This full text paper was peer reviewed at the direction of IEEE Communications Society subject matter experts for publication in the IEEE INFOCOM 2008 proceedings.

Attack-Tolerant Time-Synchronization in Wireless Sensor Networks

Xin Hu Taejoon Park Kang G. Shin
Real-Time Computing Laboratory, EECS Department
The University of Michigan, Ann Arbor, MI 48109-2121, U.S.A.

***Abstract*—Achieving secure time-synchronization in wireless sensor networks (WSNs) is a challenging, but very important problem that has not yet been addressed effectively. This paper proposes an *Attack-tolerant Time-Synchronization Protocol* (ATSP) in which sensor nodes cooperate to safeguard the time-synchronization service against malicious attacks. ATSP exploits the high temporal correlation existing among adjacent nodes in a WSN to achieve (1) adaptive management of the profile of each sensor's normal behavior, (2) distributed, cooperative detection of falsified clock values advertised by attackers or compromised nodes, and (3) significant improvement of synchronization accuracy and stability by effectively compensating the clock drifts with the *calibrated clock*. To reduce the risk of losing time-synchronization due to attacks on the reference node, ATSP utilizes *distributed, mutual synchronization* and confines the impact of attacks to a local area (where attacks took place). Furthermore, by maintaining an accurate profile of sensors' normal synchronization behaviors, ATSP detects various critical attacks while incurring only reasonable communication and computation overheads, making ATSP attack-tolerant and ideal for resource-constrained WSNs.**

I. Introduction

Recently, wireless sensor networks (WSNs) have been attracting considerable attention due mainly to the increasing number of their applications in military, environmental, medical, and industrial domains. A WSN usually consists of a large number of densely-deployed, small, low-cost and battery-powered sensor devices. Sensor nodes are often required to be time-synchronized for various applications, such as trajectory estimation of moving objects, public infrastructure surveillance, and data fusion. Without a common time base among sensors, data with inaccurate timestamps would be collected and aggregated, causing large estimation errors and incorrect scheduling of events/actions. To overcome these problems, many time-synchronization algorithms and prototypes have been proposed for WSNs [1–5]. Despite their high synchronization accuracy, they did not address the problem of protecting the time-synchronization service from node failures and malicious attacks, because they all assume a benign environment in which every sensor node behaves normally and follows the underlying protocol faithfully. However, WSNs, often deployed in an inaccessible, hostile environment, are characterized by their large-scale and unattended nature that is likely to invite many critical security attacks. Therefore, security has been identified as a major challenge for WSNs in general, and for the time-synchronization service in particular.

Many existing secure time-synchronization protocols require one or more centralized reference nodes and identify possible attacks with a heuristic threshold of propagation delay [6] or clock offset [7]. However, the reliance on reference nodes often implies susceptibility to single points of failure and violates the distributed nature of WSNs, because the failures of the reference nodes will disrupt the synchronization service in a large portion, if not the entirety, of a WSN.

In this paper, we take a distributed and mutual synchronization approach to building an *Attack-tolerant Time-Synchronization Protocol* (ATSP) for WSNs under which sensors cooperatively execute and safeguard the time-synchronization service. The rationale behind this is that a sensor network inherently relies on collective assurance among clusters of sensors to execute complex applications even in the presence of attacks. ATSP is essentially a cooperative intrusion-detection system (IDS) where sensors achieve a high level of attack-tolerance by adaptively building and tracking the normal profile of synchronization behaviors of their neighbor nodes. Any information that deviates noticeably from the normal profile is rejected, thereby forcing the attacker to weaken the attack strength so as not to be caught. However, due to the use of distributed synchronization where each sensor adjusts its own clock based on multiple neighbors' clock values, the small perturbation by the attacker(s) (that is below the detection threshold) will be smoothed out, thus causing much smaller synchronization errors than those in the centralized model. To compensate the clock drift in-between adjacent synchronization points, ATSP utilizes "calibrated clock" by which each sensor uses its neighbors' normal profiles to estimate their clock drift without additional communication. Our security analysis and simulation results demonstrate the effectiveness and robustness of ATSP in defeating critical attacks with reasonable processing and communication overheads for resource-constrained WSNs.

The rest of the paper is organized as follows. Section II discuses related work while Section III describes system models. Section IV presents our proposed approach while Section V analyzes ATSP's attack detection. Section VI details the ATSP protocol and our evaluation results are presented in Section VII. Finally, Section VIII concludes the paper.

II. Related Work

Time-synchronization has been studied extensively in distributed systems and wired networks [8]. However, they either assume unlimited computing resources or require expensive devices and thus are unsuitable for resource-constrained WSNs. Recently, there have been many time-synchronization protocols tailored to WSNs. The Reference Broadcast Synchronization (RBS) [1] intended for pair-wise and multi-domain clock synchronization seeks to reduce the unpredictable latency by exploiting the broadcast nature of the wireless communication and shortening the critical path. Palchaudhuri *et al.* [9] extended RBS and applied probabilistic clock synchronization to guarantee an upper bound on the accuracy. Ganeriwal *et al.* [2] proposed a network-wide synchronization scheme called TPSN which uses MAC-layer timestamping to reduce the non-deterministic message delay. The protocol achieves a global timescale by establishing a hierarchical structure and having each sensor node synchronize with its

978-1-4244-2026-1/08/$25.00

root. The Flooding Time-Synchronization Protocol (FTSP) [3] extended the MAC-layer timestamping to further eliminate the uncertainties in message transmission so that one single broadcast message is sufficient to synchronize the sender and the receiver. More recently, Su and Akyildiz [4] proposed the Time Diffusion Protocol (TDP) which achieves a network-wide equilibrium time based on the diffusion of messages, thereby involving all the nodes in the synchronization process. Li and Rus [10] also proposed a fully localized diffusion based method to achieve global time synchronization. Other algorithms (e.g., [5]) have also been proposed to reduce computation and storage complexities.

However, the above protocols all assume benign environments and cannot survive malicious attacks [11]. To meet the security requirements, several secure synchronization schemes [6, 7, 12, 13] have also been proposed. [6] secures pairwise and group time-synchronization by checking if the end-to-end delays exceed a pre-defined threshold and aborting the service upon detection of an attack, which can potentially be exploited by DoS attacks. Algorithms proposed in [7] detect and accommodate message-delay attacks via outlier- and threshold-based technique. [13] presents a fault-tolerant scheme that is proven to guarantee an upper bound of clock skew between non-faulty nodes in the network. Although these protocols provide high security guarantee against various attacks, many of them are based on the reference-based model and the hierarchical structure and thus suffer from a single point of failure. Therefore, an inherent distributed time-synchronization protocol—where the failure of any node affects nodes only in its proximity—is more desirable for WSNs.

ATSP differs from previous work in several ways. First, it safeguards the time-synchronization service by adopting an anomaly-detection approach, in which sensors monitor each other's behavior, and cooperatively achieve accurate detection of attacks. Second, ATSP takes a distributed approach where each sensor independently makes decisions based only on the information collected from multiple adjacent nodes, thus achieving a high level of resistence to various attacks.

III. System Models

A. Network Model

The WSN under consideration is a dense network consisting of a large number of resource-constrained sensor nodes with neither reference nodes nor a root node. This is a realistic deployment scenario in that a WSN is inherently infrastructure-less where many sensors autonomously organize themselves into a connected structure. Thus, it is often desirable to minimize the dependency of time-synchronization on infrastructure nodes. In addition, each node maintains a sufficient number of neighbors to accelerate the synchronization process. The number of neighbors can be easily adjusted by changing the transmission power when the synchronization information is broadcast. Note the bidirectional neighbor relationship is not needed in ATSP. For further reduction of the synchronization overhead, each node piggybacks the synchronization information on beacon messages that are periodically broadcast to refresh each node's neighbor list. In the current work, we assume there exist some reliable broadcast technique such as the method proposed in [14].

B. Attack Model

Since WSNs usually operate in an unattended environment, they are subjected to a variety of attacks, such as (1) physical attacks, e.g., destroying, replacing and/or cloning sensors; (2) data attacks, e.g., capturing, replaying and spoofing of packets; (3) resource-consumption and DoS attacks; and (4) sybil attacks. This paper focuses on attacks targeting the time-synchronization service, including:

- **Node compromises**: an attacker compromises and abuses a sensor node to mislead its neighbors with incorrect timing information. For example, since TPSN [2] constructs a spanning tree to propagate the synchronization service, compromising one sensor node suffices to mislead all the sensor nodes in the subtree below the node.
- **Message manipulation**: includes dropping, modifying and faking synchronization messages. For example, an attacker may fabricate messages from the reference node and/or replay an old synchronization message, causing other nodes to adjust to a wrong time.
- **Message delay**: an attacker can disrupt the time-synchronization by simply delaying messages. A remote node's clock is often inferred by exchanging messages. The message-delivery delay has been considered negligible (e.g., in FTSP) or the same for all receiving nodes (e.g., in RBS and TPSN). However, this condition can easily be broken by intercepting the message and replaying it after a unpredictable delay. This extra delay will be incorporated into the clock offsets between sensor nodes, effectively skewing the clock values between nodes.

C. Clock Model

Each sensor node has its own physical clock, calculated by counting pulses of its hardware oscillator running at a particular frequency. In practice, sensors' oscillators run at slightly different frequencies and the frequency varies unpredictably, depending on ambient factors such as temperature and humidity. Hence, sensors' clocks are always subject to a divergence or clock skew. According to [15], for a relatively extended period of time (minutes to hours), the clock can be approximated with good accuracy by an oscillator of fixed frequency. The local clock of a sensor node i can thus be approximated [16] as $C_i(t) = \alpha_i t + \beta_i$, where t is the physical time, α_i is the drift rate of i, and β_i is the offset between the local clock and the physical time.

IV. The Proposed Approach

Since it is practically impossible to completely prevent all possible attacks in a WSN, we would like to make the network attack-tolerant so that the impact of malicious attacks and node failures could be tolerated and confined within the local area. Next, we will detail the ATSP architecture, the underlying synchronization and profile-building algorithms.

A. Motivation

To maximize attack-tolerance and time-synchronization accuracy, ATSP exploits the temporal correlation among neighboring sensors' clocks and takes an *anomaly-detection* approach to detect misbehaving sensors in synchronization process. Essentially, every node maintains and dynamically updates a normal profile for each of its neighbors based

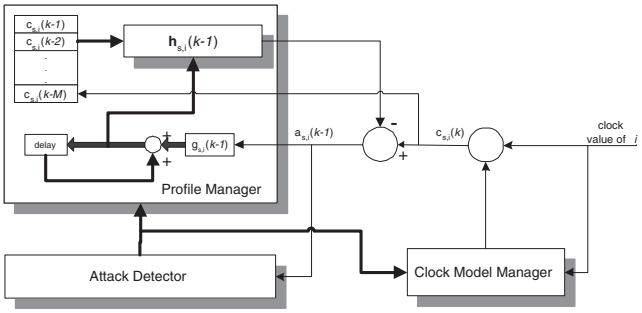

Fig. 1. The ATSP architecture in which a sensor node s processes the time announcements from its neighbor i

on their past behavior. Upon receiving a neighbor's time announcement, each sensor compares the announcement with the neighbor's expected normal behavior estimated from the profile. This way, a noticeable deviation from the normal profile will make the node suspected of having been compromised, thanks to the strong temporal correlation existing between neighbor nodes Hence, the adversary has to lower its attack strength so as not to be caught. However, because ATSP is a distributed protocol where each sensor adjusts its own clock based on inputs from a number of neighbors, false time announcements with small perturbations could easily be smoothed out, if a majority of the sensors are well-behaving. In other words, in ATSP well-behaving sensors can cooperatively identify and remove/blacklist the misbehaving sensors.

B. ATSP Architecture

ATSP consists of the following three modules (Fig. 1):

- **Clock model manager** establishes and maintains the relative clock models between a sensor node and each of its neighbors. The clock models are used to build an accurate profile of a neighbor's behavior.
- **Profile manager** constructs and maintains a compact profile of normal time-synchronization behavior.
- **Attack detector** identifies suspicious time announcements and feeds the results back to the other two modules to update the normal profiles at run-time.

These three modules closely interact with each other and cooperatively form an intrusion detection system with which sensor nodes check the validity of time announcements based on their normal profile. Instead of centralized reference nodes, ATSP uses peer sensor nodes as active information sources for updating and validating clock values, thus achieving high synchronization accuracy and the attack-detection capability.

C. The Synchronization Algorithm

ATSP is based on the Interactive Convergence Time Synchronization (ICTS) algorithm similar to the one in [17]. In ICTS, the network-wide synchronization is achieved by having each node first derive the time offsets between itself and all of its neighbors by exchanging messages. Each node then computes the average of the measured clock skews and uses it to adjust its own clock. As long as less than one third (half) of neighbor nodes are mis-behaving with Byzantine (non-Byzantine) faults, all the sensor nodes in the neighborhood will establish a common equilibrium time.

1) Calculation of Time Offsets: ATSP adopts the single message broadcast method used in FTSP [3] to compute the offset between two nodes. FTSP successfully eliminates major sources of uncertainties in the packet transmission (i.e., transmission time, access time, reception time, jitter of interrupt-handling and encoding/decoding time) by performing MAC-layer timestamping multiple times for every message at each byte boundary and embeds a final error-corrected and averaged timestamp into the message. The only uncertainty is the propagation time (for packets to traverse the wireless link) which is often very small and can be safely ignored. According to [3], using only 6 timestamps per message, FTSP achieves the timestamping accuracy of 1.4 μs on the Mica2 platform. Thus, one radio broadcast is sufficient for all the neighbors to accurately calculate the time offsets between their clocks and sender's clocks, each of which is simply the difference between transmission and reception timestamps. .

2) Details of ICTS: Let s be the sensor node performing time-synchronization and n_s is the number of s's neighbors. T_i and T_s represent the send and receive timestamps. Node s can then calculate the time offset between itself and node i as $\Delta_{s,i} = T_s - T_i$. After obtaining the time offsets $\Delta_{s,i}$ for $i = 1, \cdots, n_s$ from all of its neighbors, s computes its new clock value at time t or $C'_s(t)$ as:

$$C'_s(t) = C_s(t) + \frac{1}{n_s + 1} \sum_{i=1}^{n_s} \Delta_{s,i}. \quad (1)$$

The denominator $n_s + 1$ comes from the fact that node s's own clock is also considered for the computation of a new clock value.

Sensors terminate the initial synchronization when the local clock gets stabilized (i.e., $|C'_s(t) - C_s(t)| < \epsilon$, ϵ is a predefined parameter determining the synchronization accuracy). However, synchronization at a single point is insufficient, as the discrepancies in the clock drift rates of different sensors will cause nodes to go out of synchronization after a short period of time. To maintain an acceptable accuracy, it is necessary to periodically execute ICTS for resynchronization. The appropriate resynchronization interval can be determined by the bound of the time skew and the maximum relative drift rate among sensor nodes [18].

D. Relative Clock Models

Under ATSP, every node maintains and continuously updates the relative clock model for each of its neighbors. Maintaining relative clock models facilitates: (1) construction of the normal profile of a neighbor's behavior and (2) compensation for the clock drift in-between adjacent synchronization points.

The relative drift rate between the local clocks of nodes s and i is defined as $\alpha_{s,i} = \alpha_i/\alpha_s$. Since there is no reference to the physical time, a node's drift rate (e.g., α_i or α_s) cannot be directly measured. ATSP derives the relative drift rate indirectly as follows. Assume each sensor performs

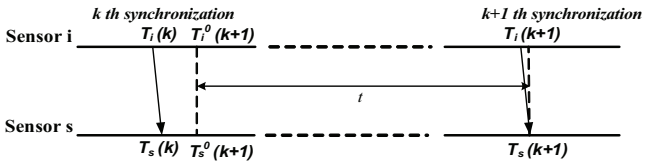

Fig. 2. Estimation of the relative drift rate between sensor nodes s and i

synchronization periodically. In Fig. 2, $T_s(k)$ and $T_i(k)$ are the MAC-layer timestamps that record the send and the receive times of the synchronization message at the k-th iteration where k = 1, 2, $\cdots$. Let $t_s(k)$ and $t_i(k)$ denote the physical times corresponding to $T_s(k)$ and $T_i(k)$. Assuming that s finishes its k-th synchronization at physical time $t_s^0(k+1)$. $T_s^0(k+1)$ and $T_i^0(k+1)$ represent the readings of local clocks of node s and i for physical time $t_s^0(k+1)$, after they complete the k-th synchronization. t is the time between s's completion of the k-th iteration and s's reception of the synchronization message from i at the $(k+1)$-th iteration. Clearly, $t = t_s(k+1) - t_s^0(k+1)$. We have two observations:

1) $T_i^0(k+1) \simeq T_s^0(k+1)$, since $T_i^0(k+1)$ and $T_s^0(k+1)$ are s's and i's local clock for the same physical time $t_s^0(t+1)$, when they are synchronized at the k-th iteration.
2) $t_s(k+1) \simeq t_i(k+1)$, because $t_i(k+1)$ is the physical time when node i sends a message, $t_s(k+1)$ represents the time when node s receives it and the propagation delay can be ignored. However, their corresponding local times, $T_s(k+1)$ and $T_i(k+1)$, are different (i.e., s and i are out of synchronization).

We can then derive $\alpha_{s,i}(k)$. First, if expressed in terms of s's local clock, the time interval t is:

$$\begin{aligned} &T_s(k+1) - T_s^0(k+1) \\ &= \alpha_s t_s(k+1) + \beta_s(k+1) - \alpha_s t_s^0(k+1) - \beta_s(k+1) \\ &= \alpha_s(t_s(k+1) - t_s^0(k+1)) = \alpha_s t. \end{aligned}$$

Likewise, if t is expressed in terms of i's local clock, the following equation holds:

$$\begin{aligned} &T_i(k+1) - T_i^0(k+1) \\ &= \alpha_i t_i(k+1) + \beta_i(k+1) - \alpha_i t_s^0(k+1) - \beta_i(k+1) \\ &= \alpha_i(t_i(k+1) - t_s^0(k+1)) \simeq \alpha_i(t_s(k+1) - t_s^0(k+1)) = \alpha_i t. \end{aligned}$$

Then, the relative drift rate $\alpha_{s,i}$ can be calculated without knowing the physical time interval t:

$$\alpha_{s,i}(k) = \frac{\alpha_i}{\alpha_s} = \frac{\alpha_i t}{\alpha_s t} = \frac{T_i(k+1) - T_i^0(k+1)}{T_s(k+1) - T_s^0(k+1)}. \quad (2)$$

Since $T_i(k+1) = \Delta_{s,i} + T_s(k+1)$ and $T_i^0(k+1) \simeq T_s^0(k+1)$ (Observation 1), the above equation can be simplified as:

$$\begin{aligned} \alpha_{s,i}(k) &\simeq \frac{\Delta_{s,i} + T_s(k+1) - T_s^0(k+1)}{T_s(k+1) - T_s^0(k+1)} \\ &= 1 + \frac{\Delta_{s,i}}{T_s(k+1) - T_s^0(k+1)}. \end{aligned} \quad (3)$$

E. Construction of Normal Profiles

Since the clock drift is an inherent characteristic of a sensor's clock with good short-term stability [3], node s's observation of a neighbor node i's synchronization behavior can be fully captured by the relative drift rates collected at each iteration using Eq. (3). Thus, we define the Profile Element (PE) of node s's profile for its neighbor i at iteration k to be:

$$c_{s,i}(k) = \frac{\Delta_{s,i}}{T_s(k+1) - T_s^0(k+1)}. \quad (4)$$

At the k-th iteration, s has processed k PEs for node i, i.e., $\{c_{s,i}(k)\}_{t=1}^{k}$. Since only recent values are useful, we define $m \times 1$ profile vector consisting of m most recent PE records: $\mathbf{c}_{s,i}(k;m) = [\, c_{s,i}(k), \cdots, c_{s,i}(k-m+1)\,]^T$ where $1 \le i \le n_s$ and $1 \le m \le k$. $c_{s,i}(k)$'s exhibit a strong temporal correlation, as they represent the quality of neighbors' clocks and are updated at each iteration. Hence, a compact description of the normal profile can be derived to accurately reflect node s's expectation of its neighbors' behaviors. If the observed behaviors deviate significantly from the expectation, s should suspect the existence of some anomalies, such as node failure or malicious attacks.

F. Problem Formulation

Our problem is cast into the design of a bank of n_s adaptive transversal filters [19] for node s, each with M taps that bookkeep the synchronization history for all s's neighbors for the past M iterations. Then, for each of n_s neighbors, the M history data are used to predict the output value at the current iteration. First, we formulate the *least squares prediction* problem as follows:

$$\begin{aligned} \hat{c}_{s,1}(t) &= \mathbf{h}_{s,1}^T(k)\, \mathbf{c}_{s,1}(t-1;M) \\ &\cdots \qquad\qquad M+1 \le t \le k \\ \hat{c}_{s,n_s}(t) &= \mathbf{h}_{s,n_s}^T(k)\, \mathbf{c}_{s,n_s}(t-1;M) \end{aligned} \quad (5)$$

where $\mathbf{h}_{s,i}(k) = [\, h_{s,i}^1(k), h_{s,i}^2(k), \cdots, h_{s,i}^M(k)\,]^T$ is the $M \times 1$ filter-weight vector for neighbor i at iteration k and $\mathbf{c}_{s,i}(t-1;M)$ is the $M \times 1$ past profile vector for i: $\mathbf{c}_{s,i}(t-1;M) = [\, c_{s,i}(t-1), \cdots, c_{s,i}(t-M)\,]^T$. which are used to estimate the current PE value. Hence, we want to find estimators (filter-weight vectors) $\{\, \mathbf{h}_{s,i}(k)\}_{i=1}^{n_s}$ at iteration k that minimize the sum of squared errors (SSE):

$$SSE_{s,i}(k) = \sum_{t=M+1}^{k} \lambda^{k-t} |\, c_{s,i}(t) - \mathbf{h}_{s,i}^T(k)\mathbf{c}_{s,i}(t-1;M)\,|^2 \quad (6)$$

where λ (≤ 1) is an exponential forgetting factor that limits the number of input samples.

G. The Recursive Update Algorithm

We apply the recursive least squares (RLS) [19] to develop an algorithm that updates the filter-weight vectors $\{\, \mathbf{h}_{s,i}(k)\}_{i=1}^{n_s}$ upon receiving a new synchronization message at the $(k+1)$-th iteration. The purpose is to find the filter-weight vector that produces the RLS of the prediction errors. Given $\{\mathbf{h}_{s,i}(k-1)\}_{i=1}^{n_s}$ where $k \ge M+1$, the RLS algorithm first calculates, for each i, *a priori* prediction error, $e_{s,i}(k)$, based on the previous estimator vector at iteration k:

$$e_{s,i}(k) = c_{s,i}(k) - \mathbf{h}_{s,i}^T(k-1)\mathbf{c}_{s,i}(k-1;M). \quad (7)$$

The filter-weight vector is updated as:

$$\hat{\mathbf{h}}_{s,i}(k) = \mathbf{h}_{s,i}(k-1) + e_{s,i}(k)\, \mathbf{g}_{s,i}(k) \quad (8)$$

where $\mathbf{h}_{s,i}(M) = \mathbf{0}$ and $\mathbf{g}_{s,i}(k)$ is an $M \times 1$ gain vector:

$$\mathbf{g}_{s,i}(k) = \frac{\mathbf{P}_{s,i}(k-1)\mathbf{c}_{s,i}(k-1;M)}{\lambda + \mathbf{c}_{s,i}^T(k-1;M)\mathbf{P}_{s,i}(k-1)\mathbf{c}_{s,i}(k-1;M)} \quad (9)$$

$\mathbf{P}_{s,i}(k)$ is an $M \times M$ inverse correlation matrix, initialized to $\mathbf{P}_{s,i}(M) = \rho^{-1}\mathbf{I}$ with a small positive ρ, and updated by

$$\begin{aligned} \mathbf{P}_{s,i}(k) &= \lambda^{-1}\mathbf{P}_{s,i}(k-1) \\ &- \lambda^{-1}\mathbf{g}_{s,i}(k)\mathbf{c}_{s,i}^T(k-1;M)\, \mathbf{P}_{s,i}(k-1). \end{aligned} \quad (10)$$

RLS achieves the optimum fit between the estimation and real measurements by recursively updating the filter-weight vectors to minimize the sum of squares of prediction errors

[19]. The benefits of using the RLS algorithm is that it doesn't require inversion of large matrices, hence being very efficient and reducing the computational requirement.

H. Profile Manger

Fig. 1 shows the architecture of the profile manager constructed based on the RLS algorithm. At the k-th iteration, the profile manager of s includes n_s RLS filters, each storing M past profile elements, i.e., $\mathbf{c}_{s,i}(k;M)$ and M filter-weights, $\mathbf{h}_{s,i}(k)$, that are adaptively and recursively updated using Eqs. (7)–(10) starting from $k = M+1$. With this architecture, the entire synchronization history of node s up to the k-th iteration is fully captured in n_s $M \times 1$ vectors, that constitute a normal profile for all of s's neighbors at iteration k. The computational cost at each iteration is $\mathcal{O}(n_s \cdot M^2)$ and the storage requirement is $\mathcal{O}(n_s \cdot M)$. To determine an optimal value of M, one may use the Akaike Information Criterion (AIC) or the minimum description length criterion [19]. In ATSP, our simulation results (Section VII) show that a small value of M (4 or 5) suffices to achieve high prediction accuracy, thanks to the high temporal correlation between neighboring nodes.

V. Attack Detection

We now describe the design of an attack detector and then qualitatively analyze its accuracy.

A. Detection Algorithm

A compromised node i may attempt to have a falsified clock value accepted by another node s so that $\Delta_{s,i}$ can be boosted to a large value. This may, in turn, cause the clock adjustment to have a large deviation from its desired value (Eq. (1)), making it impossible, or at least take a very long time, for all sensor nodes' clocks to converge. Hence, each node must verify the trustworthiness of synchronization messages it receives from its neighbors before using them to adjust its own clock values. We, therefore, propose an incremental detection scheme activated at iteration $M+1$ (first M iterations are used for establishing normal profiles), at which every node compares the new time announcements with the expected value estimated from the normal profile during the previous iteration. Clearly, the prediction error (Eq. (7)) $e_{s,i}(k)$, $1 \le i \le n_s$ quantifies the deviation of node i's new announcement from the value predicted from the most-recent profiles and hence, s should suspect i to be compromised or misbehaving, if $e_{s,i}(k)$ exceeds a certain threshold. Accordingly, for each node i, node s decides the time announcement to be disruptive to the time-synchronization service if

$$|e_{s,i}(k)| \ge \max\{\eta_t \cdot c_{s,i}(k), e_{\min}\} \tag{11}$$

where η_t (≤ 1) is a network-wide propositional threshold for detecting anomalies. e_{min} is the minimum error threshold, below which the prediction error is negligible.

Possible attacks during the first M iterations of the synchronization process can be detected easily, The profile manager of s initially keeps populating its buffer with PE values and then activates the prediction mechanism at the $(M+1)$-th iteration when the buffer is filled. This means the malicious neighbor i cannot arbitrarily falsify its announcements (from iteration $M+1$ on) because, the announced timestamp must be consistent with $\hat{c}_{s,i}(M+1)$. Consequently, if i made false announcements during the first M iterations, it has to keep misbehaving (i.e., announcing false timestamps deliberately chosen to evade Eq. (11)) after this initial period, but doing so will get caught because its PE value will soon become conspicuous among s's neighbors. Thus, the malicious neighbors won't be able to confuse/disrupt the synchronization process by attacking the initial M iterations.

Fig. 1 shows how ATSP's attack detector interacts with the profile and clock-model managers. At each iteration, the attack-detector module determines if there exist anomalies in his neighborhood by computing $e_{s,i}(k)$ with Eqs. (7) and (11) based on the information provided by the clock-model and profile managers. If s determines the presence of anomalies that meet Eq. (11), it will establish and update reputation records for the neighbors that are suspected to have been compromised. Node s may blacklist and/or reject announcements from a neighbor if it behaved anomalously more than N_B times out of N_0 iterations. The ratio $N_B/N_0 (\le 1)$ is a predefined parameter reflecting how aggressively node s reacts to anomalies. The the choice of N_B/N_0 close to 1 indicates the lenience against anomalies and vice versa. This information will then be fed back to the profile and clock-module managers to reject announcements from those neighbors.

B. Accuracy of Attack Detection

ATSP achieves high detection capability and accuracy by *amplifying* the prediction errors caused by false clock announcements for M consecutive iterations (each presenting multiple chances to be caught). In other words, a false announcement from node i will cause ATSP to continuously produce high $e_{s,i}(\cdot)$ values over M consecutive iterations leading to the high probability of detecting the misbehaving node i, unless it's too weak to pose any threat to the synchronization.

Let us consider the case where a malicious node i mounted an attack to node s causing s to compute $c_{s,i}(k)$ that differs from the expected $c^*_{s,i}(k)$ by Δ, i.e., $c_{s,i}(k) = c^*_{s,i}(k) + \Delta$ (Δ can be viewed as the *attack strength*). This will also increase the prediction error $e_{s,i}(k)$ by Δ, namely, $e_{s,i}(k) = e^*_{s,i}(k) + \Delta$ from Eq. (7). Furthermore, the prediction error will get amplified rapidly over the subsequent iterations for the following reason. From Eq. (8), the filter-weight vectors are updated as $\hat{\mathbf{h}}_{s,i}(k) = \hat{\mathbf{h}}^*_{s,i}(k) + \Delta \cdot \mathbf{g}_{s,i}(k)$. Then, $e_{s,i}(k+1)$ can be rewritten as a function of Δ:

$$e_{s,i}(k+1) = e^*_{s,i}(k+1) - \Delta^2 \cdot g_{s,i,1}(k) - \Delta \cdot \left[\hat{h}_{s,i,1}(k) + \mathbf{g}^T_{s,i}(k)\mathbf{f}_{s,i}(k;M)\right] \tag{12}$$

where $g_{s,i,1}(k)$ is the first element of $\mathbf{g}_{s,i}(k)$. Therefore, the perturbation Δ introduced at iteration k results in a very large prediction error at the next iteration. Moreover, this amplification of prediction error will persist over M consecutive iterations, during which the abnormal $c_{s,i}(k)$ remains cached inside the profile manager, thus making it very difficult for the attacker to evade ATSP. This explains the high level of ATSP's attack-detection capability. Thanks to the attack-amplification property of ATSP that boosts $e_{s,i}(k)$ to a high value, the choice of η_t is not critical.

C. Security Analysis

In Section III-B, we classified possible attacks against time-synchronization into 3 types. The goal of attackers is to

influence and cause a significant bias in the synchronization process by providing false clock information. All of the 3 attack types are equivalent to modifying the send or receive timestamps in the synchronization messages, so that the acceptance of these falsified timestamps by victim nodes will lead to incorrect adjustment of their local clocks. Specifically, node-compromise or message-manipulation attack is to change the send timestamps and the message-delay attack is equivalent to modifying the receive timestamps. ATSP is resilient to these attacks, because in ATSP every node keeps refining its estimation of neighbors' clocks with the most recent history of behaviors, so that the time announcements from its neighbors should conform to their expected behavior profiles. Under the accurate prediction framework provided by the adaptive transversal filter and the RLS algorithm, any clock announcement that deviates considerably from the corresponding prediction will be judged to be suspicious. This places great stress on the attackers and forces them to weaken the attack strength in order to evade the attack-amplification property of ATSP. However, as long as malicious nodes do not form a majority within the local region of interest, such low-strength attacks cannot disrupt the time-synchronization service, because any attempt by a malicious node i to modify the clock value of its neighbor s makes only one n_s-th contribution to s's clock value thanks to the distributed synchronization protocol used in ATSP. (Notice that the slow poisoning attack that attempts to gradually speed up/slow down the clocks is not a problem, because our goal is to achieve the equilibrium time among sensors rather than the synchronization to the absolute time.) This creates a dilemma for the attacker that either he must risk being caught when falsifying the time too much, or he will not do any harm if staying below the detection threshold. Thus, the only effective way to evade detection of ATSP *locally* is to compromise a majority of nodes within the local region. However, this attack only disrupts nodes inside or near the affected region and the rest of the network can still compute a correct clock value. This demonstrates the cooperative nature of ATSP and evading such detection is very difficult even for determined attackers.

Another serious attack against ATSP is the sybil attack [20], where a single malicious node illegitimately presents multiple identities in the network to control a substantial fraction of the ID space, thus invalidating the fundamental assumption of ATSP that a majority of nodes in the network are well-behaving. Many solutions have been proposed to counter sybil attacks. First, the resource-verification method detects a sybil attack by checking if each claimed identity has as much resources as the real physical device, assuming that each physical entity has limited storage, computation [20], and radio channels [21]. Second, if a secure localization service is available, the position-based verification is also an effective countermeasure, as sybil nodes will appear to be at the same position. In addition, the random key distribution [22, 23] is also a promising measure against sybil attacks, where each node compute a unique pairwise key it shares with each neighbor based on the preloaded key-related information. Thus, an attacker cannot fabricated the identity of a sensor node unless he compromises the sensor and reads the key-related information. Any of these techniques can be adopted into ATSP to defeat the Sybil attack. We consider the random key predistribution technique such as the Blundo scheme [23] used in [24] as an appropriate choice because it not only incurs a relatively small overhead but also provides additional security for the radio communication already required by many other services and applications.

Initial values:
1. Init_Phase = TRUE
2. $k = 0$
3. $\mathbf{h}_{s,i}(M) = 0$
4. $\mathbf{P}_{s,i}(M) = \rho^{-1}\mathbf{I}$

Iteration:
5. **while** (1)
6. $k = k + 1$ *// Start the k-th iteration*
7. broadcast the Sync message
8. receive T_i from every neighbor i
9. **for** i = 1 **to** n_s
10. compute $\Delta_{s,i}$ and $c_{s,i}(k)$ using Eq. (4)
11. **if** $k \geq M + 1$ *// Activate detection at* $M + 1$
12. **for** i = 1 **to** n_s
13. compute $e_{s,i}(k)$ using Eq. (7)
14. **if** $|e_{s,i}(k)| \geq \max\{\eta_t \cdot c_{s,i}(k), e_{\min}\}$
15. $c_{s,i}(k) = \mathbf{h}^T_{s,i}(k)\mathbf{c}_{s,i}(k-1; M)$
16. $\Delta_{s,i} = c_{s,i}(k) \times (T_s(k) - T^0_s(k))$
17. **if** ++ blacklist_count(i) $\geq N_B$
18. blacklist i
19. **if** i is blacklisted
20. deactivate $\mathbf{h}_{s,i}(k)$
21. **else**
22. update $\mathbf{h}_{s,i}(k)$, $\mathbf{g}_{s,i}(k)$ and $\mathbf{P}_{s,i}(k)$

// Update the clock value

23. $C_{old}(t) = C_s(t)$
24. $C_s(t) = C_s(t) + \frac{1}{m+1}\sum_{i=1}^{m} \Delta_{s,i}$

// Determine if Init_Phase is done

25. **if** Init_Phase = TRUE and $|C_s(t) - C_{old}(t)| < \epsilon$
26. Init_Phase = FALSE
27. **if** Init_Phase = TRUE
28. sleep(Init_Interval)
29. **else**
30. sleep(Resync_Interval)

Fig. 3. Pseudocode for ATSP at sensor node s

VI. Protocol Description

Fig. 3 provides the pseudocode of ATSP executed at node s (N_0 is set to be unbounded in the pseudocode for simplicity). It integrates the operations for the attack detector, the profile manager and the ICTS algorithm.

A. Two-Phase Execution of ATSP

ATSP is executed in two phases: **Initialization Phase** and **Resynchronization Phase**, because the discrepancies of clock drift rates necessitate the periodic resynchronization after the initial synchronization process. ATSP at node s starts with initialization of the profile manager (lines 3–4). s broadcasts the Sync message and collects Sync messages from all of its neighbors. Then, s executes up to M iterations with the detection module disabled (to establish the initial normal profile), turns it on at iteration $M + 1$, and performs the anomaly detection described in Section V throughout the remainder of time-synchronization. Upon detecting suspicious clock announcements (line 14), ATSP takes the following two measures. First, the anomalous clock values at iteration k must be excluded from (i) the recursion for the profile manager by letting $c_{s,i}(k) = \mathbf{h}^T_{s,i}(k)\mathbf{c}_{s,i}(k-1; M)$, and (ii) the computation of s's clock value by setting $\Delta_{s,i} =$

$c_{s,i}(k) \times (T_s(k) - T_s^0(k))$, Second, the malicious neighbor i must be removed from the future synchronization process (i.e., by deactivating $\mathbf{h}_{s,i}(k)$) if caught more than N_B times.

B. Clock Types

Since different sensors' clocks run at different frequencies, the offsets among sensors' clocks will monotonically increase and reach the maximum before the next resynchronization point. Normally, to prevent this degradation of accuracy, sensor nodes must be re-synchronized frequently, thus incurring a significant overhead in terms of bandwidth and computation. ATSP alleviates this problem by maintaining a *Calibrated Clock* in addition to the Local Clock (the system clock driven by the crystal oscillator in each sensor node) and Global Clock (a logic clock synchronized at each synchronization point). Calibrated Clock utilizes the normal profiles (or the relative drift rate models) to estimate the neighbors' clocks (Eqs. (3) and (4)). By assuming that sensors' clocks drift in a linear fashion (i.e., $C_i(t) = \alpha_i t + \beta_i$) and the drift rate α_i has good short-term stability [3], each node can accurately predict its neighbors' clock values at any time and use them to resynchronize with each other with no extra overhead. The calibrated time is thus a logic clock calculated by this on-demand resynchronization using the estimated clocks. It works as follows.

Assume at node s's global time T_s, s needs to calculate its calibrated time CT_s. Let T_s^0 denote the global time when s finishes the last resynchronization, and $\hat{\alpha}_{s,i}(k),\ i = 1, 2, \ldots, n_s$ indicate the estimated relative drift rate for each of s's neighbors during the current (k-th) iteration. Based on Eqs. (3) and (4), we have: $\alpha_{s,i}(k) = 1 + \Delta_{s,i}/(T_s(k+1) - T_s^0(k+1)) = 1 + c_{s,i}(k)$. The PE value for the k-th iteration ($\hat{c}_{s,i}(k)$) can be estimated as: $\hat{c}_{s,i}(k) = \mathbf{h}_{s,i}^T(k)\mathbf{c}_{s,i}(k-1; M)$. Thus the relative drift rate between s and i is: $\hat{\alpha}_{s,i}(k) = 1 + \mathbf{h}_{s,i}^T(k)\mathbf{c}_{s,i}(k-1; M)$. Based on the definition of relative drift rate, the estimated remote clock $\hat{T}_i$ at neighbor i can be computed as: $\hat{T}_i = T_s^0 + \hat{\alpha}_{s,i}(k)(T_s - T_s^0)$. The estimated offsets between i and s are then: $\hat{\Delta}_{s,i} = T_s - \hat{T}_i = (1 - \hat{\alpha}_{s,i})(T_s - T_0)$. Now, the calibrated time can be easily computed using Eq. (1) with estimated offsets: $CT_s = T_s + \frac{1}{n_s+1}\sum_{i=1}^{n_s} \hat{\Delta}_{s,i}$.

VII. Evaluation

We evaluate ATSP via simulation with NESLsim [2, 25] which is a PARSEC [26] based simulation platform for sensor networks. Our simulation environment consists of 150 sensor nodes randomly deployed in a circular area of diameter $100m$. The maximum communication range is set to $35m$. Each sensor has a clock drift rate randomly chosen between 0 and 30 μs per second and the coefficient of variation is set to 0.1. The maximum initial offset among sensors is chosen to be $2 \times 10^8 \mu s$ (200 s). The default resynchronization interval is fixed at 60 s and the threshold of synchronization accuracy is $350\mu s$. Throughout the simulation, ATSP uses the following default parameter values: $\lambda = 0.95$, $\rho = 0.1$ and $M = 5$. All results are averaged over 50 simulation runs.

A. Accuracy of Time-Synchronization

Although The distributed and mutual synchronization protocol like ATSP has the advantage of high resilience to a number of attacks, it inherently incurs more overhead since a sensor node cannot trust any single neighbor. It has to consult multiple nodes and requires multiple iterations for the clock value to converge. To evaluate the tradeoff, we execute ATSP in an attack-free environment and plot the tradeoff (Fig. 4) between the number of iterations and the resulting synchronization accuracy during the initialization phase. In the figure, the network synchronization error is the maximum difference between the clocks of a pair of sensor nodes in the network, and the neighbor synchronization error is the maximum difference between the clocks of a pair of neighboring nodes. ATSP is shown to be able to achieve accurate synchronization within a moderate number of iterations; it suppresses both the network and neighbor synchronization errors from the initial $2 \times 10^8 \mu s$ to less than 10 μs within 50 iterations. Considering that only a single broadcast message is needed per node for each iteration, ATSP achieves, at the expense of reasonable overhead, accurate synchronization.

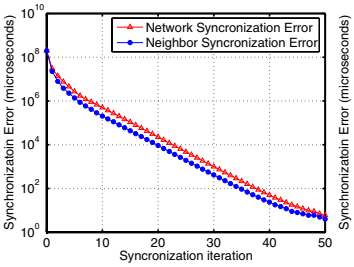

Fig. 4. Sync errors in the initial phase

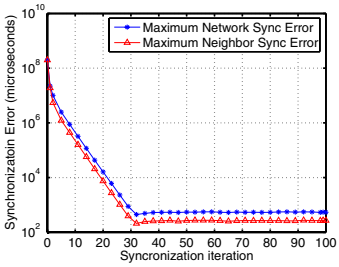

Fig. 5. Sync errors in two phases

After the initialization phase, the network performs periodic resynchronization to compensate for the different drift rates that cause nodes to lose synchronization. Fig. 5 shows the synchronization accuracy for both initialization and resynchronization phases. The transition from initialization to resynchronization phase occurs at iteration 31. The initialization phase reduces the synchronization errors from $2 \times 10^8 \mu s$ to about 300 μs while the resynchronization phase maintains this accuracy. The resynchronization interval represents a trade-off between the synchronization accuracy and the resulting overhead. Intuitively, better accuracy can be achieved with a smaller resynchronization interval at a higher communication overhead. Fig. 6 plots the relationship between the resynchronization interval and the corresponding accuracy. Clearly, the frequency of resynchronization is proportional to the accuracy of ATSP. Hence, an appropriate choice of resynchronization interval is very important in balancing the cost and accuracy, and could be determined from Fig. 6.

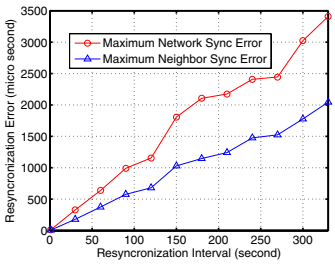

Fig. 6. Resync interval and accuracy

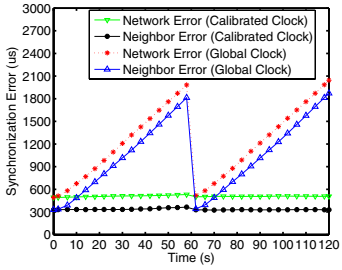

Fig. 7. Calibrated vs global clocks

Next, we evaluate the effectiveness of calibrated clocks in compensating for the clock drift in-between synchronization

points. As mentioned before, the clock drift will monotonically increase between the two synchronization points and reaches its maximum value just before the next synchronization. This is illustrated by the upper two curves in Fig. 7 which depict errors of global clocks during two consecutive synchronization iterations, the interval of which is 60 seconds. The figure shows that even though the global clock is synchronized at time 0 and 60 seconds, the error keeps increasing and reaches the peak (around 2000μs) just before the next synchronization is triggered. On the other hand, using the calibrated clocks (the lower two curves in Fig. 7) is found to stabilize and preserve the synchronization accuracy almost at the same level as that achieved by the real synchronization with no extra overhead.

B. Performance of Profile Manager

We evaluate the performance of the profile manager in terms of its prediction accuracy in an attack-free environment. The metric used to quantify the accuracy is the *normalized prediction error* (NPE) defined as: $NPE_{s,i}(k) = \frac{|e_{s,i}(k)|}{max\{c_{s,i}, c_{min}\}}$, $i = 1 \cdots n_s$ and $k \geq M + 1$. That is, $NPE_{s,i}(k)$ is the absolute prediction error for neighbor i's clock announcement at iteration k, normalized by the real PE value $c_{s,i}(k)$. We collect NPE values of all nodes from 50 independent simulation runs and plot the average $NPEs, i(k)$ over all sensor nodes as a function of k in the first graph of Fig. 8. From the figure, the NPE value is found to decrease very quickly and stay constantly around 0.05 after collecting enough data, i.e., after iteration $M+1$ (= 6). This demonstrates that each sensor node could construct a ready-to-use profile very fast with high accuracy. Moreover, the profile manager incurs relatively small processing overhead because of the small value of M used in the filter. Therefore, the proposed profile management based on adaptive filtering captures the synchronization behavior in a compact and efficient form.

C. Performance of Attack Detector

To evaluate the attack-detection capability, we construct attack scenarios in which a malicious node injects false clock values at iteration 40 that differ from its real clock by 0.5, 1 and 2 ms, respectively. We use NPE as the evaluation metric because it effectively quantifies the attack strength and identifies the source of false information. Fig. 8 plots the NPE values under these attack scenarios. From the figure, we observe that the attack mounted at the 40-th iteration increases $NPE_{s,i}(40)$ in proportion to the attack strength. Moreover, the NPE values of subsequent M (= 5) iterations continue to exhibit unusually high values that differ from the normal values by orders-of-magnitude. This means that an adversarial impact resides in the profile for M consecutive iterations, during which it continuously produces abnormal prediction errors. These results clearly agree with the analysis in Section V. For this reason, any attack that yields even small deviations of clock value is likely to be detected by ATSP, thanks to its error-amplification property.

D. Attack-Tolerance

We now evaluate ATSP's attack-tolerance to show that the synchronization service is disrupted by neither "weak" nor "strong" attacks (of strength just below, or well above, the detection threshold). We first ran simulation with the attack-detector turned off to find the range of attack strengths that our synchronization can withstand. We then turned on the attack detector for the subsequent simulation to quantify the gain with our attack detector in terms of withstanding stronger attacks.

1) ATSP without Attack Detector: Fig. 9 plots the synchronization errors in the presence of attacks of varying strengths represented by the clock announcements that differ from the true values by 0.5, 1, 5 and 10 ms, respectively. Clearly, the attack of 0.5ms strength is completely canceled out. Even for an attack of 1ms strength, no noticeable increase exists in the neighbor synchronization error. By contrast, according to Fig. 8, even the 0.5ms attack produces highly unusual NPE values and will easily be detected. Therefore, attackers have to lower their attack strength below 0.5ms to evade ATSP's detection, but doing so will not harm the synchronization, as demonstrated in Fig. 9. The remaining two plots in Fig. 9 show the cases of noticeable surges in synchronization errors, simply because the attack strengths exceed the tolerance bound of distributed protocol. However, it is important to note that the impact of attacks is still reduced by orders-of-magnitude. This demonstrates ATSP's capability of mitigating the impact of attacks. However, it also calls for an attack-detection module to identify and exclude notable attacks that may defeat ICTS.

2) ATSP with Attack Detector: Finally, we ran simulation with the attack-detector module enabled to evaluate the overall attack-tolerance of ATSP. Sensor nodes evaluate Eq. (11) to locate and exclude false clock announcements from clock synchronization and updating of normal profiles. Fig. 10 shows the simulation results when nodes experience the same degree of attacks as that in Fig. 9 Unlike previous results, there are no synchronization-error spikes even under severe attacks, indicating that strong attacks that cannot be tolerated by ICTS are eliminated by our attack-detector module.

In summary, the attack-detector module and the ICTS algorithm effectively complement each other. ATSP uses the attack detector to block strong attacks before they reach and defeat ICTS, while ICTS will smooth out the weak attacks missed by the attack detector. Therefore, the attack detector and ICTS form a cooperative multiple-layer defense mechanism; the attack detector serves as the first line of defense that detects and counters attacks in new time-advertisements received, while ICTS is the second line of defense that reduces or eliminates the impact of attacks evaded the attack detector.

VIII. Conclusion

We proposed an attack-tolerant time-synchronization protocol (ATSP) for WSNs by exploiting temporal correlations among adjacent sensor nodes to build compact profiles of normal synchronization behaviors. ATSP is a distributed, mutual synchronization scheme which provides a high level of attack-tolerance at a reasonable cost. ATSP, as a cooperative intrusion/anomaly detection system, compares a new time announcement with an adaptively-managed profile and identifies time announcements that deviate significantly from the expected normal behavior. ATSP compensates for the clock drift in-between the synchronization points by evaluating calibrated clock which utilizes relative drift rate models to perform resynchronization with the neighbors' estimated clocks. Using a recursively-refined relative drift model, the calibrated clock significantly improves both accuracy and stability during each resynchronization interval while incurring no extra overhead.

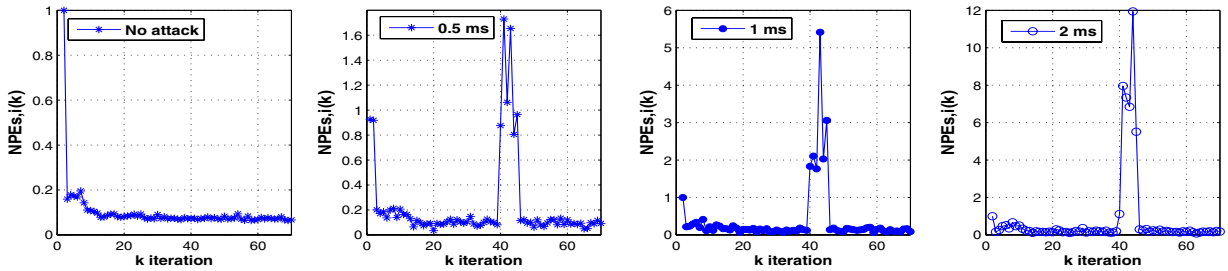

Fig. 8. NPE in the presence of attacks of strengths 0(no attack), 0.5, 1 and 2 ms

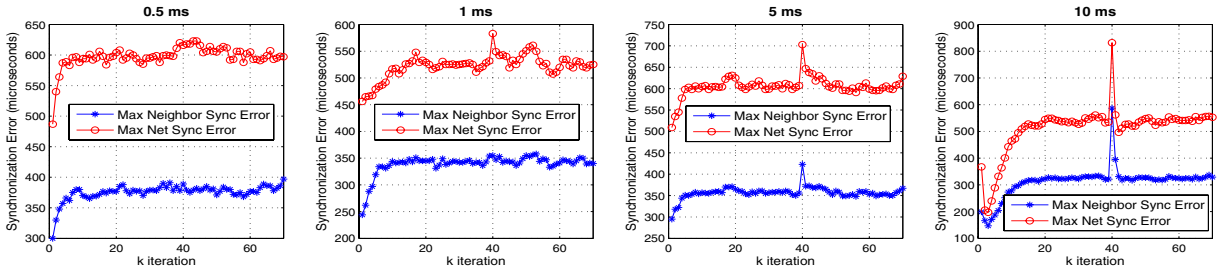

Fig. 9. Synchronization errors with the attack-detector disabled in the presence of attacks of different strengths

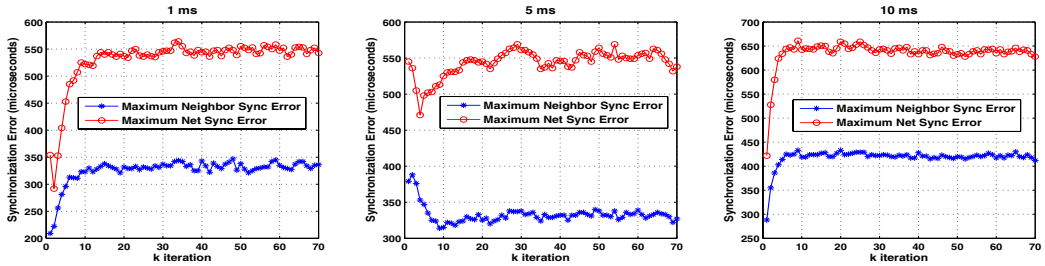

Fig. 10. Synchronization errors with the attack-detector module enabled in the presence of attacks of different strengths

The analysis and evaluation of ATSP demonstrate its high attack-tolerance and feasibility for resource-limited WSNs.

REFERENCES

[1] J. Elson, L. Girod, and D. Estrin, "Fine-Grained Network Time Synchronization using Reference Broadcasts," in *Proceedings of OSDI '02*.

[2] S. Ganeriwal, R. Kumar, and M. B. Srivastava, "Timing-sync Protocol for Sensor networks," in *The First ACM Conference on Embedded Networked Sensor System(SenSys)*, November 2003.

[3] M. Maróti, B. Kusy, G. Simon, and Ákos Lédeczi, "The flooding time synchronization protocol," in *SenSys '04: Proceedings of the 2nd international conference on Embedded networked sensor systems*.

[4] W. Su and I. F. Akyildiz, "Time-diffusion synchronization protocol for wireless sensor networks," *IEEE/ACM Trans. Netw.*, 2005.

[5] J. van Greunen and J. Rabaey, "Lightweight time synchronization for sensor networks," in *WSNA '03: Proceedings of the 2nd ACM international conference on Wireless sensor networks and applications*.

[6] S. Ganeriwal, S. Capkun, C.-C. Han, and M. B. Srivastava, "Secure time synchronization service for sensor networks," in *WiSe '05: Proceedings of the 4th ACM workshop on Wireless security*, 2005.

[7] H. Song and S. Zhu, and G. Cao, "Attack-Resilient Time Synchronization for Wireless Sensor Networks," in *MASS05*, November 2005.

[8] D. Mills, "Network time protocol (ntp)," RFC 958, September 1985.

[9] S. PalChaudhuri, A. K. Saha, and D. B. Johnson, "Adaptive clock synchronization in sensor networks," in *Proceedings of 3rd international symposium on Information processing in sensor networks*, 2004.

[10] Q. Li and D. Rus, "Global clock synchronization in sensor networks." *IEEE Trans. Computers*, vol. 55, pp. 214–226, 2006.

[11] M. Manzo, T. Roosta, and S. Sastry, "Time synchronization attacks in sensor networks," in *SASN '05*, 2005.

[12] K. Sun, P. Ning, and C. Wang, "Secure and resilient clock synchronization in wireless sensor networks," *IEEE Journal on Selected Areas in Communications*, vol. 24, no. 2, pp. 395– 408, February 2006.

[13] K. Sun, "Fault-tolerant cluster-wise clock synchronization for wireless sensor networks," *IEEE Trans. Dependable Secur. Comput.*, vol. 2, 2005.

[14] K. Tang and M. Gerla, "Mac reliable broadcast in ad hoc networks," in *IEEE MILCOM'01*, 2001.

[15] M. L. Sichitiu and C. Veerarittiphan, "Simple, accurate time synchronization for wireless sensor networks," in *IEEE Wireless Communications and Networking Conference,WCNC 2003*, March 2003.

[16] Fikret Sivrikaya and Bülent Yener, "Time Synchronization in Sensor Networks: A Survey," *IEEE Network Magazine's special issue on "Ad Hoc Networking: Data Communications & Topology Control"*, 2004.

[17] L. Lamport and P. Melliar-Smith, "Synchronizaing Clocks in the Presence of Faults," *Journal of the Association for Computing Machinery*, vol. 32, no. 1, pp. 52–78, January 1985.

[18] P. Ramanathan, D. D. Kandlur, and K. G. Shin, "Hardware-assisted software clock synchronization for homogeneous distributed systems," *IEEE Trans. Comput.*, vol. 39, no. 4, pp. 514–524, 1990.

[19] S. Haykin, *Adaptive Filter Theory*, 2nd ed. Prentice-Hall, 1991.

[20] J. Douceur, "The Sybil Attack," in *Proceedings of 1st International Workshop on Peer-to-Peer Systems*, 2002.

[21] J. Newsome, E. Shi, D. Song, and A. Perrig, "The Sybil Attack in Sensor Nework: Analysis & Defense," in *Proceedings of 3rd IEEE/ACM Information Processing in Sensor Networks (IPSN'04)*, 2004.

[22] D. Liu and P. Ning, "Establishing pairwise keys in distributed sensor networks," in *CCS '03: Proceedings of the 10th ACM conference on Computer and communications security*, 2003.

[23] C. Blundo, A. D. Santis, A. Herzberg, S. Kutten, U. Vaccaro, and M. Yung, "Perfectly-secure key distribution for dynamic conferences," *Lecture Notes in Computer Science*, vol. 740, pp. 471–486, 1993.

[24] K. Chang and K. G. Shin, "Distributed authentication of program integrity verification in wireless sensor networks," Appears in Securecomm and Workshops, 2006, Aug 2006.

[25] S. Ganeriwal, V. Tsiatsis, C. Schurgers, and M. B. Srivastava, "Neslsim: A parsec based simulation platform for sensor networks," http://www.ee.ucla.edu/ saurabh/NESLsim/, 2002.

[26] R. Bagrodia and R. Meyer, "Parsec: A parallel simulation environment for complex system," "http://pcl.cs.ucla.edu/projects/parsec/", 1998.