRT. Specifically, SRST achieves 62.5 % lower communication overhead than PBS and 25 % lower than SPiRT across the tested scenarios shown in Fig. 10c.

• *Precision.* SRST addresses the physical issue of clock drift, a natural result of manufacturing imperfections and temperature variations in microcontrollers. It ensures long-term synchronization accuracy through periodic synchronization via message exchanges. The synchronization error of SRST is compared to that of the STSP, SPiRT, PBS, TPSN, and FTSP protocols. In these protocols, the propagation of timing information over multiple hops results in an accumulation of errors, reducing synchronization accuracy. In contrast, SRST synchronizes RO nodes using reference nodes within two hops, which minimizes error accumulation. Additionally, unlike consensus-based time synchronization approaches such as the ATS protocol, which averages all neighbors, the RO node in SRST synchronizes its clock with its synchronized 1-hop and 2-hop neighbors. This optimization accelerates

convergence time, enabling SRST to achieve better accuracy, under 1 μs, in a single synchronization round with only 68 message exchanges. In contrast, ATS requires 85 rounds and a total of 2125 messages to reach the same level of accuracy. Thus, SRST accelerates convergence time by a factor of 31.25 compared to the ATS protocol.

• *Security.* SRST prevents faulty timestamps caused by abnormal delays from participating in the time synchronization process. It uses two delays to detect any timestamp alterations made by MNs, as described at the end of Section 5.2. We analyzed the two delays in the absence of faulty timestamps, and the statistics are presented in Table 5. Based on the obtained results, the maximum error that can be injected by MNs is less than 6 μs. The evaluation of SRST's robustness to errors injected by MNs shows that SRST can detect an error of 6 μs with a 100 % success rate. SRST is a fault-tolerant protocol that remains robust against node failure by synchronizing the RO node with multiple reference nodes. Several factors, such as communication interference, hardware malfunctions, and malicious attacks, can lead to node failure. The RO node receives multiple timestamps from reference nodes and can successfully synchronize its clock as long as it receives at least one correct timestamp, i.e., from at least one non-malicious reference node. Thus, SRST can tolerate up to $S-1$ faulty timestamps, where S is the total number of received timestamps. Compared to STSP, a method proposed to secure the SR methodology against faulty timestamps, SRST demonstrates superior resilience to faulty timestamp injection attacks, especially in scenarios with continuous multi-hop MNs where STSP fails. The limitation of SRST is that the RO node cannot synchronize its clock if it has only one reference node and that node is malicious; nonetheless, such a scenario is extreme and cannot be handled by any RO-based approach.

9. Conclusion

In this paper, we have proposed SRST, a novel protocol for time synchronization that is secure against faulty timestamps injection attacks. SRST extends the Receiver-Only (RO) methodology to enable synchronizing RO nodes to reference nodes within two hops, enhancing robustness against node failure and malicious attacks, minimizing error accumulation over multiple hops, and optimizing convergence time. An RO node identifies reference nodes as MNs through detection of faulty timestamps using delay thresholds. The precision of the maximum likelihood estimators for clock offset and clock skew is shown to converged to the corresponding Cramer-Rao lower bounds as the number of timestamps increases. The effectiveness of SRST is validated through simulation and experiments with real devices. The protocol has been compared to state-of-the-art protocols. The results indicated that SRST achieves 62.5 % lower communication overhead than PBS protocol and 25 % lower overhead than SPiRT protocol. Furthermore, SRST outperformed existing protocols in accuracy, achieving synchronization within less than 1 μs, and accelerating convergence time by a factor of 31.25 compared to the ATS protocol. Additionally, SRST proved more effective than STSP protocol in mitigating faulty timestamp injection attacks, successfully detecting errors of 6 μs with a 100 % success rate and ensuring that the clocks of any two nodes do not deviate by more than 6 μs due to attacks.

SRST assumed relatively static node positions and known delay distributions, future work will focus on dynamic environments by incorporating dynamic threshold mechanisms based on real-time delay analysis. This will extend the protocol's applicability to mobile wireless sensor network.

References

[1] M. Jamshed, K. Ali, H. Abbasi, M. Imran, M. Ur-Rehman, Challenges, Applications, and Future of Wireless Sensors in Internet of Things: A Review, IEEE Sensors J. 22 (6) (2022) 5482–5494.

[2] K. Wójcicki, M. Bieganska, B. Paliwoda, J. Górna, Internet of Things in Industry: Research Profiling, Application, Challenges and Opportunities—A Review, Energies 15 (5) (2022) https://doi.org/10.3390/en15051806.

[3] N. Duy Tan, D. Nguyen, H. Hoang, T. Le, EEGT: Energy Efficient Grid-Based Routing Protocol in Wireless Sensor Networks for IoT Applications, Computers 103 (12) (2023) https://doi.org/10.3390/computers12050103.

[4] R. Sreedharan, Applications of wireless sensor networks in IoT, Intelligent Wireless Sensor Networks and the Internet of Things: Algorithms, Methodologies, and Applications (2024) 270.

[5] J. Elson, D. Estrin Birman, Fine-grained Network Time Synchronization using Reference Broadcast, In Proc. of the 5th Symp. on Oper. Syst. Design and Implementation (OSDI'02) (2002) 147–163.

[6] D. Djenouri, N. Merabtine, F. Z. Mekahlia, M. Doudou, Fast Distributed Multi-hop Relative Time Synchronization Protocol and Estimators for Wireless Sensor Networks, Ad Hoc Networks 11 (8) (2013) 2329–2344.

[7] S. Ganeriwawal, R. Kumar, M. Srivastava, Timing-Sync Protocol for Sensor Networks, In Proc. of the 1st ACM Conf. on Embedded Networked Sensor Systems (SenSys) (2003) 138–149.

[8] M. Maróti, B. Kusy, G. Simon, A. Lédezi, The Flooding Synchronization Protocol, In Proc. of the 2nd ACM Conf. on Embedded Networked Sensor Systems (SenSys'04) (2004) 39–49.

[9] K. L. Noh, E. Serpedin, K. Qaraqe, A New Approach for Time Synchronization in Wireless Sensor Networks: Pairwise Broadcast Synchronization, IEEE Trans. Wireless Comm. 9 (2008) 3318–3322.

[10] C. Benzaid, M. Bagaa, M. Younis, Efficient Clock Synchronization for Clustered Wireless Sensor Networks, Ad Hoc Network 56 (2017) 13 – 27.

[11] K. L. Noh, E. Serpedin, K. Qaraqe, Extension of Pairwise Broadcast Clock Synchronization for Multicluster Sensor Networks, EURASIP J. on Advances in Signal Processing 2008 (71) (2008) 10.

[12] G. Liu, S. Yan, L. Mao, Receiver-Only-Based Time Synchronization Under Exponential Delays in Underwater Wireless Sensor Networks, IEEE Internet of Things Journal 7 (10) (2020) 9995–10009.

[13] J. Wu, L. Jiao, R. Ding, Average Time Synchronization in Wireless Sensor Networks by Pairwise Messages, Journal of Computer Communications (2012) 221–233.

[14] H. Wang, D. Xiong, L. Chen, P. Wang, A Consensus-Based Time Synchronization Scheme With Low Overhead for Clustered Wireless Sensor Networks, IEEE Signal Processing Letters 25 (2018) 1206–1210.

[15] J. Wu, L. Zhang, Y. Bai, Y. Sun, Cluster-based Consensus Time Synchronization for Wireless Sensor Networks, IEEE Sensors Journal 15 (3) (2015) 1404–1413.

[16] F. Shi, X. Tuo, L. Ran, S. Ren, Z.and Yang, Fast Convergence Time Synchronization in wireless Sensor Networks based on Average Consensus, IEEE Trans. Ind. Inf. 16 (2) (2020) 1120–1129.

[17] L. Phan, T. Kim, Fast Consensus-Based Time Synchronization Protocol Using Virtual Topology for Wireless Sensor Networks, IEEE Internet of Things Journal 8 (9) (2021) 7485–7496.

[18] F. Shi, S. Yang, M. Mukherjee, H. Jiang, D. Costa, W. Wong, Parameter-Sharing-Based Average-Consensus Time Synchronization in IoT Networks, IEEE Internet of Things Journal 10 (9) (2023) 8215–8227.

[19] L. Phan, T. Kim, Hybrid Time Synchronization Protocol for Large-scale Wireless Sensor Networks, Journal of King Saud University - Computer and Information Sciences 34 (10) (2022) 10423–10433.

[20] Y. Koo, M. Mahyuddin, M. Mukherjee, M. Ab Wahab, Novel Control Theoretic Consensus-Based Time Synchronization Algorithm for WSN in Industrial Applications: Convergence Analysis and Performance Characterization, IEEE Sensors Journal 23 (4) (2023) 4159–4175.

[21] Z. Wang, T. Yong, X. Song, Fast and Low-Overhead Time Synchronization for Industrial Wireless Sensor Networks with Mesh-Star Architecture, Sensors 23 (8) (2023) https://doi.org/10.3390/s23083792.

[22] H. Wang, Y. Zou, X. Liu, M. Li, A Fast Convergence Scheme for Distributed Consensus Time Synchronization Using Multi-Hop Virtual Links in Industrial Wireless Sensor Networks, IEEE Sensors Journal (2024) doi:10.1109/JSEN.2024.3394051.

[23] Y. Wang, Y. Zhang, A Survey of Secure Time Synchronization, Applied science Journal 13 (6) (2023) https://doi.org/10.3390/app13063923.

[24] S. Ganeriwal, S. Capkun, M. Srivastava, Secure time synchronization in sensor networks, ACM Trans. on Information and Syst. Security 11 (4) (2008) 1–35.

[25] Y. Z. K. Sun, P. Ning C. Wang A. Liu, Tinysersync: Secure and resilient time synchronization in wireless sensor networks, In Proc. of the 13th ACM Conf. on Computer and Commun. Security (CCS'06) (2006) 36–42.

[26] C. Benzaid, A. Saiah, N. Badache, An Enhanced Secure Pairwise Broadcast Time Synchronization Protocol in Wireless Sensor Networks, Parallel, Distributed and Network-Based Processing (PDP) (Feb. 2014).

[27] Y. Wei, W. Yadong, W. Qin, L. Chong, Security Countermeasures for Time Synchronization in IEEE802.15.4e-Based Industrial IoT, Journal of Computer Research and Development 54 (9) (2017) 2032–2043.

[28] T. Qiu, X. Liu, M. Liu, H. Ning, D. Wu, A Secure Time Synchronization Protocol Against Fake Timestamps for Large-Scale Internet of Things, IEEE Internet of Things 4 (6) (2017) 1879–1889.

[29] S. Jha, A. Gupta, N. Panigrahi, Security Threat Analysis and Countermeasures on Consensus-Based Time Synchronization Algorithms for Wireless Sensor Network, SN Computer Science 2 (409) (2021) https://doi.org/10.1007/s42979–021–00796–1.

[30] H. Song, S. Zhu, G. Cao, Attack-Resilient Time Synchronization for Wireless Sensor Networks, Ad Hoc Networks Journal 5 (1) (2007) 112–125.

[31] P. Jia, X. Wang, K. Zehng, Distributed Clock Synchronization Based on Intelligent Clustering in Local Area Industrial IoT Systems, IEEE Transactions on Industrial Informatics 16 (6) (2020) 3697–3707.

[32] L. Phan, T. Kim, T. Kim, Robust Neighbor-Aware Time Synchronization Protocol for Wireless Sensor Network in Dynamic and Hostile Environments, IEEE Internet of Things Journal 8 (3) (2021) 1934–1945.

[33] Y. Jeon, T. Kim, T. Kim, Fast and Robust Time Synchronization with Median Kalman Filtering for Mobile Ad-Hoc Networks, Sensors Journal 21 (2) (2021) https://doi.org/10.3390/s21020590.

[34] S. Jha, A. Gupta, N. Panigrah, Resilient Consensus-Based Time Synchronization with Distributed Sybil Attack Detection Strategy for Wireless Sensor Networks: A Graph Theoretic Approach, International Journal of Computer Networks and Applications 10 (1) (February 2023).

[35] Z. Wang, D. Sun, C. Yu, Reference Broadcast-Based Secure Time Synchronization for Industrial Wireless Sensor Networks, Applied Science 13 (16) (2023) https://doi.org/10.3390/app13169223.

[36] A. Perrig, R. Szewczyk, V. Wen, D. Culler, J. D. Tygar, SPINS: Security Protocols for Sensor Networks, in Proceedings of the 7th Annual International Conference on Mobile Computing and Networking (Mobicom '01) (2001) 189–199.

[37] K. Fan, S. Wang, Y. Ren, K. Yang, Z. Yan, H. Li, Y. Yang, Blockchain-based Secure Time Protection Scheme in IoT, IEEE Internet of Things 3 (6) (2018) 4671–4679.

[38] K. Fan, Z. Shi, R. Su, Y. Bai, P. Huang, K. Zhang, H. Li, Y. Yang, Blockchain-Based Trust Management for Verifiable Time Synchronization Service in IoT, Peer-to-Peer Networking and Applications https://doi.org/10.1007/s12083-021-01276-2 (2022) 1152–1162.

[39] C. Benzaid, A. Saiah, N. Badache, Secure Pairwise Broadcast Time Synchronization in Wireless Sensor Networks, In Proc. of the 7th IEEE International Conference on Distributed Computing in Sensor Systems (DCOSS 2011) (2011) 1–6.

[40] M. Jadliwala, Q. Duan, S. Upadhyaya, J. Xu, Towards a Theory for Securing Time synchronization in Wireless Sensor Networks, In Proc. of the 2nd ACM Conf. on Wireless Network Security (WiSec'09) (2009) 201–212.