HAL
open science

Efficient Time Synchronization in a Wireless Sensor Network by Adaptive Value Tracking

Kasim Sinan Yildirim, Önder Gürcan

To cite this version:

Kasim Sinan Yildirim, Önder Gürcan. Efficient Time Synchronization in a Wireless Sensor Network by Adaptive Value Tracking. IEEE Transactions on Wireless Communications, 2014, 13, pp.3650 - 3664. 10.1109/TWC.2014.2316168 . cea-01807005

HAL Id: cea-01807005

https://cea.hal.science/cea-01807005v1

Submitted on 4 Jun 2018

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Efficient Time Synchronization in a Wireless Sensor Network by Adaptive Value Tracking

Kasım Sinan Yıldırım and Önder Gürcan

***Abstract*—A desirable flooding-based time synchronization protocol in Wireless Sensor Networks (WSNs) should neither demand fast propagation of up-to-date time information nor keeping track of the neighboring nodes. Moreover, such a protocol is strictly required to have low computational and communication overhead as well as small memory footprint. Would there be a protocol which meets these requirements' We answer this question positively by introducing a novel time synchronization protocol whose main component is "adaptive-value tracking". Thanks to this component, each sensor node synchronizes the rate of its clock to that of a reference clock through successive feedbacks with a considerably low computational and memory overhead. By adjusting time offset of the rate-synchronized clocks, the network-wide synchronization is established even without demanding rapid propagation of the reference clock and keeping track of the neighboring nodes. In the light of our experimental evaluation in a testbed of 20 MICAz sensor nodes, we observed that the proposed protocol provides similar synchronization under the same communication frequency with an approximately 97% less CPU overhead and 80% less memory allocation compared to the recent flooding based time synchronization protocols in WSNs.**

***Index Terms*—Distributed algorithms, flooding based time synchronization, adaptive value tracking (AVT).**

I. Introduction

THE accurate and efficient operation of many applications and protocols in wireless sensor networks (WSNs) require synchronized notion of time. Unfortunately, built-in *hardware clocks* of sensor nodes are not sufficient alone to fulfill this requirement since they frequently drift apart. By means of time synchronization, each sensor node establishes a *logical clock* whose value at any time represents the network-wide global time. However, this establishment is not straightforward and requires coping with several aspects such as environmental dynamics, various error sources arising from communication, frequent topological changes and power, memory and computation constraints of the sensor nodes. These aspects make time synchronization challenging.

Up until now, researchers proposed several protocols for time synchronization in WSNs. These protocols can be roughly classified into *fully-distributed* protocols where sensor nodes interact only with their direct neighbors in a peer-to-peer fashion and *flooding based* protocols where one or more special node act as time reference for the other nodes. Fully distributed protocols, e.g., protocols in [1], [2], exploit all neighborhood information and they strive to locally optimize the synchronization error. Hence, these protocols are more appropriate for protocols such as TDMA [3] which require better synchronization among the neighboring nodes. However, local optimization comes at the price of slow dissemination of the time information and hence a large global synchronization error [4]. On the other hand, the flooding based protocols are known as to exhibit better global synchronization error [1]. Moreover, if synchronization to stable time sources such as Coordinated Universal Time (UTC) is required, i.e., external synchronization, employing the method of flooding to provide time synchronization becomes crucial [5].

In this article our focus is on flooding-based time synchronization in WSNs where one or more predefined or dynamically elected reference nodes periodically disseminate their stable information through the network via flooding. The receiver nodes collect reference time information and synchronize by calculating the offset and frequency difference of their clocks with respect to the reference clock. To establish network-wide time synchronization, they also propagate the value of their synchronized clock to the other nodes in the network. This approach is robust to dynamic topological changes since there is no demand to construct and maintain a topological infrastructure. In case of reference node failures, dynamic leader election or having multiple reference nodes increase the robustness of this approach.

There are several flooding based time synchronization protocols in the WSN literature. Among them, Flooding Time Synchronization Protocol (FTSP) [6] received considerable attention from both researchers and practitioners due to its simplicity, public availability, robustness to network dynamics and low communication overhead. In FTSP, sensor nodes synchronize to a dynamically elected reference node by collecting its flooded time information and perform least-squares regression on the collected data to estimate future clock values of the reference node. The network-wide synchronization is achieved by having sensor nodes broadcast their estimates for their neighboring nodes in a *slow* manner. Interestingly, recent studies [4], [7], [8] pointed out that slow-propagation of time information and performing estimation with least-squares regression together give rise to a substantial degradation of synchronization accuracy as the network diameter increases.

Manuscript received April 17, 2013; revised October 1, 2013 and January 20, 2014; accepted March 29, 2014. Date of publication April 9, 2014; date of current version July 8, 2014. The associate editor coordinating the review of this paper and approving it for publication was A. Vosoughi.

K. S. Yıldırım is with the Department of Computer Engineering, Ege University, İzmir 35100, Turkey (e-mail: sinan.yildirim@ege.edu.tr).

Ö. Gürcan is with the Department of Computer Engineering, Ege University, İzmir 35100, Turkey, and also with the Laboratory of Model Driven Engineering for Embedded Systems, CEA, LIST, F-91191 Gif-sur-Yvette, France (e-mail: onder.gurcan@cea.fr; onder.gurcan@ege.edu.tr).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TWC.2014.2316168

1536-1276 © 2014 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission.
See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

In contrast, PulseSync [4] offers the time information of the reference node to be propagated as fast and reliably as possible through pulses to improve the scalability. It has been shown by experimental and simulation results that such *rapid* flooding improves the synchronization accuracy drastically and it is desirable to adapt to system dynamics quickly. However, establishing rapid flooding in WSNs is difficult [5], [9]. Current trend to provide rapid flooding in WSNs is to exploit *constructive interference* [10], which allows multiple senders to transmit an identical packet simultaneously. Unfortunately, this mechanism requires fragile hardware-dependent software solutions and additional hardware capabilities [11], and worse, it has a serious scalability problem [12], [13].

A recent protocol, namely Flooding with Clock Speed Agreement (FCSA) [8], eliminates the demand of rapid flooding by employing a clock speed agreement algorithm that requires to keep track of the neighboring nodes. Nodes whose clocks are running at identical rates synchronize to a reference node which floods stable time for the whole network in a slow manner, as in FTSP. Unfortunately, FCSA suffers from the crucial problem of deciding *which neighbors to keep track of and which ones to discard* since it is infeasible for sensor nodes to keep track of all their neighbors due to their memory constraints. Moreover, it takes a long time to provide tight synchronization with FCSA.

Having mentioned the drawbacks of these recent studies, we claim that existing flooding based time synchronization in WSNs is still unsatisfactory. From our point of view, a desirable protocol should neither demand propagating the time information in a rapid manner nor force sensor nodes to keep track of their neighboring nodes. In addition to exhibiting these desirable properties, such a protocol is strictly required to have low computational and communication overhead as well as small memory footprint. To this end, we present in this article a novel protocol which meets these requirements.

A. Contributions

In this article, we consider the problem of synchronization to a reference clock from a different perspective compared to the existing flooding based approaches. We handle this problem as a *search process* in which each sensor node is trying to find the rate of the reference clock *without knowing its correct value*. Due to the dynamics of the WSN environment, e.g., where clock drifts are subject to changes quite frequently, an adaptive search technique is required for the search process. To this respect, we employ the technique of Adaptive Value Tracking (AVT) [14], which finds and tracks a *dynamic* searched value in a given search space through successive feedbacks.

In particular, we propose a novel flooding based protocol for time synchronization in WSNs. The proposed protocol, namely Adaptive Value Tracking Synchronization (AVTS), synchronizes the rates of the clocks of sensor nodes to a reference clock by employing the AVT technique. By means of adjusting time offset of the rate-synchronized clocks, the whole network becomes synchronized. With this simple mechanism that neither requires least-squares nor distributed agreement, we observed drastic improvements over recent flooding based protocols through experiments performed in our testbed of 20 MICAz sensor nodes and simulations. In the light of these observations, we list the superiorities and desirable properties of AVTS as follows:

- AVTS catches the synchronization performances of PulseSync and FCSA under the same communication frequency with an approximately 97% less CPU overhead and considerably smaller code size.
- AVTS neither requires memory storage to collect time information of the reference node, e.g., a regression table, nor keeps track of the neighboring nodes. Hence, our implementation has approximately 80% less memory footprint when compared to FTSP, PulseSync and FCSA.
- Since errors accumulate additively at each hop, AVTS is quite scalable and does not demand the time information to be propagated in a rapid manner, hence eliminating the drawbacks of rapid flooding.
- AVTS achieves tight synchronization quickly compared to another slow flooding approach FCSA, which requires a long time for sensor nodes to agree on a common logical clock speed.

B. Organization

The remainder of this article is organized as follows. In Section II we present the challenges of time synchronization in WSNs. By considering these challenges, we describe our system model in Section III. We introduce AVTS protocol in Section IV and we describe the technique of adaptive value tracking in detail in Section V. In Section VI, we analyze the synchronization performance of AVTS. Our implementation details and experimental work are presented in Section VII. Section VIII describes the related work and finally, we present our conclusions in Section IX.

II. Challenges for Time Synchronization in WSNs

Time synchronization is an important service for the collaborative and coordinated operations in WSNs. For instance, network protocols such as time division multiple access (TDMA) strictly demands synchronization among sensor nodes. Capabilities such as *temporal event ordering* of data collected from sensor nodes or providing access to a *global time* source, i.e., UTC, to integrate sensor network to Internet can be provided only if sensor nodes are synchronized.

Besides, low-cost clocks and resource constraints of the sensor nodes, error sources during communication, frequent topological changes and node failures are the main factors which affect time synchronization of WSNs. These factors should be considered during the design and implementation of the synchronization protocols.

The most important factor which affects synchronization is the low-cost materials used for the implementation of the built-in clocks of the sensor nodes. In WSNs, each sensor node is equipped with a read-only clock that contains a *counter register* and a low-cost *crystal oscillator*. The counter register is increased with each periodical oscillator pulse event which is called the *tick* of the clock. The duration between two

consecutive ticks is the *rate* of the built-in clock. External crystal oscillators in sensor nodes are used as clock source for their built-in clocks. *Clock drift* occurs where the built-in clock does not generate ticks at the exact speed of real-time. Environmental factors such as temperature, voltage level and aging of the crystal affect the drift rate. The most prominent environmental factor is the temperature whose change affects the frequency of the clocks non-linearly (See Fig. 1 in [4]). Generally, crystal oscillators in sensor nodes are reported to exhibit a *bounded drift* which is at most 100 ppm[1] [1]. Due to their different rates, the built-in clocks of the sensor nodes frequently drift apart, thus they only provide local time notion. Hence, sensor nodes are required to exchange their clock information periodically to synchronize their built-in clocks. Since the built-in clocks are read-only, each sensor node computes a *logical clock value* (software clock) that represents synchronized notion of time.

However, there are sources of errors that affect the computation of the logical clock, hence the synchronization error. One of the major error sources is the *transmission delay* introduced by the wireless communication, which is defined as the time that passes between the start of the broadcast attempt and the receipt by the receiver node. The transmission delay is composed of deterministic and non-deterministic components [6]. The non-deterministic components of the transmission delay affect the error of the synchronization in sensor networks directly, since nodes receive outdated and hence inaccurate time information due to delays. Apart from the transmission delay, another major error source is the granularity of the built-in clock that effects the error of the timing measurements. Since the time elapsed between consecutive ticks of the built-in clock depends on its frequency, *quantization errors* occur with low-frequency built-in clocks, which introduce additional errors to the calculation of the logical clock.

Unlike traditional distributed systems, sensor nodes typically are powered by batteries and energy is a very scarce resource. Since communication is expensive in terms of energy, time synchronization must be kept infrequent and small amounts of data should be exchanged. However, it is desirable that sensor nodes detect variations in their clock speeds and adapt quickly. Hence, reducing the frequency of the communication and quick adaptation are contradictory goals. In addition, computing, storage, and the communication capabilities of a single sensor node are limited. Technologies like GPS cannot be used for synchronization of nodes due to their high cost, size and energy overhead. Nodes in a sensor network may be mobile, they can stop operating due to depleted batteries or environmental factors, new sensor nodes may be added to network dynamically. The network topology may change frequently in an unpredictable way and there is no guarantee of stable connectivity between nodes. Moreover, the delays on the communication links between the sensor nodes are unpredictable. Thus, time synchronization protocols designed for WSNs must consider those dynamics.

[1]ppm = parts per million (microsecond). An oscillator with 100 ppm running at 1 MHz drifts apart 100 microsecond in one second.

III. System Model

In this section, we introduce a system model by considering the design challenges introduced in the previous section to present and analyze the time synchronization algorithms in this article.

A. Network Model

As in many studies in the literature, our mathematical model which represents a real sensor network is a graph $G = (V, E)$ with a vertex set $V = \{1, \ldots, N\}$ representing the sensor nodes and an edge set $E \subseteq V \times V$ representing the bidirectional communication links between these nodes. The set of nodes inside the wireless broadcast region of any node $u \in V$ is referred to as the neighbors of that node. These nodes are directly connected to node u in G and denoted by $\mathcal{N}_u = \{v \in V | \{u, v\} \in E\}$. To simplify our analysis in the next sections, we assume that communication is reliable and the network is static. Hence, a message sent by a node $u \in V$ is received by all of its neighbors $\mathcal{N}_u$. We denote the *transmission delay* on the communication link $\{u, v\} \in E$ at time t by $\gamma_{u,v}(t)$.

B. Hardware Clock Model

We assume that each node is equipped with a read-only hardware clock (built-in clock) subject to clock drift. The hardware clock of any sensor node $u \in V$ is denoted by $H_u()$, whose value at any real time t is modeled as

$$H_u(t) = \int_0^t h_u(\tau) d\tau. \tag{1}$$

Here, $h_u(\tau)$ represents the *rate* of the hardware clock at time τ. Since the crystal oscillators in sensor nodes are reported to have bounded drifts, we assume that the rate of the hardware clock H_u at any time t as $1 - \varepsilon \leq h_u(t) \leq 1 + \varepsilon$ where ε is a constant which satisfies $0 < \varepsilon \ll 1$. Moreover, we denote the *quantization error* due to the frequency of the hardware clock H_u at time t by $q_u(t)$.

C. Logical Clock Model

We denote the logical clock of any sensor node $u \in V$ by $L_u()$ which is an estimate for the global time. The value of the logical clock at time t is calculated as

$$L_u(t) = L_u(t_{up}) + l_u(t_{up}) \left(H_u(t) - H_u(t_{up})\right) \tag{2}$$

where t_{up} is the latest time such that a recent global time information is received and the progress rate and the offset of the logical clock are updated. As can be observed, the progress rate of the logical clock at time t is

$$\frac{dL_u}{dt}(t) = l_u(t_{up}) \frac{dH_u}{dt}(t) = l_u(t_{up}) h_u(t) \tag{3}$$

where $l_u(t_{up})$ denotes the *rate multiplier* which is calculated at the time t_{up}. Upon receiving a recent global time information, node u may speed up or slow down its logical clock by modifying l_u to synchronize its progress speed. It can be

noticed, according to (2), the amount of the *estimated progress* of the global time since the latest update, i.e., $l_u(t_{up})(H_u(t) - H_u(t_{up}))$, is added up to the value $L_u(t_{up})$.

D. Synchronization Error

The objective in time synchronization is to minimize the synchronization error which is defined as instantaneous differences of the logical clock values in the system, i.e., *clock skew*. The network synchronization error (*global skew*) at any time instant t is defined as the largest clock skew observed between arbitrary nodes at that time, which is formally defined as $\max_{u,v \in V}\{|L_u(t) - L_v(t)|\}$. We also formally define the *average global skew* as $(1/N)\sum_{u \in V} \max_{v \in V}\{|L_u(t) - L_v(t)|\}$. Similarly, the neighbor synchronization error (*local skew*) at any time instant t is defined as the largest clock skew observed between the neighboring nodes at that time, which is formally defined as $max_{u \in V, v \in \mathcal{N}_u}\{|L_u(t) - L_v(t)|\}$. Finally, we formally define the *average local skew* as $(1/N)\sum_{u \in V} max_{v \in \mathcal{N}_u}\{|L_u(t) - L_v(t)|\}$.

IV. Time Synchronization With Adaptive Value Tracking

To tackle the challenge of time synchronization given in Section II, we designed the Adaptive Value Tracking Synchronization (AVTS) protocol, whose objective is to synchronize sensor nodes to the clock of a reference node. In AVTS, a dynamically elected reference node floods its time information (e.g., the value of its hardware clock value) into the network. By using the flooded time information, each sensor node adjusts its time offset and tries to find the rate of the reference clock *without knowing its correct value*. To find the right rate value in a dynamic WSN environment (e.g., where clock drifts are subject to changes quite frequently), an adaptive search technique is required.

Algorithm 1 AVTS pseudo-code for node u with a fixed reference node ref

1: **Initialization**
2: $avt_u.init(\Delta_{\min}, \Delta_{\max}, v_{\min}, v_{\max})$
3: $seq_u \leftarrow 0$
4: set periodic timer with period B
5:
6: □ **Upon receiving** $< L_v, seq_v >$ **such that** $seq_u < seq_v$
7: $skew \leftarrow L_u - L_v$
8: **if** $skew > \delta$ **then** $avt_u.adjust(f \downarrow)$
9: **else if** $skew < -\delta$ **then** $avt_u.adjust(f \uparrow)$
10: **else** $avt_u.adjust(f \approx)$ **endif**
11: $L_u \leftarrow L_v$
12: $seq_u \leftarrow seq_v$
13:
14: □ **Upon timer timeout**
15: **if** $u = ref$ **then** $seq_u \leftarrow seq_u + 1$ **endif**
16: **broadcast** $< L_u, seq_u >$

In this sense, each sensor node $u \in V$ executing AVTS maintains an *adaptive value tracker*, i.e., avt_u, which searches and tracks the rate of the clock of the reference node ref with respect to its hardware clock rate h_u through successive feedbacks. Hence, the value provided by avt_u at any time t represents the rate multiplier l_u of the logical clock L_u. Upon receiving synchronization messages, sensor node u adjusts the value of avt_u through feedbacks, to speed up or slow down the progress rate of the logical clock L_u. As another variable, node u maintains a seq_u variable to store the largest sequence number received from the reference node.

The pseudo-code of the AVTS protocol is presented in Algorithm 1.[2] When node u is powered on, avt_u is initialized, which assigns upper and lower bounds for the searched rate value ($[v_{\min}, v_{\max}]$) and its adjustment step ($[\Delta_{\min}, \Delta_{\max}]$) (Algorithm 1, line 2). After this operation, avt_u assigns 1 to its initial value v_0 and $\Delta_{\max}$ to its initial adjustment step Δ_0 (the internal details of adaptive value tracking is given in Section V). Hence, the logical clock progresses at the same rate of the hardware clock unless it is modified upon receiving a recent synchronization message. Then, seq_u variable is initialized to zero (Algorithm 1, line 3). As a last step of the initialization, a periodic timer is started, which will fire at every B ticks of the hardware clock (Algorithm 1, line 4).

Receiving a synchronization message carrying a greater sequence number than seq_u indicates that the reference node initiated a new synchronization round (Algorithm 1, line 6). Hence, the received logical clock value can be considered as a fresh estimate of the reference clock.[3] To adjust its logical clock rate, node u calculates its clock skew by subtracting the received reference clock estimate from the value of its logical clock (Algorithm 1, line 7). If the skew is greater than a predefined *tolerance* δ, node u sends a decrease feedback $f \downarrow$ to avt_u to inform that the logical clock needs to progress at a lower rate, since the value of its logical clock is greater than that of the reference clock (Algorithm 1, line 8). Similarly, if the skew is smaller than *tolerance* δ, node u informs avt_u to progress its logical clock at a greater rate by sending an increase feedback $f \uparrow$ (Algorithm 1, line 9). If the skew is within tolerance bounds, the progress rate of the logical clock is considered to be closer to its desired value and a good feedback $f \approx$ is sent to avt_u (Algorithm 1, line 10). After informing its adaptive value tracker avt_u for adjusting the progress rate of its logical clock, node u updates the value of its logical clock to the received logical clock value (Algorithm 1, line 11). Finally, the sequence number is updated (Algorithm 1, line 12).

Upon a timer timeout (Algorithm 1, line 14), solely the reference node increments its sequence number and thus a new flood round is initiated (Algorithm 1, line 15). However, all of the sensor nodes broadcast the value of their logical clocks and sequence numbers to achieve network-wide synchronization (Algorithm 1, line 16). It should be noted that time information

[2]It should be noted that AVTS can be configured to work with a predefined reference node or with a dynamic reference node election mechanism. For simplicity, we present the algorithm with a predefined reference node ref.

[3]It should be noted that the value of the logical clock at any time is an estimate of the clock of the reference node at that time.

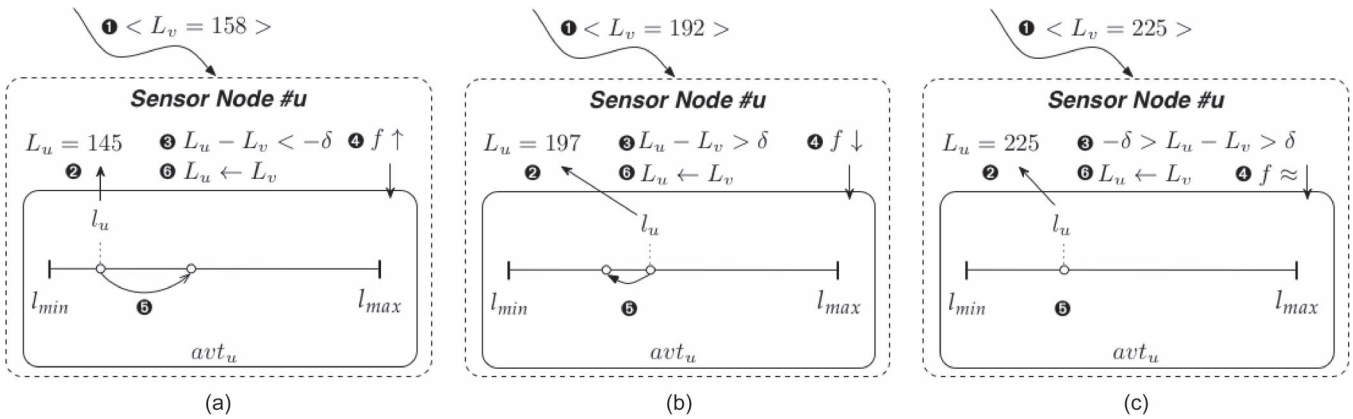


Fig. 1. Successive reactions of the sensor node u (a, b, and c, respectively) upon receiving successive synchronization messages carrying a greater sequence number than seq_u from its neighbor v. When a new message is received (a.1), the logical clock value (L_u) of u is calculated by using its latest rate multiplier value l_u (a.2). Then by comparing the logical clock values, the sensor node u sees that the skew is smaller than the predefined *tolerance* δ (a.3) and thus sends an increase feedback $f \uparrow$ to its adaptive value tracker avt_u (a.4). Upon receiving $f \uparrow$, avt_u increases its value (a.5). And then sensor node u updates the value of its logical clock L_u to the received logical clock value L_v (a.6). When a second message is received (respectively a third message), the same steps are executed. However, this time since the skew is greater than the predefined *tolerance* δ (resp. the skew is within tolerance bounds), a decrease feedback $f \downarrow$ (resp. a good feedback $f \approx$) is sent to avt_u and avt_u decreases its value (resp. avt_u does not change its value).

of the reference node may follow many paths to reach any node u in any synchronization round. Since the firstly received message can be considered as carrying the most up-to-date estimate of the reference node, node u discards other messages in that synchronization round (Algorithm 1, line 6).

An illustrative execution example of the AVTS protocol is shown in Fig. 1. As shown, the heart of AVTS protocol is adjusting the progress rate of the logical clock through an adaptive value tracking mechanism. We give the internal details of this mechanism in the next section.

V. Method of Adaptive Value Tracking

Adaptive Value Tracking is a search technique which was firstly proposed in [14]. In this method, an Adaptive Value Tracker (AVT) finds and tracks a *dynamic* searched value[4] in a given search space. The tracking is established via the successive feedbacks coming from the *environment* of the AVT[5] which indicate the direction that *probably* lead to the searched value.

Formally speaking, an avt searches (and tracks) a *dynamic* value $v^\star$ inside a given real interval (search space) $AVT_{ss} = [v_{\min}, v_{\max}] \subset \mathbb{R}$ where $v_{\min}$ is the lower boundary and $v_{\max}$ is the upper boundary for the searched value $v^\star$. At any time instant t, avt is able to propose a value $v_t \in AVT_{ss}$ to its environment which can be accessed using an action of the form $v_t = avt.value(t)$.

The objective of the environment of avt is to determine if the searched value $v^\star$ is smaller than, equal to or greater than the current proposed value v_t, without knowing the value $v^\star$. After this determination, the environment interacts with avt using an action of the form $avt.adjust(f_t \in \mathcal{F})$ for sending a *feedback* f_t from the feedback set $\mathcal{F} = \{f \uparrow, f \downarrow, f \approx\}$. The feedback f_t can be about increasing $v_t(f \uparrow)$, decreasing $v_t(f \downarrow)$ or informing that v_t is good $(f \approx)$.

After receiving the feedback, avt derives its next value v_{t+1} from v_t as

$$v_{t+1} = \begin{cases} v_t + \Delta_{t+1}, & f_t = f \uparrow \\ v_t - \Delta_{t+1}, & f_t = f \downarrow \\ v_t, & f_t = f \approx \end{cases} \tag{4}$$

where Δ_{t+1} is the *adjustment step* at time $t+1$.[6] It should hold for the adjustment step that

$$\Delta_t \in [\Delta_{\min}, \Delta_{\max}] \subset (0, |v_{\max} - v_{\min}|] \tag{5}$$

where $\Delta_{\min}$ is the lower boundary and $\Delta_{\max}$ is the upper boundary for Δ_t. $\Delta_{\min}$ represents the minimum adjustment step that avt can use. $\Delta_{\min}$ is also called *precision* since avt can only guarantee that it will approximate the $v^\star$ value within a margin of $\pm\Delta_{\min}$: $\Delta_{\min} \geq |v_t - v^\star|$. $\Delta_{\max}$, on the other hand, represents the maximum adjustment step that avt can use and as a result it is the *maximum evolution speed* of v_t.

During search process, the more the avt receives successive feedbacks of the *same direction*, the more v_t is supposed to be far away from $v^\star$ (or worse: v_t is moving far away from $v^\star$). It is then necessary to *accelerate* the adjustment of v_t to reach $v^\star$ more quickly. Consequently, at each feedback receipt, the adjustment step is increased as

$$\Delta_{t+1} = \Delta_t \cdot \lambda_{incr}. \tag{6}$$

On the contrary, the more avt receives successive feedbacks of *opposite directions*, the more v_t is oscillating around a more or less stable $v^\star$ value.[7] To get closer to this latter value (not to jump to a further value suddenly) and thus to reach $v^\star$ more quickly, it is necessary to *decelerate* the adjustment.

[4] i.e., a searched value that may change in the time due to the dynamics of the system. For example, considering WSNs, the right rate value can be affected by changing temperatures.

[5] Recalling Section IV, the environment of avt_u is its sensor node u.

[6] Note that since v_{t+1} cannot exceed the boundary values of AVT_{ss}, if $v_t + \Delta_{t+1} > v_{\max}$ then $v_{t+1} = v_{\max}$ and if $v_t + \Delta_{t+1} < v_{\min}$ then $v_{t+1} = v_{\min}$.

[7] Assuming that the feedbacks coming from the environment are correct.

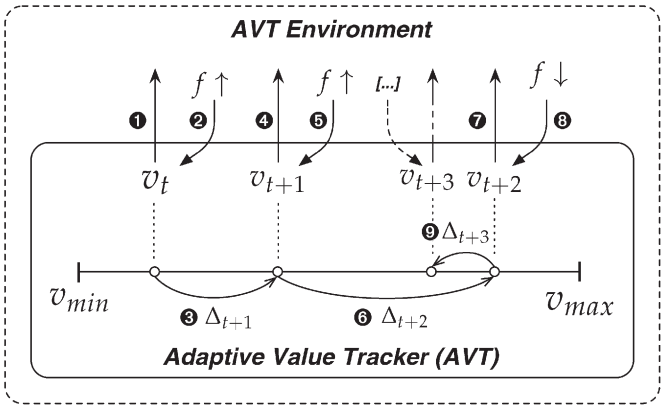


Fig. 2. Interaction between an AVT and its environment. The adaptive value tracking process starts with an initial value v_0 and includes several cycles of search iteration: e.g., (1) the AVT proposes a value v_t to its environment, (2) the environment sends a feedback $f\uparrow$, (3) from this feedback (and from the previous feedback), the AVT determines the best tuning step Δ_{t+1} to reach a value $v_{t+1} = v_t + \Delta_{t+1}$ that is as close as possible to $v^\star$, (4) the new value v_{t+1} is proposed to the environment, (5) then another feedback $f\uparrow$ is sent by the environment, (6) from this feedback the AVT determines to go further by increasing its tuning step by Δ_{t+2}, (7) the new value v_{t+2} is proposed to the environment, (8) however this time an opposite feedback $f\downarrow$ is sent by the environment, (9) consequently the AVT reduces the tuning step and moves toward the opposite direction, then another new value is proposed and the process goes on till $v^\star$ is reached.

Consequently, at each feedback receipt, the adjustment step is decreased as

$$\Delta_{t+1} = \Delta_t \cdot \lambda_{decr}. \tag{7}$$

Lastly, when avt receives a good feedback, this means that v_t has reached a (at least, briefly) correct value. Consequently, the value v_{t+1} remains the same as $v_{t+1} = v_t$ since $v^\star$ is probably close to v_t. The adjustment step, on the other hand, is decreased as presented in equality (7), since in the next step if a different feedback is received, $v^\star$ is probably less far away from the value v_{t+1}.

The above search mechanism is illustrated in Fig. 2. As shown in this figure, the Δ_t value reflects the occurrence of successive feedbacks and the decision of increasing or decreasing Δ_t is taken from the last two feedbacks.

As a result of this process, it can be shown that after a sufficient number of iterations the proposed value and the adjustment step value at time $t^\star$ satisfy

$$v^\star - \Delta_{\min} \leq v_{t^\star} \leq v^\star + \Delta_{\min}, \tag{8}$$

$$\Delta_{t^\star} = \Delta_{\min}. \tag{9}$$

It should be noted that even if the dynamic value $v^\star$ changes somehow after reaching this state, avt continues to track this value via the feedbacks coming from its environment.

A. Performing Fast Search Using AVTs

In this subsection, we answer the important question of what the optimal values for λ_{incr} and λ_{decr} are for the fastest search. We assume that the searched value $v^\star$ is static and we try to find the values of λ_{incr} and λ_{decr} which force AVT to perform a *dichotomic* search. A dichotomy is a binary search method that splits the search area into two non-overlapping parts and processes by eliminating one of these parts. In the literature dichotomy methods are known to have fairly good convergence, yielding close values in logarithmic time ([15] referring to [16]).

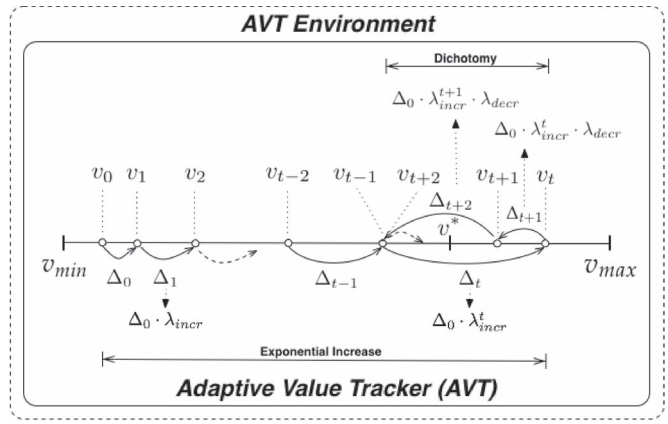


Fig. 3. Starting from an initial value v_0 and an initial adjustment step Δ_0, the AVT first exponentially changes the value until it crosses the searched value v^*, and then starts a dichotomy around v^*.

Consider the case shown in Fig. 3, such that the initial value proposed by AVT is $v_0 < v^*$. In the beginning of the search process, the successive feedbacks are identical, and consequently the proposed value comes closer to the objective value v^* with exponentially increasing steps, which are factors of λ_{incr}. Assume that at a time t the objective value v^* is crossed. In this case, the search process continues in the opposite direction and the adjustment value Δ_t is decreased by a factor λ_{decr}. Then the proposed value approaches the objective value v^* again with a factor of λ_{incr} and changes its direction each time it crosses the objective value v^*. This process continues until v^* is reached. In the process described here, if $\lambda_{decr} = 1/(1+\lambda_{incr})$ holds,[8] the process starting at t is a dichotomy with intervals of proportion $1/1+\lambda_{incr}$ and $\lambda_{incr}/1+\lambda_{incr}$.

We now focus on the derivation of time t as a function of the distance between the objective value and the proposed value v_t. We have for v_t that

$$\begin{aligned} v_t &= v_0 + \Delta_0 \left(1 + \lambda_{incr} + \lambda_{incr}^2 + \cdots + \lambda_{incr}^{t-1}\right) \\ &= v_0 + \Delta_0 \frac{1-\lambda_{incr}^t}{1-\lambda_{incr}}. \end{aligned} \tag{10}$$

Since t is the first time such that $v_t \geq v^*$ holds, we get from (10) that

$$\lambda_{incr}^t \geq \frac{v^* - v_0}{\Delta_0}(\lambda_{incr} - 1) + 1. \tag{11}$$

It can also be shown that if $v_0 > v^*$ then $\lambda_{incr}^t \geq -((v^* - v_0)/\Delta_0)(\lambda_{incr} - 1) + 1$. By considering the two cases $v_0 < v^*$ (as in Fig. 3) and $v_0 > v^*$, we reach that

$$t' = \left\lfloor \log_{\lambda_{incr}} \left(\frac{|v^* - v_0|}{\Delta_0}(\lambda_{incr} - 1) + 1 \right) \right\rfloor. \tag{12}$$

[8]As shown in Fig. 3, $\Delta_t = \Delta_{t+1} + \Delta_{t+2}$ and thus $\Delta_0 \cdot \lambda_{incr}^t = \Delta_0 \cdot \lambda_{incr}^t \cdot \lambda_{decr} + \Delta_0 \cdot \lambda_{incr}^{t+1} \cdot \lambda_{decr}$. And this equation implies $\lambda_{decr} = 1/(1+\lambda_{incr})$.

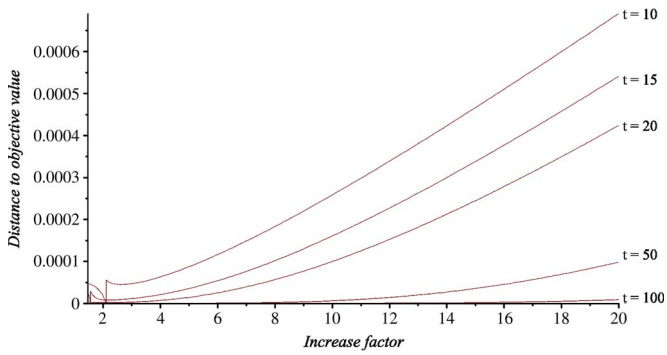


Fig. 4. Bounds on the distance to objective value for times $t = 10, 15, 20, 50, 100$ as a function of the increase factor λ_{incr}. Apparently, $\lambda_{incr} = 2$ is a satisfactory value for fast convergence.

It should be noted that as soon as the dichotomy process has started, $|v^* - v_0|$ at time $t' > t$ is bounded by

$$\Delta_0 \lambda_{incr}^t \left(\frac{\lambda_{incr}}{1 + \lambda_{incr}} \right)^{t'-t}. \tag{13}$$

In order to find the best value for λ_{incr} we studied the minima of these bounds, setting $v_0 = 0, \Delta_0 = 10^{-7,5}$ and $v^* = 5 \cdot 10^{-5}$ with the same order of magnitude as in our experimental setup (see Section VII). In these simulations, we traced the distance to objective value for different number of steps $t = 10, 15, 20, 50, 100$ as a function of λ_{incr} (see Fig. 4). The result of these simulations show that $\lambda_{incr} = 2$ is a satisfactory value for fast convergence. As a consequence, $\lambda_{decr} = 1/3$ is chosen.

Besides, as stated above, whenever $\lambda_{incr} > 1$ and $\lambda_{decr} = 1/(1 + \lambda_{incr})$ an AVT converges to its objective value for any initial value v_0 and any initial step size $\Delta_0 > 0$. Consequently, for any finite number of wrong feedbacks, an AVT still converges to its objective value since it can be consider that the AVT is restarting its search as $v_0' = v_{t_e}$ and $\Delta_0' = \Delta_{t_e}$ where t_e is the time instant of the last erroneous feedback.

VI. Comparison of Time Synchronization With AVT and Least-Squares

In this section, we provide a theoretical comparison of AVTS protocol and least-squares based protocols. To this end, we consider a simple synchronization scenario between two sensor nodes. Let r and u be two nodes such that r is the reference node with a perfect clock, i.e., $h_r = 1$. For simplicity, assume that these two nodes are powered on at the same time, there is no transmission delay between them, there are no quantization errors in the system and the hardware clock rate h_u is constant. Let t_k, where $k = 0, 1, \ldots,$ are the time instants such that node u receives synchronization messages from r. Let t_k^+ denotes the time instant just after t_k. Since the reference node r has a perfect clock, it holds that $t_k = kB$. We focus on the *actual synchronization error* between the reference node r and node u, which is defined as

$$e_u(t_k) = L_u(t_k) - L_r(t_k) = L_u(t_k) - kB. \tag{14}$$

Whenever node u receives a message from r, it calculates the *estimated synchronization error* as

$$\hat{e}_u(t_k) = L_u(t_k) - \hat{L}_r(t_k) \tag{15}$$

where $\hat{L}_r(t_k)$ is the received clock value.[9] Since there is no transmission delay, $\hat{L}_r(t_k) = L_r(t_k)$ and $e_u(t_k) = \hat{e}_u(t_k)$.

According to Algorithm 1, upon node u receives a message from r at time t_k, $\Delta_u(t_k^+)$ is calculated as follows:

$$\Delta_u\left(t_k^+\right) = \Delta_u(t_k)\mathcal{F}_u(t_k) \tag{16}$$

where $\mathcal{F}_u(t_k)$ is the adjustment function which returns λ_{incr}, λ_{dec} or 1 by considering the error functions $\hat{e}_u(t_k)$ and $\hat{e}_u(t_{k-1})$, as we defined in Section V. By considering $\Delta_u(t_k^+)$, the rate multiplier $l_u(t_k^+)$ is updated as follows:

$$\begin{aligned} l_u\left(t_k^+\right) &= l_u(t_k) + \mathbf{1}_u(t_k)\Delta_u\left(t_k^+\right) \\ &= l_u(t_k) + \mathbf{1}_u(t_k)\Delta_u(t_k)\mathcal{F}_u(t_k) \end{aligned} \tag{17}$$

where $\mathbf{1}_u(t_k)$ is a function which is defined as

$$\mathbf{1}_u(t_k) = \begin{cases} \hat{e}_u(t_k) < 0 & 1 \\ \hat{e}_u(t_k) > 0 & -1 \\ \hat{e}_u(t_k) = 0 & 0. \end{cases} \tag{18}$$

By considering (17), we have for $e_u(t_k)$ that

$$\begin{aligned} e_u(t_k) &= L_u(t_k) - kB \\ &= Bh_u\left(l_u\left(t_{k-1}^+\right) - l_u(t_{k-1})\right) \\ &\quad + L_u(t_{k-1}) - (k-1)B \\ &= Bh_u\left(l_u\left(t_{k-1}^+\right) - l_u(t_{k-1})\right) + e_u(t_{k-1}) \\ &= Bh_u\mathbf{1}_u(t_{k-1})\Delta_u(t_{k-1})\mathcal{F}_u(t_{k-1}) \\ &\quad + e_u(t_{k-1}). \end{aligned} \tag{19}$$

With an abuse of notation, we denote t_k by k and the overall evolution of the system can be written in the matrix form as follows:

$$\begin{bmatrix} e_u(k+1) \\ l_u(k+1) \\ \Delta_u(k+1) \end{bmatrix} = \mathcal{A}(k) \begin{bmatrix} e_u(k) \\ l_u(k) \\ \Delta_u(k) \end{bmatrix} \tag{20}$$

where the system matrix $\mathcal{A}(k)$ is defined as

$$\mathcal{A}(k) = \begin{bmatrix} 1 & 0 & Bh_u\mathbf{1}_u(k)\mathcal{F}_u(k) \\ 0 & 1 & \mathbf{1}_u(k)\mathcal{F}_u(k) \\ 0 & 0 & \mathcal{F}_u(k) \end{bmatrix}. \tag{21}$$

The entries of the system matrix $\mathcal{A}$ are time-dependent constants and the AVT system is an *almost linear* dynamical system. As explained in the beginning of this section, $\Delta_{\min}$ is the precision of the search and after a finite number of steps, Δ_u will converge to $\Delta_{\min}$. Hence, $\lim_{h\to\infty} \Delta_u(t_k^+) = \Delta_u(t_k)\mathcal{F}_u(t_k) = \Delta_{\min}$ and the rate multiplier will have value within a margin of $\pm\Delta_{\min}$: $\Delta_{\min} \geq |v_t - v^\star|$.

[9] $\hat{e}_u(t_k)$ corresponds to the *skew* in Algorithm 1.

Now, assume that there are *measurement errors* in the system due to factors such as transmission delay and quantization. Hence, the received clock values from the reference node are not completely accurate, i.e., $\hat{L}_r(t_k) = kB + \gamma_{r,u}(t_k) + q_r(t)$ such that $\gamma_{r,u}(t_k)$ is the error of the logical clock due to the transmission delay between r and u, and $q_r(t)$ is the quantization error of the hardware clock H_r at time t. Similarly, $\hat{L}_u(t_k) = L_u(t_k) + q_u(t_k)$ due to the quantization error of the hardware clock H_u at time t. Thus, $\hat{e}_u(t_k) = e_u(t_k) + q_u(t_k) - \gamma_{r,u}(t_k) - q_r(t)$. Since $\hat{e}_u(t_k)$ is inaccurate, the output of the adjustment function $\mathcal{F}_u(t_k)$ may introduce errors. It can be observed that, as long as $|e_u(t_k)| > |q_u(t_k) - \gamma_{r,u}(t_k) - q_r(t)|$, $\mathcal{F}_u(t_k)$ is not affected by the transmission delays. Hence, the system without transmission delays is identical to the system with transmission delays. By considering this observation, during the period such that $|e_u(t_k)| > |q_u(t_k) - \gamma_{r,u}(t_k) - q_r(t)|$ holds, the equality (19) becomes

$$\begin{aligned} e_u(t_k) &= L_u(t_k) - kB \\ &= Bh_u\left(l_u\left(t_{k-1}^{+}\right) - l_u(t_{k-1})\right) + e_u(t_{k-1}) \\ &\quad + q_u(t_{k-1}) - \gamma_{r,u}(t_{k-1}) - q_r(t_{k-1}) \\ &= Bh_u\mathbf{1}_u(t_{k-1})\Delta_u(t_{k-1})\mathcal{F}(t_{k-1}) + e_u(t_{k-1}) \\ &\quad + q_u(t_{k-1}) - \gamma_{r,u}(t_{k-1}) - q_r(t_{k-1}). \end{aligned} \tag{22}$$

By considering the equality above, the system evolution can be written as follows:

$$\begin{bmatrix} e_u(k+1) \\ l_u(k+1) \\ \Delta_u(k+1) \end{bmatrix} = \mathcal{A}(h) \begin{bmatrix} e_u(k) \\ l_u(k) \\ \Delta_u(k) \end{bmatrix} - \mathcal{B}(k) \tag{23}$$

where the error matrix $\mathcal{B}(k)$ is defined as

$$\mathcal{B}(k) = \begin{bmatrix} q_u(k) - \gamma_r(k) - q_r(k) \\ 0 \\ 0 \end{bmatrix}. \tag{24}$$

As can be observed from the matrix description of the system, *the errors enter linearly to the system evolution*.

Consider the least-squares regression, which assumes a linear relationship between the $H_u(t_k)$ and $L_r(t_k)$ pairs, i.e., $L_r(t_k) = \alpha + \beta H_u(t_k) + \varepsilon$ such that $\varepsilon \sim \mathcal{N}(0, \sigma^2)$. If we denote the N collected points as (x_i, Y_i) for $i = 0, \ldots, N-1$, if we let $\overline{x} = \sum x_i/N$, $\overline{Y} = \sum Y_i/N$, $S_{xx} = \sum(x_i - \overline{x})^2$ and $S_{xY} = \sum(x_i - \overline{x})(Y_i - \overline{Y})$, the estimated least-squares line is calculated as $\hat{Y} = \hat{\alpha} + \hat{\beta}x$, where $\hat{\beta} = S_{xY}/S_{xx}$ is the *slope* and $\hat{\alpha} = \overline{Y} - \hat{\beta}\overline{x}$ is the *intercept*. Assume that node u stores the two most recent time information of the reference node. For this case, i.e., $\mathrm{N} = 2$, we have

$$\hat{\beta}(k) = \frac{L_r(k) - L_r(k-1)}{H_u(k) - H_u(k-1)}. \tag{25}$$

When measurement errors are present in the system, the slope equation above can be rewritten as follows:

$$\begin{aligned} \hat{\beta}(k) = &\left(L_r(k) + q_r(k) + \gamma_{r,u}(k) - L_r(k-1)\right. \\ &\left. - q_r(k-1) - \gamma_{r,u}(k-1)\right) \\ &/\left(H_u(k) - H_u(k-1) + q_u(k) - q_u(k-1)\right). \end{aligned} \tag{26}$$

As can be observed, *the errors enter non-linearly to the system evolution*. This reality also holds for the intercept equation.

Based on our analysis, we list the following facts about time synchronization with AVT:

- When multi-hop time synchronization is considered, nodes send the value of their logical clocks for the other nodes. Since the errors enter to the system evolution linearly in AVT, these errors are *additive* at each hop. Thus, the synchronization error grows with the *square root* of the network diameter in AVTS while it is *exponential* if least-squares regression is employed.[10]
- AVT converges in a finite amount of time while least-squares regression converges in a fixed amount of time, as we have shown in Section V-A. Hence, AVTS has slower convergence time when compared to the least-squares based solutions.
- AVT stores only two variables $l_u(k)$ and $\Delta_u(k)$ in the main memory to calculate the rate multiplier of the logical clock. On the other hand, least-squares regression requires a table of N entries in the main memory to store $(H_u(k), L_r(k))$ pairs for this calculation. As a consequence, AVTS has lower memory overhead for time synchronization.
- AVTS has lower computation overhead when compared to least-squares based time synchronization. AVT performs only 1 addition and 1 multiplication to calculate the rate multiplier of the logical clock. On the other hand, least-squares regression requires $2N$ subtractions, $2N$ additions, $2N$ multiplications and 3 divisions to calculate slope $\hat{\beta}$.

VII. Testbed Experiments and Simulations

In this section, the real-world performances of the flooding based time synchronization protocols we mentioned in this article are compared by presenting their experimental results collected from a small testbed of 20 sensor nodes. Our evaluation metrics were the instantaneous *global skew*, *local* skew, *average global skew* and *average local skew* values, as we defined in Section III. For larger networks, we present an evaluation based on simulation results to make a comparison of these protocols in terms of their scalability.

A. Hardware Platform

The hardware platform used for the implementation is MICAz sensor nodes from Memsic.[11] The microcontroller included in this platform is 8-bit Atmel Atmega128L microcontroller which has 4 kB RAM and 128 kB flash memory. The transceiver on the MICAz board is Chipcon CC2420 radio chip which provides a 250 kb/s data rate at 2.4 GHz frequency. We used 7.37 MHz quartz oscillator on the MICAz board as the clock source for the timer used for timing measurements. The timer operates at 1/8 of that frequency and thus each timer tick occurs at approximately every 921 kHz, i.e., approximately

[10]The exponential behavior of the least-squares based time synchronization has also been shown in [4].

[11]http://www.memsic.com/, last access on 27 September 2013.

1 microsecond. The CC2420 transceiver has the capability to timestamp packets using this timer. As a final detail, oscillators of MICAz nodes are reported to exhibit a drift of ± 40 ppm [17].

B. Implementation Details

We carried out our experiments using publicly available implementations of FTSP and FCSA. On the other hand, we implemented AVTS and PulseSync by ourselves. The implementation of the aforementioned protocols has been done on top of TinyOS operating system [18] that is designed for networked embedded systems. nesC [19], a component-based programming language which was also used for the implementation of TinyOS and its applications, has been used for these implementations. The nesC compiler, ncc, avoids dynamic memory allocation, detects and eliminates dead-codes, reduces the code size and prevents most of concurrency related race conditions by static analysis. The output of ncc is a C language file which is compiled by the C compiler of the target microcontroller to produce a target object file. We did not perform any explicit optimization other than the optimizations performed by ncc and avr-gcc which is the C compiler for Atmel microprocessors.

An implementation detail worth mentioning here is the way we compensated the non-deterministic error sources during communication to achieve high precision time synchronization. MAC layer timestamping is a common method used for this compensation to increase the quality of time synchronization [1], [6], [20]–[22]. For the MAC layer time-stamping, we used *packet level time synchronization* interfaces [23] provided by TinyOS. The compensation mechanism employed by these interfaces can be summarized as follows. When any node u has decided to send its time information via a packet, it timestamps this event at real-time t_e by storing $H_u(t_e)$ as a timestamp. Then, it sends a request to the MAC layer. When the MAC layer accesses the wireless channel and the packet starts being transmitted, a corresponding event at real-time t_{tx} is timestamped with $H_u(t_{tx})$ by the radio chip. Then the difference $H_u(t_{tx}) - H_u(t_e)$, how much time ago the event had occurred, is stored at the timestamp field of the packet. When any node v starts receiving the packet at real-time t_{rx}, the packet is timestamped with the local clock at reception, i.e., $H_v(t_{rx})$. Then, the receiver node v adds $H_u(t_{tx}) - H_u(t_e)$ to compensate for the non-deterministic error sources, and can calculate $H_v(t_{rx}) - H_u(t_{tx}) - H_u(t_e)$ which is the approximate time of the event t_e in the node v's local clock.

We did not perform any additional compensation mechanism other than the MAC layer time-stamping employed by TinyOS. It should be noted that the compensation error of this mechanism is $(h_u - h_v)(t_{tx} - t_e)$. As can be observed, if $(t_{tx} - t_e) \leq 10^5$ μ sec $= 100$ ms, then the compensation error will be at most 8 μsec in the worst case, i.e., 8 ticks, in MICAz platform since the maximum drift is between ± 40 ppm, i.e., $h_u - h_v \leq 8 \times 10^{-5}$.

C. Testbed Setup

Our experiments were performed on a testbed of 20 MICAz sensor nodes. These sensor nodes are placed in the communication range of a *query broadcaster* sensor node as shown in Fig. 5. We constructed line and grid topologies with the help of the software by configuring each node such that it will accept incoming messages from the nodes which are its neighbors in the corresponding topology. Experiments on the line topology allowed us to observe the scalability of the protocols since the performance of flooding based time synchronization degrades as the diameter of the network increases [4]. With the experiments on the grid topology, we could evaluate the performances of the protocols under contention, congestion and increased packet collisions.[12]

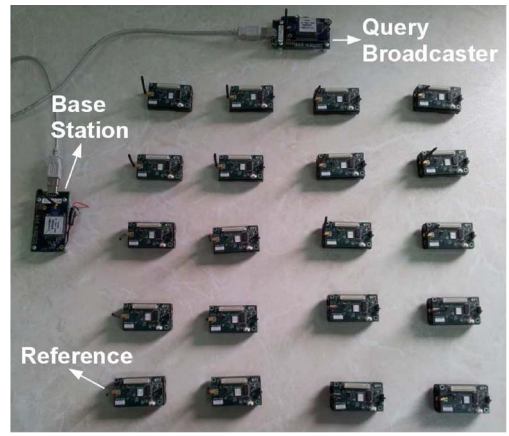


Fig. 5. Testbed setup for the experiments: 20 MICAz sensor nodes (incuding the *reference* node of the time synchronization protocol), a *query broadcaster* node that periodically transmits query packets, a *base station* that collects and transfers logical clock values to the serial port of our PC.

TABLE I
PARAMETERS OF AVTS USED DURING THE EXPERIMENTS

$\mathbf{B}$	δ	$\mathbf{v_{min}}$	$\mathbf{v_{max}}$	$\mathbf{\Delta_{min}}$	$\mathbf{\Delta_{max}}$
30 seconds	0	-10^{-4}	10^{-4}	10^{-10}	10^{-5}

To collect instantaneous logical clock values from sensor nodes, the query broadcaster transmits query packets periodically. The interval between successive query packets is uniformly distributed between 20 and 23 seconds. Each of these packets are received approximately at the same time by all nodes. Upon receiving a query packet, each sensor node responds with a *reply* packet which carries the value of its logical clock. A *base station* node listens the reply packets and transfers them to the serial port of our PC for the logging of the logical clock values. At the end of the experiments, the evaluation metrics are applied to the collected data and the results are analyzed.

D. Protocol Parameters

Each of our experiments took approximately 20000 seconds (approximately 5.5 hours). We powered on sensor nodes randomly in the first 3 minutes. The evaluated protocols had an identical beacon period of 30 seconds (B=30 seconds). The

[12]Remembering that each sensor node sends feedbacks to its adaptive value tracker upon receiving a synchronization message, it can be said that each packet collision causes a feedback loss.

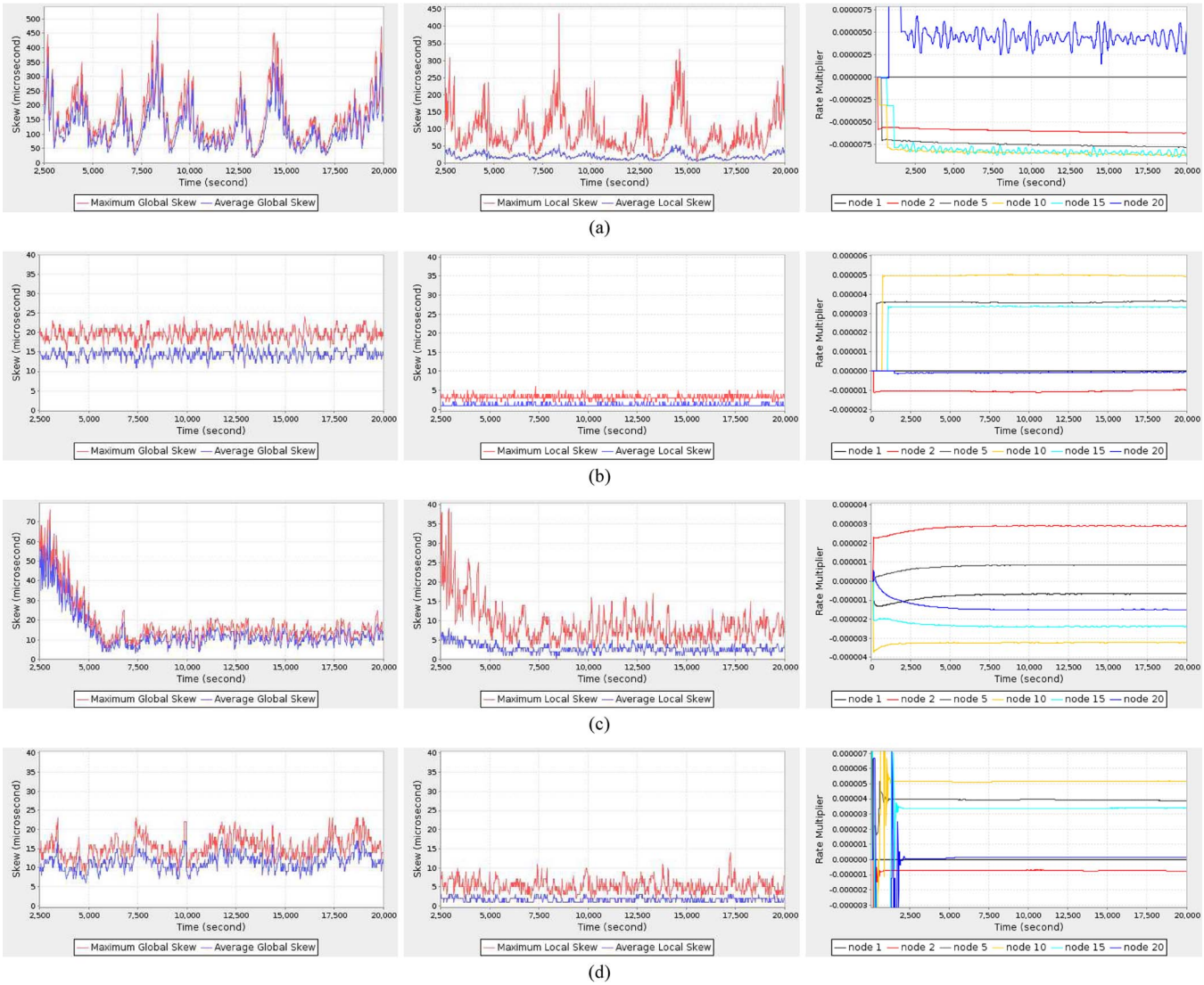


Fig. 6. Global skew (left column), local skew (middle column) and rate multipliers (from which 1.0 is subtracted) on the line topology for FTSP, PulseSync, FCSA and AVTS, respectively. (a) FTSP; (b) PulseSync; (c) FCSA; (d) AVTS.

number of entries in the protocol tables of FTSP, PulseSync and FCSA, e.g., regression table and neighbor repository, was 8. The parameters of the AVTS used for the experiments are presented in Table I. The tolerance value δ in Algorithm 1 was set to zero, because we require the synchronization error to be as small as possible. Since the hardware clocks are reported to have a drift of ± 100 ppm, i.e., 10^{-4} seconds, we defined the upper bound and lower bounds of the searched logical clock rate as $v^{\star} \in [v_{\min}; v_{\max}] = [-10^{-4}, 10^{-4}]$. We adjusted the upper and lower bounds for the adjustment step Δ_t as $\Delta_t \in [\Delta_{\min}, \Delta_{\max}] = [10^{-10}, 10^{-5}]$ since a precision of 10^{-10} for the searched logical clock rate is sufficient to achieve tight synchronization. These parameters of adaptive value tracker can also be verified empirically.

E. Experimental Results

Fig. 6 presents the maximum and average values of global and local synchronization skews and the rate multipliers during the experiments on the line topology. Table II summarizes these results. Even if a line topology of 20 sensor nodes is a small network, we observed more than 0.5 milliseconds maximum global and local skew values with FTSP. This observation is quite consistent with the experimental results presented in [4], [7], [8]. The least-squares regression together with the slow-propagation of time information led to quite unstable rate multipliers as the distance from the reference node increases. Hence, nodes poorly determine the relative speed of the reference clock with respect to the speed of their hardware clocks, which also leads to poor synchronization quality. As we showed theoretically in Section VI, FTSP employing least-squares for

TABLE II
SUMMARY OF THE MEASURED SKEW VALUES FOR FTSP, PULSESYNC, FCSA AND AVTS ON THE LINE TOPOLOGY

	Line Topology			
	FTSP	**PulseSync**	**FCSA**	**AVTS**
Max. Global	518 μsec	24 μsec	25 μsec	23 μsec
Max. Avg. Global	422 μsec	18 μsec	20 μsec	18 μsec
Max. Local	437 μsec	6 μsec	16 μsec	14 μsec
Max. Avg. Local	55 μsec	2 μsec	5 μsec	3 μsec

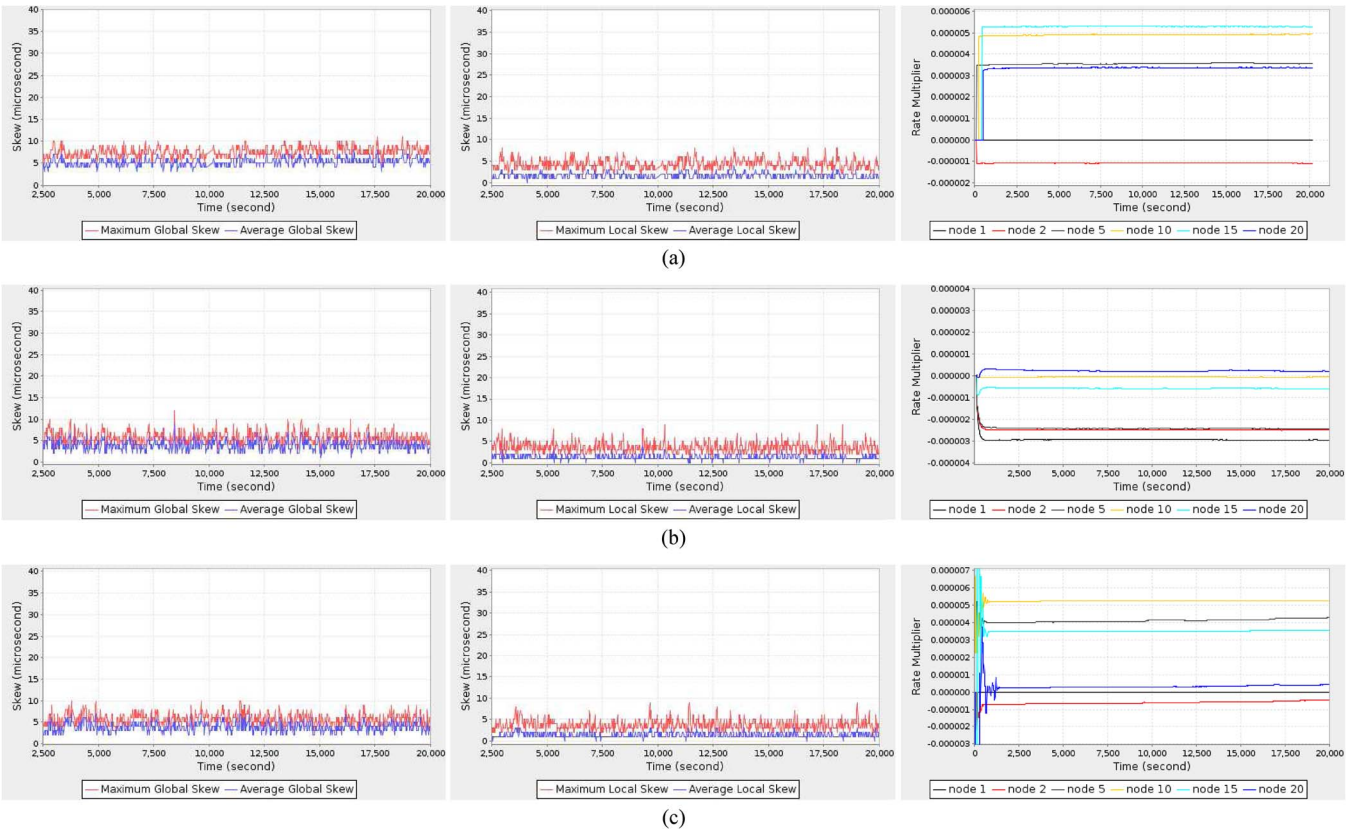


Fig. 7. Global skew (left column), local skew (middle column) and rate multipliers (from which 1.0 is subtracted) on the 5 × 4 grid topology for FTSP, PulseSync, FCSA and AVTS, respectively. (a) PulseSync; (b) FCSA; (c) AVTS.

multi-hop time synchronization exhibits a synchronization error growing exponentially with the network diameter.

When PulseSync is considered, it can be noticed that the rapid-flooding strategy leads to a drastic improvement in the synchronization quality. Since the time information of the reference node is propagated through reliable and fast pulses, the error of the multi-hop time synchronization with PulseSync grows with the square root of the network diameter [4]. Hence, the rate multipliers are quite stable and their variations are reduced. Forwarding the pulses from the reference node with the rapid-flooding approach prevented the error of the propagated time information from being amplified at each hop, hence reduced both global and local skew values. On the other hand, experiments with FCSA showed that the improvement of the synchronization quality achieved with rapid-flooding can also be achieved by employing a slow-flooding strategy together with a distributed agreement strategy. It can be observed that the synchronization performances of FCSA and PulseSync are nearly identical in terms of their global skew. Besides, PulseSync exhibited smaller local skew values when compared to FCSA, as expected. Since nodes collect the recent time information and update their logical clocks more quickly with rapid-flooding, the occurrence of large instant local skews is eliminated with PulseSync.

Among the aforementioned protocols, AVTS achieved the best performance with its simple approach during our experiments on the line topology. We observed that the rate multipliers, i.e., the rates of the logical clocks, were quite stable with AVTS. Hence, the adaptive value tracking mechanism performed pretty good performance and the nodes were able to estimate the relative clock rate of the reference node with respect to the rate of their hardware clocks quite well. Due to the high performance of this estimation as we presented in Section VI, local and global skew values of AVTS were similar to that of PulseSync although AVTS propagates time information slowly. A desirable property of AVTS compared to FCSA is that tight network-wide synchronization is established quickly. It took approximately 7500 seconds for FCSA to reach the performance of AVTS. Before the convergence, the rate multiplier values of AVTS are fluctuating a lot. The reason is that, in the beginning of the experiments, the rates of the logical clocks are not synchronized and they have big skews. When unsynchronized sensor nodes forward their logical values to their neighboring nodes, the receiver nodes get the time information of the reference node with big errors and hence send wrong feedbacks to their AVTs. It should be noted that during this initial synchronization period, nodes which are closer to the reference node get more accurate time information when compared to the far-away nodes. As time passes, the synchronization is achieved hop by hop. Thus, far-away nodes start to get more accurate time information of the reference node, they start to send correct feedbacks to their AVTs and hence they get synchronized.

To compare the performances of FCSA, PulseSync and AVTS under increased contention and packet collisions, we also performed experiments on a 5 × 4 grid topology. Fig. 7 presents the maximum skew values and the rate multipliers on this topology. Table III summarizes these results. It can be observed

TABLE III
SUMMARY OF THE MEASURED SKEW VALUES FOR PULSESYNC, FCSA, AND AVTS ON THE 5 × 4 GRID TOPOLOGY

	5x4 Grid Topology		
	PulseSync	**FCSA**	**AVTS**
Max. Global	11 μsec	12 μsec	10 μsec
Max. Avg. Global	8 μsec	9 μsec	7 μsec
Max. Local	8 μsec	9 μsec	9 μsec
Max. Avg. Local	4 μsec	4 μsec	3 μsec

TABLE IV
MEMORY REQUIREMENTS, CPU OVERHEAD, AND SYNCHRONIZATION MESSAGE LENGTH OF FTSP, PULSESYNC, FCSA AND AVTS DURING THE EXPERIMENTS. $|\mathcal{N}|$ IS USED TO DENOTE MAXIMUM NEIGHBORHOOD CARDINALITY

	FTSP	**PulseSync**	**FCSA**	**AVTS**
CPU overhead	$\approx$ 5440 μs	$\approx$ 5440 μs	5620 μs for $\|\mathcal{N}\| = 1$	$\approx$ 175 μs
Message Length	9 $bytes$	9 $bytes$	19 $bytes$	9 $bytes$
Main Memory Overhead	40 $bytes$	40 $bytes$	64*$\|\mathcal{N}\|$ $bytes$	9 $bytes$
Flash Memory Requirements	18000 $bytes$	17856 $bytes$	20660 $bytes$	15696 $bytes$

that the performances of these protocols are nearly identical, but with AVTS we observed a slight decrease on the clock skew. We think that even in this small network, rapid-flooding suffered from packet collisions and losses since it requires the time information to be propagated reliably. Due to the increased neighbor cardinality, flooding the time information rapidly is more difficult on the grid topology than on the line topology. Since AVTS does not demand rapid-flooding, its performance on the grid topology is quite comparable to its performance on the line topology.

In the light of our experiments, we conclude that AVTS performs tight synchronization without demanding rapid propagation of time information and increasing communication frequency. In the next subsection, we present the superiority of AVTS over the existing approaches in terms of energy and memory requirements.

F. Energy Consumption and Memory Requirements

Previously, we focused on the synchronization quality of the protocols in consideration and evaluated them under identical communication frequencies. In this subsection, we evaluate them in terms of their CPU overhead, memory requirements and the length of the synchronization messages. The results collected from our testbed are presented in Table IV.

To compare the overall energy requirements of these protocols, it is required to determine their CPU overhead. From the time where a recent synchronization message is received to the time it is processed and time information is updated, AVTS required drastically smaller CPU processing time compared to other protocols. For instance, a node with only one neighbor consumed at most 5620 microseconds CPU time with FCSA and this requirement increases as the number of neighbors

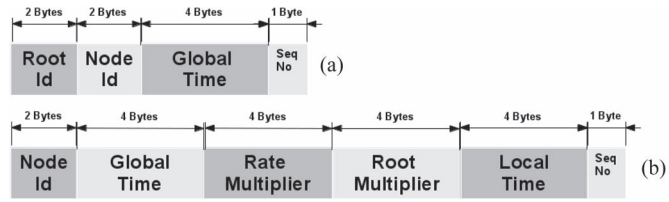


Fig. 8. Package field descriptions of the synchronization messages, (a) for FTSP, PulseSync, AVTS and (b) for FCSA, in their TinyOS implementations.

of this node increases. On the other hand, AVTS consumed approximately 175 microseconds! In conclusion, AVTS has approximately a factor of 31 lower CPU overhead compared to FTSP, PulseSync and FCSA, respectively.

Another property that effects the energy consumption of the protocol is the length of the synchronization messages. The reason is that the longer the synchronization messages are, the more time is required to transmit and receive them. As can be observed from Table IV and Fig. 8, AVTS requires the length of the synchronization messages to be 9 bytes, which is identical to that of PulseSync and FTSP and smaller than that of FCSA. Hence, AVTS does not increase the duration of the communication due to its small message length requirement. As a conclusion, AVTS can be characterized as being able to achieve high quality synchronization with low energy requirements compared to the existing methods.

Since memory is also a scarce resource for WSN nodes, we are also required to evaluate the memory requirements of time synchronization. The amount of memory required to store collected time information determines the major main memory (RAM) requirements of the protocols. Since FTSP and PulseSync employ the method of least-squares, they both require 40 bytes of memory to store the time information of the reference node and perform regression on the collected data. In FCSA, a least-squares table is allocated for each neighbor and hence together with the additional stored information, a total of 64 bytes of memory is allocated to keep track of a neighboring node. As a substantial superiority, AVTS does not require any table and it requires only 9 bytes of memory to achieve time synchronization. It can be concluded that AVTS decreased the amount of RAM required for time synchronization by approximately a factor of 4.5 compared to FTSP and PulseSync, and more than a factor of 7 compared to FCSA. Another scarce resource is the flash memory (ROM) which stores the program code of the applications. As presented in Table IV, AVTS reduced the code size of the application by more than 2160 bytes compared to the other approaches. This is also a remarkable gain for sensor nodes.

G. Simulation Results

To compare the scalability of the aforementioned flooding based protocols, we performed simulations in our WSN simulator which is a discrete event simulator implemented in Java language. Our simulator models the hardware clocks of nodes with a random drift of ± 50 ppm and the message delay with a normally distributed random variable. For MAC layer, we implemented a CSMA protocol inspired from the MAC protocol in TinyOS.

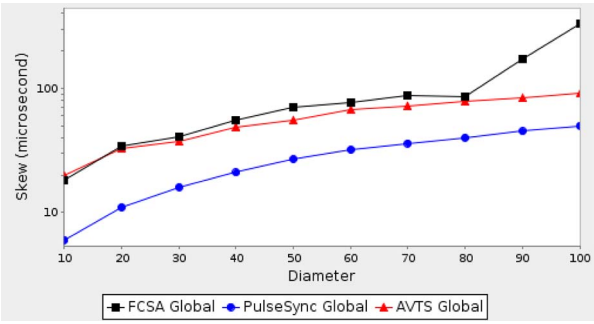


Fig. 9. Simulation results of FCSA, AVTS and PulseSync on different networks. The synchronization errors of FCSA, AVTS and PulseSync grow quite similarly with the network diameter.

We implemented PulseSync, FCSA and AVTS protocols in Java language to work with our simulator. The error of time synchronization is a function of the network diameter [4] and simulations on longer line topologies allowed us to observe the scalability of the protocols in the worst case. Hence, we constructed networks of diameter from 10 to 100 and performed 5 simulation runs for each network which simulated a real-world experiment of 55 hours. Then, we averaged the maximum global skew and local skew values of these runs.

Fig. 9 presents the results of our simulations. It can be observed that the performance of AVTS does not degrade quickly with the diameter of the network. As we presented in subsection VI, the errors enter the system dynamics of AVTS linearly and the global skew of AVTS grows with the square root of the network diameter (like as reported in [8] for FCSA and in [4] for PulseSync). Hence, we conclude that AVTS not only provides tight synchronization on a small real network of 20 sensor nodes, but also preserves its quality on longer networks, which makes it scalable.

It can also be observed that AVTS performed better than FCSA but slightly worse than PulseSync. The reason is that PulseSync disseminates time information fast to minimize transmission delays and maintains a history, i.e., a regression table, to reduce the effect of these errors. In AVTS, the logical clocks of the nodes are instantaneously updated to the received time information. Due to this update strategy, the logical clocks are more sensitive to the large message delays.

VIII. Related Work

Due to being a fundamental building block in distributed systems, time synchronization has received considerable attention from researchers. There are both theoretical and practical dimensions of time synchronization research in the literature. The main focus of the theoretical studies is to bound the synchronization error in any network and develop optimal distributed algorithms in terms of the proven bounds [24]–[37]. Under the light of the theoretical results, protocol based time synchronization studies led the research with the emergence of wireless sensor networks [1], [2], [4]–[6], [8], [17], [20], [21], [38]–[41]. A common method employed by most of these protocols is to flood stable time information of a reference node into the network [4]–[6], [8], [21], [38]–[41]. By means of establishing a relationship between their hardware clocks and the reference clock, sensor nodes can estimate future global clock values with a small error without communicating too frequently. The method of least-squares and occasionally distributed agreement [8] are the main techniques employed for this establishment in these protocols. Up to our knowledge, AVTS is the only protocol in WSN literature that puts least-squares regression and agreement techniques away from flooding based time synchronization by employing an adaptive value tracking mechanism.

Besides, until know, AVT approach has been successfully used in several projects such as maritime surveillance [42], [43], scenario control in games [44], self-organizing neural networks [45], [46], manufacturing control [47], user profiling [48]. Moreover, its use by engineers of UPETEC Ltd[13] in various industrial projects have shown good results. Recently, we have applied AVT for self-organizing and fully distributed time synchronization in WSNs [49]. This simulation based study does not present a real-world implementation. Moreover, we still do not have any mathematical proof of convergence for the algorithm in this paper and the convergence time of this algorithm is too long which makes it impractical.

IX. Conclusion

In this paper, we considered the recent flooding based time synchronization schemes in the WSN world. We revealed the major drawbacks of these schemes: demanding rapid flooding or keeping track of the neighboring nodes, even worse, a common shortcoming of having considerable overhead in terms of computation and memory allocation. We asked the question of whether it is possible to achieve high quality time synchronization without having these drawbacks.

We considered the problem of synchronization as a search process in which each sensor node is trying to find the rate of the reference clock without knowing its correct value. Due to the dynamics of the WSN environment, we employed an adaptive search technique for this search process. Within this context, we introduced the Adaptive Value Tracking Synchronization (AVTS) protocol, which employs an adaptive value tracking algorithm to synchronize the clock rates of the sensor nodes to that of a reference node through successive feedbacks. We presented an experimental evaluation of our approach, which exposed that tight synchronization can be achieved with low computational and memory overhead, with small memory code size and without demanding rapid flooding. In the light of the simulations, we observed that AVTS is also scalable.

We believe that AVTS brought a new dimension for time synchronization in WSNs by eliminating the major drawbacks of the existing studies without compromising synchronization quality. It would be interesting to observe the actual performance of AVTS on real-world deployments of applications which require microsecond precision time synchronization and we leave this issue as a future work.

Acknowledgment

The authors would like to thank Simon Stuker from Paul Sabatier University, Toulouse, France for his help and remarks.

[13] http://www.upetec.com/, last access on 20 September 2013.

REFERENCES

[1] P. Sommer and R. Wattenhofer, "Gradient clock synchronization in wireless sensor networks," in *Proc. Int. Conf. IPSN*, San Francisco, CA, USA, Apr. 2009, pp. 37–48.

[2] L. Schenato and F. Fiorentin, "Average timesynch: A consensus-based protocol for time synchronization in wireless sensor networks," *Automatica*, vol. 47, no. 9, pp. 1878–1886, Sep. 2011.

[3] R. Fan and N. Lynch, "Gradient clock synchronization," *Distrib. Comput.*, vol. 18, no. 4, pp. 255–266, Mar. 2006.

[4] C. Lenzen, P. Sommer, and R. Wattenhofer, "Optimal clock synchronization in networks," in *Proc. 7th ACM Conf. Embedded Netw. SenSys*, Berkeley, CA, USA, Nov. 2009, pp. 225–238.

[5] T. Schmid, Z. Charbiwala, Z. Anagnostopoulou, M. B. Srivastava, and P. Dutta, "A case against routing-integrated time synchronization," in *Proc. 8th ACM Conf. Embedded Netw. SenSys*, 2010, pp. 267–280.

[6] M. Maróti, B. Kusy, G. Simon, and A. Lédeczi, "The flooding time synchronization protocol," in *Proc. 2nd Int. Conf. Embedded Netw. SenSys*, 2004, pp. 39–49.

[7] K. S. Yildirim and A. Kantarci, "Drift estimation using pairwise slope with minimum variance in wireless sensor networks," *Ad Hoc Netw.*, vol. 11, no. 3, pp. 765–777, May 2013.

[8] K. Yildirim and A. Kantarci, "Time synchronization based on slow-flooding in wireless sensor networks," *IEEE Trans. Parallel Distrib. Syst.*, vol. 25, no. 1, pp. 244–253, Jan. 2014.

[9] J. Lu and K. Whitehouse, "Flash flooding: Exploiting the capture effect for rapid flooding in wireless sensor networks," in *Proc. IEEE INFOCOM*, Rio de Janeiro, Brazil, 2009, pp. 2491–2499.

[10] F. Ferrari, M. Zimmerling, L. Thiele, and O. Saukh, "Efficient network flooding and time synchronization with glossy," in *Proc. 10th Int. Conf. IPSN*, 2011, pp. 73–84.

[11] Y.-S. Kuo, P. Pannuto, T. Schmid, and P. Dutta, "Reconfiguring the software radio to improve power, price, portability," in *Proc. 10th ACM Conf. Embedded Netw. Sens. Syst.*, 2012, pp. 267–280.

[12] Y. Wang *et al.*, "Exploiting constructive interference for scalable flooding in wireless networks," in *Proc. IEEE INFOCOM*, 2012, pp. 2104–2112.

[13] Y. Wang, Y. He, D. Cheng, Y. Liu, X.-Y. Li, Triggercast: Enabling Wireless Collisions Constructive 2012, arXiv preprint arXiv:1208.0664.

[14] S. Lemouzy, V. Camps, and P. Glize, "Principles and properties of a mas learning algorithm: A comparison with standard learning algorithms applied to implicit feedback assessment," in *Proc. IEEE/WIC/ACM Int. Conf. WI-IAT*, 2011, vol. 2, pp. 228–235.

[15] I.-J. Wang, E. K. P. Chong, and R. Quong, "Sample complexity of continuous binary search with noisy information," in *Proc. 33rd IEEE Conf. Decision Control*, 1994, vol. 1, pp. 650–651.

[16] J. Traub, G. Wasilkowski, and H. Woźniakowski, *Information-Based Complexity*. New York, NY, USA: Academic, 1988, ser. Computer science and scientific computing.

[17] K. Sun, P. Ning, and C. Wang, "Tinysersync: Secure and resilient time synchronization in wireless sensor networks," in *Proc. 13th ACM Conf. CCS*, 2006, pp. 264–277.

[18] J. Hill, R. Szewczyk, A. Woo, S. Hollar, D. Culler, and K. Pister, "System architecture directions for networked sensors," *SIGPLAN Notices*, vol. 35, no. 11, pp. 93–104, Nov. 2000.

[19] D. Gay, P. Levis, R. von Behren, M. Welsh, E. Brewer, and D. Culler, "The NESC language: A holistic approach to networked embedded systems," in *Proc. ACM SIGPLAN Conf. PLDI*, 2003, pp. 1–11.

[20] J. Elson, L. Girod, and D. Estrin, "Fine-grained network time synchronization using reference broadcasts," *SIGOPS Oper. Syst. Rev.*, vol. 36, no. SI, pp. 147–163, 2002.

[21] S. Ganeriwal, R. Kumar, and M. B. Srivastava, "Timing-sync protocol for sensor networks," in *Proc. 1st Int. Conf. Embedded Netw. SenSys*, 2003, pp. 138–149.

[22] K. Römer, P. Blum, and L. Meier, "Time synchronization and calibration in wireless sensor networks," in *Handbook of Sensor Networks: Algorithms and Architectures*, I. Stojmenovic, Ed. Hoboken, NJ, USA: Wiley, 2005, pp. 199–237.

[23] M. Maroti and J. Sallai, Packet-level time synchronization, TinyOS Core Working Group, Tech. Rep., May 2008. [Online]. Available: http://www.tinyos.net/tinyos-2.x/doc/pdf/tep133.pdf

[24] J. Lundelius and N. A. Lynch, "An upper and lower bound for clock synchronization," *Inf. Control*, vol. 62, no. 2/3, pp. 190–204, Aug./Sep. 1984.

[25] J. Y. Halpern, N. Megiddo, and A. A. Munshi, "Optimal precision in the presence of uncertainty," in *Proc. 17th Annu. ACM STOC*, 1985, pp. 346–355.

[26] S. Biaz and J. L. Welch, "Closed form bounds for clock synchronization under simple uncertainty assumptions," *Inf. Process. Lett.*, vol. 80, no. 3, pp. 151–157, Nov. 2001.

[27] B. Patt-Shamir and S. Rajsbaum, "A theory of clock synchronization (extended abstract)," in *Proc. 26th Annu. ACM Symp. STOC*, 1994, pp. 810–819.

[28] T. K. Srikanth and S. Toueg, "Optimal clock synchronization," *J. ACM*, vol. 34, no. 3, pp. 626–645, Jul. 1987.

[29] R. Fan, I. Chakraborty, and N. A. Lynch, "Clock synchronization for wireless networks," in *Proc. Lect. Notes Comput. Sci.*, 2005, vol. 3544, pp. 400–414.

[30] T. Locher and R. Wattenhofer, "Oblivious gradient clock synchronization," in *Proc. 20th Int. Symp. DISC*, Stockholm, Sweden, Sep. 2006, pp. 520–533.

[31] C. Lenzen, T. Locher, and R. Wattenhofer, "Clock synchronization with bounded global and local skew," in *Proc. 49th Annu. IEEE Symp. FOCS*, Philadelphia, PA, USA, Oct. 2008, pp. 509–518.

[32] R. M. Pussente and V. C. Barbosa, "An algorithm for clock synchronization with the gradient property in sensor networks," *J. Parallel Distrib. Comput.*, vol. 69, no. 3, pp. 261–265, Mar. 2009.

[33] C. Lenzen, T. Locher, and R. Wattenhofer, "Tight bounds for clock synchronization," in *Proc. 28th ACM Symp. PODC*, Calgary, AB, Canada, Aug. 2009, pp. 46–55.

[34] F. Kuhn and R. Oshman, "Gradient clock synchronization using reference broadcasts," *CoRR*, vol. abs/0905.3454, 2009.

[35] C. Lenzen, T. Locher, and R. Wattenhofer, "Tight bounds for clock synchronization," *J. ACM*, vol. 57, no. 2, pp. 1–42, Jan. 2010.

[36] F. Kuhn, T. Locher, and R. Oshman, "Gradient clock synchronization in dynamic networks," in *Proc. 21st ACM SPAA*, Calgary, AB, Canada, Aug. 2009, pp. 270–279.

[37] F. Kuhn, C. Lenzen, T. Locher, and R. Oshman, "Optimal gradient clock synchronization in dynamic networks," in *Proc. 29th Symp. PODC*, Zurich, Switzerland, Jul. 2010, pp. 430–439.

[38] J. van Greunen and J. Rabaey, "Lightweight time synchronization for sensor networks," in *Proc. 2nd ACM Int. Conf. WSNA*, 2003, pp. 11–19.

[39] H. Dai and R. Han, "TSync: A lightweight bidirectional time synchronization service for wireless sensor networks," *ACM SIGMOBILE Mobile Comput. Commun. Rev.*, vol. 8, no. 1, pp. 125–139, Jan. 2004.

[40] B. Kusy, P. Dutta, P. Levis, M. Maroti, A. Ledeczi, and D. Culler, "Elapsed time on arrival: A simple and versatile primitive for canonical time synchronisation services," *Int. J. Ad Hoc Ubiquitous Comput.*, vol. 1, no. 4, pp. 239–251, 2006.

[41] T. Schmid, Z. Charbiwala, R. Shea, and M. Srivastava, "Temperature compensated time synchronization," *IEEE Embedded Syst. Lett.*, vol. 1, no. 2, pp. 37–41, Aug. 2009.

[42] N. Brax, E. Andonoff, M. Gleizes, and P. Glize, "Self-adapted aided decision-making: Application to maritime surveillance," in *Proc. 5th Int. Conf. Agents Artif. Intell.*, 2013, pp. 419–422.

[43] N. Brax, E. Andonoff, J.-P. Georgé, M. P. Gleizes, and J.-P. Mano, "MAS4At, un SMA auto-adaptatif pour le déclenchement d'alertes dans le cadre de la surveillance maritime," *Revue Intell. Artif.*, vol. 27, no. 3, pp. 371–395, 2013.

[44] L. Pons, C. Bernon, and P. Glize, "Scenario control for (serious) games using self-organizing multi-agent systems (regular paper)," in *Proc. ICCS*, Agadir, Morocco, 2012, pp. 1–6.

[45] Ö. Gürcan, K. S. Türker, J.-P. Mano, C. Bernon, O. Dikenelli, and P. Glize, "Mimicking human neuronal pathways in silico: An emergent model on the effective connectivity," *J. Comput. Neurosci.*, vol. 36, no. 2, pp. 235–257, Apr. 2014.

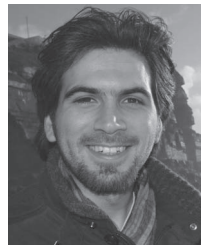
[46] O. Gürcan, C. Bernon, K. S. Türker, J.-P. Mano, P. Glize, and O. Dikenelli, "Simulating human single motor units using self-organizing agents," in *Proc. IEEE 6th Int. Conf. SASO*, Sep. 2012, pp. 11–20.

[47] E. Kaddoum and J.-P. Georgé, "Collective self-tuning for complex product design (short paper)," in *Proc. IEEE Int. Conf. SASO*, Lyon, France, 2012, pp. 193–198.

[48] S. Lemouzy, V. Camps, and P. Glize, "Real time learning of behaviour features for personalised interest assessment," in *Advances in Practical Applications of Agents and Multiagent Systems*, vol. 70, Y. Demazeau, F. Dignum, J. Corchado, and J. Pérez, Eds. Berlin, Germany: Springer-Verlag, 2010, ser. Advances in Soft Computing, pp. 5–14.

[49] Ö. Gürcan and K. S. Yildirim, "Self-organizing time synchronization of wireless sensor networks with adaptive value trackers," in *Proc. IEEE 6th Int. Conf. SASO*, Sep. 2013, pp. 91–100.

Kasım Sinan Yıldırım received the B.Sc.Eng, M.Sc., and Ph.D. degrees from Ege University, İzmir, Turkey, in 2003, 2006, and 2012, respectively. He works as an Assistant Professor at the same department. His research interests are embedded systems, distributed systems, distributed algorithms and wireless sensor networks.



Önder Gürcan received the B.Sc.Eng and an M.Sc. degrees in computer engineering from Ege University, İzmir, Turkey in 2004 and 2007, respectively. He then obtained two Ph.D. degrees, one in computer engineering from Ege University, İzmir, Turkey and the other one in artificial intelligence from Paul Sabatier University, Toulouse, France in 2013 under the conditions of co-tutelle agreement (the French joint Ph.D. protocol). He works as Post-Doctoral Researcher in the Laboratory of Model-driven Engineering for Embedded Systems (LISE) of the French Alternatve Energies and Atomic Energy Commission (CEA), Paris, France since December 2013. His main research interest is complex adaptive systems modeling and engineering of such systems.