High-Precision Low-Power Wireless Nodes' Synchronization via Decentralized Control

Alberto Leva, *Member, IEEE*, Federico Terraneo, Luigi Rinaldi,
Alessandro Vittorio Papadopoulos, *Member, IEEE*,
and Martina Maggio, *Member, IEEE*

***Abstract*— Time synchronization is crucial for wireless sensor networks (WSNs), where operations often rely on time ordering of events. WSNs are deployed in different scenarios, and therefore their timing requirements are often related to the peculiar characteristics of the specific environment they have to act in. Synchronization is anyway always an issue: transactional applications need monotonicity of the nodes' clocks to avoid time reversal, ultralow power applications call for minimal overhead to allow for low-duty-cycle operation, applications facing extreme environments have to maintain the needed precision in the presence of unforeseen thermal drift, and so on. Specially, control applications on battery-powered devices, where timing is an issue and low-power operation is highly desired, benefit from synchronization. However, to date, the problem of synchronization has been differently faced depending on the application domain. This paper proposes a general solution to the problem of synchronization in WSNs, which seamlessly integrates with the radio stack. In addition, it guarantees monotonic and continuous node clocks with low overhead for the infrastructure. The solution is based on a decentralized control scheme that is stable and robust to thermal stress, without the need for temperature measurements. The control scheme is simulated and implemented on real WSN nodes. The efficiency of the scheme is evaluated with simulations and experiments, providing insights on the maximum synchronization error between nodes, on the communication overhead, and on the limited nodes' power consumption. The solution is also compared with state-of-the-art alternatives.**

***Index Terms*— Decentralized control, linear control, low-power operation, time synchronization, wireless sensor networks (WSNs).**

I. Introduction

A NETWORK of connected devices, or wireless sensor networks (WSNs), has proven to be useful in a wide variety of scenarios, ranging from monitoring active volcanoes [1] to civil structures [2]. WSNs can provide technological solutions to a variety of different engineering problems, but come with their own challenges.

Two of these challenges are extremely difficult to solve. First, a radio channel cannot be as efficient as a wired bus at transmitting the signals required to keep the clock of the connected nodes synchronized. Second, nodes of a WSN are usually battery powered, and cannot afford to leave the radio transceiver always powered ON, not even in receive mode. This requires to minimize the time the nodes' transceivers should be active to guarantee synchronization. These challenges fostered the quest for accurate synchronization strategies that could offer low transmission and power overhead [3]–[11]. However, to date, none of the proposed solutions offers guarantees on clock monotonicity with quantifiable nodes' synchronization error and power overhead.

Recently, Leva and Terraneo [12], [13] have introduced feedback low power synchronization (FLOPSYNC), a solution for the synchronization problem, based on a control-centric design approach [14]. FLOPSYNC is based on a convenient model of the synchronization-related phenomena. This model allows us to state the problem as a feedback control problem, where the controlled system is linear, time invariant, and free of modeling errors and uncertainty is relegated to the generation of a conveniently defined exogenous disturbance. The resulting control system is a proportional and integral (PI) controller and can compensate only for a constant or slowly varying clock skew. We overcame this limitation by introducing FLOPSYNC-2 [15], a message-passing synchronization scheme that relies on a tailored controller structure for the synchronization. In this paper, we describe and detail the control system behind FLOPSYNC-2, and we introduce a simulation model to compare the proposed control system with state-of-the-art synchronization algorithms. Also, we provide the details about how FLOPSYNC-2 should be parameterized based on the deployment conditions of the WSN. To assess FLOPSYNC-2, we provide experiments on real nodes, showing the advantages of the proposal against the relevant competitors.

The resulting scheme guarantees the following properties.

1) Nodes' clocks are inherently continuous and monotonic.
2) The effects of crystal imperfections and aging are eliminated.
3) Thermally induced synchronization errors are canceled, without the need for temperature measurements.

Manuscript received November 3, 2014; revised June 5, 2015; accepted September 5, 2015. Manuscript received in final form September 21, 2015. Date of publication October 29, 2015; date of current version June 9, 2016. This work was supported in part by the Swedish Research Council through the Project entitled Cloud Control and Power and Temperature Control for Large-Scale Computing Infrastructures and in part by the Lund Center for Control of Complex Engineering Systems Linnaeus and Excellence center at Linköping Lund in Information Technology Excellence Centers. Recommended by Associate Editor L. Xie.

A. Leva and F. Terraneo are with the Dipartimento di Elettronica, Informazione e Bioingegneria, Politecnico di Milano, Milan 20133, Italy (e-mail: leva@elet.polimi.it; federico.terraneo@polimi.it).

L. Rinaldi is a former graduate student at the Dipartimento di Elettronica, Informazione e Bioingegneria, Milan 20133, Italy (e-mail: luigi.rinaldi@mail.polimi.it).

A. V. Papadopoulos and M. Maggio are with the Department of Automatic Control, Lund University, Lund 221 00, Sweden (e-mail: alessandro.papadopoulos@control.lth.se; martina.maggio@control.lth.se).

It is worth noting that these properties are not achieved sacrificing simplicity. In fact, the resulting control scheme also proves to be very simple to tune. We show that FLOPSYNC-2 can be tuned offline with minimal information on the nodes' *nominal* characteristics, even for the worst case environmental conditions. This means that no tuning in the field is required—a huge advantage when the conditions for the WSN deployment are extreme, like a volcano. The FLOPSYNC-2 tuning parameters provide a straightforward and interpretable means to adapt to the wide variety of deploy conditions found in WSN applications, with requirements varying from sporadic transmissions for low-bandwidth environmental data collection, through more frequent communication with strict requirements for periodic event-triggered real-time control, up to high-precision almost continuous communication as required for safety critical devices.

We tested FLOPSYNC-2 over state-of-the-art alternatives, both with simulations and experimental results obtained with real hardware. This paper presents experimental evidence that FLOPSYNC-2 is better than existing alternatives, particularly when nodes are exposed to thermal stress. FLOPSYNC-2 is capable of achieving lower synchronization errors and consuming less power with respect to the state-of-the-art solutions.

This paper discusses in detail the control strategy used for FLOPSYNC-2, while implementation details can be found in [15]. It is worth mentioning that the isolation of the core synchronization functionality and its realization as a single feedback controller per node allows one to exploit modern hardware and firmware at their best. For example, a convenient use of a recently proposed flooding scheme [16], coupled with hardware-based packet retransmission timing, makes the synchronization error variance of FLOPSYNC-2 highly independent of the number of network hops between the node holding the reference clock and the current one—another relevant progress compared with the state of the art.

II. Related Work

The node synchronization problem can be posed in many different ways. A common formulation is the symmetric problem, where the clocks of a certain number of sensors should be synchronized at the end of the computation, with no prior information. The symmetric problem has received a lot of attention in the recent years and is typically addressed with consensus-based algorithms [17]–[19].

The asymmetric synchronization problem, on the contrary, addresses clock synchronization in a *master–slave* formulation, where slave nodes, at the end of the computation, should be synchronized with a reference clock, the one held by the master node. This paper contributes to the state of the art with a control-theoretical solution to the *asymmetric* synchronization problem, under the assumption that the master node has already been selected and is known to all the slave nodes.

Despite master–slave architectures have a single point of failure, i.e., the master node, it is quite common in WSNs to have a node that acts as a gateway to collect the information sensed by the other nodes, which is a single point of failure as well. Solutions to the symmetric problem would make more robust the approach, but, to the best of the authors' knowledge, none of them can guarantee clock monotonicity as done adopting FLOPSYNC-2.

The network time protocol (NTP) [20] discusses how to provide synchronized clocks in wired networks and defines the terminology for synchronization-related issues. The difference between the values of two clocks is denoted by *offset*, while the first time derivative of the offset is called *skew*. The variation of the skew in time (the second time derivative of the offset) is named *drift*. NTP provides algorithms to synchronize the value of multiple nodes, but assumes zero drift. It also does not account for short-term frequency variations, commonly called *jitter*. In general, the directives specified in NTP cannot be directly applied to the wireless case [21].

The terms defined by NTP quite directly correspond to the components of the synchronization algorithms proposed in the mainstream literature, which are compared and analyzed in [3] and [9].

The majority of the synchronization schemes [21]–[28] first aim at eliminating the offset, with various algorithms, that behaves similarly to a *clock synchronization* algorithm that is in principle quite simple. The reference node usually transmits a timestamp to all the other nodes (usually periodically) with as few flight time variabilities as possible. Modern flooding schemes like Glossy [16] achieve the coverage of all the nodes of the network in a way that is highly independent of the network topology. The slaves overwrite their clocks with received timestamp, possibly corrected to account for transmission delay. This operation guarantees the instantaneous cancellation of the offset.

Some schemes also include a *skew compensation* part. This second component is installed on top of the clock synchronization and usually consists in estimating a factor by which the total time is multiplied to compensate for the rate difference with respect to the reference one and improve the clock precision in between receptions of the clock synchronization timestamps. Most of the schemes use a window of past skew measurements, obtained by subtracting the timestamps contained in the synchronization packets from the local clock upon synchronization events. These measurements are then divided by the synchronization period. Based on that, to estimate the skew, each algorithm uses a different strategy. flooding time synchronization protocol (FTSP) [25] uses linear regression, Tiny-Sync [26] attempts to constrain the possible skew values by a set of inequalities, and the scheme in [27] uses a maximum-likelihood estimator. A notable work of the same category is feedback based synchronization (FBS) [28], which obtains skew compensation by means of a PI controller. Such compensation is additive instead of multiplicative, but still superimposed to clock synchronization. To the best of the authors' knowledge, the sole exception available to the synchronization plus compensation paradigm is self correcting time synchronization [29], which tries to merge the two in a single control scheme. However, the approach is derived from the phase-locked loop (PLL) technology, which makes the system nonlinear. Finally, all the mentioned schemes require broadcast of the reference clock (i.e., transmission of timestamps).

These synchronization schemes have two major flaws in our opinion. The first one is that instantaneous corrections risk the local clock to be nonmonotonic. The second is that regression-based compensation and its siblings are structurally weak in the face of drift if the timescale of the consequent skew variation is shorter than the duration of the past samples' window. Note that this is quite frequent, especially when there are thermal disturbances. If high precision is required, regression-based skew compensation is thus risky and clock synchronization alone could lead to better results. However, the precision of the system depends on the period of the clock synchronization part, which could be reduced, but would result in higher power consumption. If the radio is kept ON almost continuously for other purposes—a notable example being [30]—a lower period can be tolerated, but this is not always the case. In normal cases—for example, with process control or data logging applications—fast sampling for synchronization is not an option and skew compensation becomes mandatory.

Compared with the literature solutions, the main novelty of the approach presented here is to synchronize based on the *arrival time* of the synchronization packet and not of its content—hence, no timestamp has to be transmitted if not at boot time.

As a summary, different applications require synchronization schemes that are radically different from the *structural* standpoint, contain different components, and have different requirements for integration with the radio and network stack. The solution proposed in this paper allows us to treat all these requirements with the same paradigm and a structurally unified solution. The solution is robust and accurate and can be easily tuned for specific applications' requirements when needed by changing two clearly interpretable parameters.

III. Synchronization as a Control Problem

Consider a WSN with a master node holding the reference clock. The master broadcasts a synchronization packet at a fixed period T, by means of a flooding scheme.

As anticipated, a key idea of the presented research is to move all the synchronization-related information from the content of the synchronization packet to its arrival time. More precisely, each slave computes an *expected* arrival time for the next synchronization packet. If this matches the *actual* arrival time, this means that the slave is synchronized with the master, except for an initial offset to be canceled when the node joins the WSN.

To eliminate the initial offset, the new slave node joining the WSN contacts the master, asking for the synchronization period T and for the reference time corresponding to the transmission of the *next* synchronization packet. The slave then keeps the radio ON until it receives that packet. When the packet arrives, the slave initializes its local time to the time value received at join time. This is the only moment in time where the offset is canceled by overwriting the time of the node and corresponds to the initialization of the time for the specific node.

For convenience, we set the origin of both the reference time t and the local one t_{loc} at the transmission of the synchronization packet corresponding to the initialization of the slave clock. Also, let k be an integer index to count the synchronization packet transmissions, occurring at $t(k)$ and $t_{\mathrm{loc}}(k)$ in the master and slave clocks, respectively; recall that the flooding scheme compensates for the transmission delay and practically eliminates its variability. Therefore, we can safely consider packet transmission and (compensated) reception time to coincide; for more details on the matter, see [16]. Finally, we associate the initialization packet above with $k = 0$ so that $t(0) = t_{\mathrm{loc}}(0) = 0$.

Let now f_o be the nominal frequency of the slave node oscillator, and $\delta_f(t)$ its variation over the reference (continuous) time t, no matter what the cause of said variation is. In the absence of any synchronization action, the offset evolves in the continuous time according to

$$o(t) := t - t_{\mathrm{loc}}(t) = t - \int_0^t \frac{f_o + \delta_f(\tau)}{f_o} d\tau = -\int_0^t \frac{\delta_f(\tau)}{f_o} d\tau \tag{1}$$

whence, with the introduced discrete time index

$$o(k) = -\int_0^{kT} \frac{\delta_f(\tau)}{f_o} d\tau. \tag{2}$$

At the reception of each kth synchronization packet, the slave measures the synchronization error as the difference $e(k)$ between its expected and actual arrival time, respectively, denoted by $t^e_{\mathrm{loc}}(k)$ and $t^a_{\mathrm{loc}}(k)$ and both counted in the local time as

$$e(k) = t^e_{\mathrm{loc}}(k) - t^a_{\mathrm{loc}}(k). \tag{3}$$

Based on the so-obtained error, the slave computes an additive correction $u(k)$ attempting to match a span of $T + u(k)$ in the local time to a span of T in the reference one and sets the expected (local) time for the next packet $t^e_{\mathrm{loc}}(k+1)$ to

$$t^e_{\mathrm{loc}}(k+1) = t^e_{\mathrm{loc}}(k) + T + u(k). \tag{4}$$

As a result, the expected and actual arrival times for the kth packet take the form

$$t^e_{\mathrm{loc}}(k) = kT + \sum_{h=0}^{k-1} u(h), \quad t^a_{\mathrm{loc}}(k) = kT + \int_0^{kT} \frac{\delta_f(\tau)}{f_o} d\tau. \tag{5}$$

Substituting (5) into (3), we have

$$e(k) = \sum_{h=0}^{k-1} u(h) - \int_0^{kT} \frac{\delta_f(\tau)}{f_o} d\tau \tag{6}$$

which shows that synchronization is exact when the sum of the corrections u equals the integral of the normalized frequency variation δ_f / f_o. This confirms that (3) is a correct way to measure the error. In fact, in the absence of control, one sees that $e(k) = o(k)$.

Furthermore

$$\begin{aligned} e(k+1) &= t^e_{\text{loc}}(k+1) - t^a_{\text{loc}}(k+1) \\ &= \sum_{h=0}^{k} u(h) - \int_0^{(k+1)T} \frac{\delta_f(\tau)}{f_o} d\tau \\ &= \underbrace{\sum_{h=0}^{k-1} u(h) - \int_0^{kT} \frac{\delta_f(\tau)}{f_o} d\tau}_{e(k) \text{ from (6)}} + u(k) \\ &\quad - \int_{kT}^{(k+1)T} \frac{\delta_f(\tau)}{f_o} d\tau \\ &= e(k) + u(k) - \int_{kT}^{(k+1)T} \frac{\delta_f(\tau)}{f_o} d\tau \end{aligned} \tag{7}$$

so that defining

$$d(k) = -\int_{kT}^{(k+1)T} \frac{\delta_f(\tau)}{f_o} d\tau \tag{8}$$

the dynamics of the synchronization error—i.e., the system to be controlled—are described by

$$e(k+1) = e(k) + u(k) + d(k) \tag{9}$$

where $u(k)$ assumes the role of the control signal and $d(k)$ that of a disturbance to be rejected. Therefore, in transfer function form, the system under control is

$$E(z) = P(z)\big(U(z) + D(z)\big), \quad P(z) = \frac{1}{z-1}. \tag{10}$$

As such, one could bring in an arbitrarily complex model to account for crystal imperfections and aging, thermally induced frequency variations, oscillator nonlinearities, short-term jitter, and so on, but from the viewpoint of the controller to be synthesized, all of the above collectively take the form of the single exogenous disturbance $d(k)$, to which they contribute in different frequency bands. This relegates any uncertainty to the generation of $d(k)$, which happens outside the control loop, and leads to address a problem where the controlled system (10) is linear, time invariant, device independent, and uncertainty free.

The definition of the problem allows us to synthesize a computationally lightweight control algorithm—an undoubted merit when the hardware is far from powerful, and microseconds are precious. Finally, this problem definition permits each node to be independent and the synchronization problem to be solved with a completely decentralized solution.

IV. Control Synthesis

Section III framed the synchronization problem as a control problem. More specifically, the controller denoted by $R(z)$ in Fig. 1 should be synthesized, when $P(z)$ is the plant given by (10). The controller should effectively reject the unknown but expectable disturbances $d(k)$ and obtain a synchronization error $e(k)$ as low as possible, by acting on the additive correction $u(k)$ introduced in (4).

$P(z)$ is simple and uncertainty free. Due to that, a natural choice for the synthesis of $R(z)$ is to prescribe the closed-loop

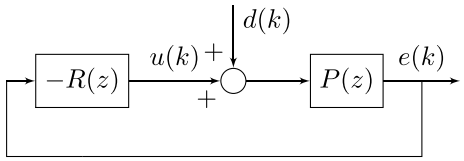


Fig. 1. Feedback control scheme for the generic slave node.

disturbance-to-error transfer function $E(z)/D(z)$ to be equal to a desired function $F^o(z)$, obtaining

$$R(z) = \frac{P(z) - F^o(z)}{P(z)F^o(z)}. \tag{11}$$

The stability of the closed-loop system and its robustness properties are self-evident. The control problem is reduced to the choice of $F^o(z)$ based on a disturbance analysis.

A. Disturbance Analysis

The disturbance $d(k)$ has four main components: 1) crystal Imperfections; 2) aging; 3) short-term jitter; and 4) thermal stress. Crystal imperfections contribute to a constant skew term, and aging phenomena are extremely slow, as they act on a timescale of days while no reasonable synchronization period T can be longer than minutes. The effect of both these disturbance components thus vanishes if $F^o(z)$ has (at least) a unity zero.

Short-term jitter, on the contrary, is invariantly too fast a phenomenon to be counteracted by feedback unless T is made unacceptably small. For our purposes, the jitter component of $d(k)$ is best viewed as a zero-mean random signal. Since on reasonably long horizons, the jitter can be described as a stationary stochastic process, i.e., as the output of an asymptotically stable dynamic system fed with a white noise, to limit its effect on $e(k)$, it is thus required to keep the $\mathcal{H}_2$ norm of $F^o(z)$ as small as possible.

To quantify a reasonable bound for that norm, given the difficulty of obtaining jitter measurements directly, we proceeded as follows. We assumed that the ultimate jitter source, namely, that of the oscillator period with respect to its nominal value, can be described by a white plus a flicker noise process [31]. We wrote a C program that simulates the generation of such a jitter and the measurement of the oscillator Allan deviation as per [32]. We determined the white and flicker noise variances so that the simulated Allan deviation matched that of a typical 32-kHz off-the-shelf crystal [33], the most widely used type in WSN nodes. We finally took the so-generated jitter, accumulated it to obtain its contribution to $d(k)$, and identified a value of $\|F^o(z)\|_2$ so that the standard deviation of its effect on $e(k)$ be within 1 μs, which is more than enough in virtually any application. As a result, we can state that a good default constraint is to have $\|F^o(z)\|_2$ below 1.5, and the experimental activity presented later on confirms that such an estimate is correct in practice. In any case, however, short-term jitter provides an upper bound for synchronization quality, which can ultimately trespassed only by acting on the hardware.

Thermal stress, finally, is of particular concern for the choice of $F^o(z)$, since the timescale of its effect is very often comparable with T. Here, we base our considerations on what the literature agrees to be the harshest thermal disturbance that a

slave may face, which is the radiative heat rate step caused by an abrupt variation, like a shade–sunlight transition [5].

For the purpose of this section, we can describe a node with a single thermal capacity, which exchanges heat convectively with an external environment at the exogenously determined temperature θ_e and is subject to a radiative thermal flux Φ_r. Hence, the dynamics of the node temperature, thus of the crystal one θ_x, since a single capacity is considered, are ruled by

$$C\frac{d\theta_x(t)}{dt} = S_r\Phi_r(t) - \gamma\, S_c(\theta_x(t) - \theta_e(t)) \tag{12}$$

where C is the node heat capacity, S_r is the radiated surface, S_c is that involved in convective heat exchange, and γ is the convective heat transfer coefficient. If a Φ_r step from 0 to $\overline{\Phi}_r$ is applied at $t = 0$, with $\theta_e(t) = \overline{\theta}_e$ and the node initially at the equilibrium with the environment, θ_x evolves as

$$\theta_x(t) = \overline{\theta}_e + \frac{S_r}{\gamma\, S_c}\overline{\Phi}_r\left(1 - e^{-\frac{\gamma S_c}{C}t}\right). \tag{13}$$

For the crystal temperature–frequency relationship, a standard form is the one proposed in [34], namely

$$f(t) = f_o\left(1 + \frac{\beta}{10^6}(\theta_x(t) - \theta_o)^2\right) \tag{14}$$

where θ_o is the temperature corresponding to the nominal frequency and $\beta < 0$ is a crystal-specific parameter expressed in ppm/°C^2. Works like [35] introduce in (14) a small linear term and suggest to account for thermal hysteresis. However, both these phenomena are hardly ever mentioned in crystal data sheets, and no parameters on them are normally provided. As such, we stick to (14) for applicability reasons.

Putting it all together, a first-cut representation of the thermal contribution to $d(k)$, adequate for this analysis, is

$$d_\theta(k) = -\frac{\beta}{10^6}\int_{kT}^{(k+1)T}\left(\overline{\theta}_e + \frac{S_r\overline{\Phi}_r}{\gamma\, S_c}\left(1 - e^{-\frac{\gamma S_c}{C}t}\right) - \theta_o\right)^2 dt. \tag{15}$$

When realistic figures are put into (15), it turns out that $d_\theta(k)$ can steadily increase for several periods from the heat rate step, with a ramplike shape (see the temperature plot in Fig. 4). Thus, since the error should approach zero *before* the oscillator temperature settles, or in many cases of practical interest, both the peak error and its recovery time would be too large, in force of the final value theorem, $F^o(z)$ has to contain *two* unity zeros so as to achieve zero steady-state error with a ramplike disturbance.

B. Choice of $F^o(z)$

According to the analysis above, we need an $F^o(z)$ with one unity zero (two if thermal stress is relevant in the sense above) and an $\mathcal{H}_2$ norm below 1.5 (see Section IV-A). We thus select two transfer functions, to be used cooperatively as explained below, namely

$$F_1^o(z) = \frac{z-1}{z^2}, \quad F_2^o(z) = \frac{(z-1)^2}{(z-\alpha)^3}, \quad 0 \le \alpha < 1 \tag{16}$$

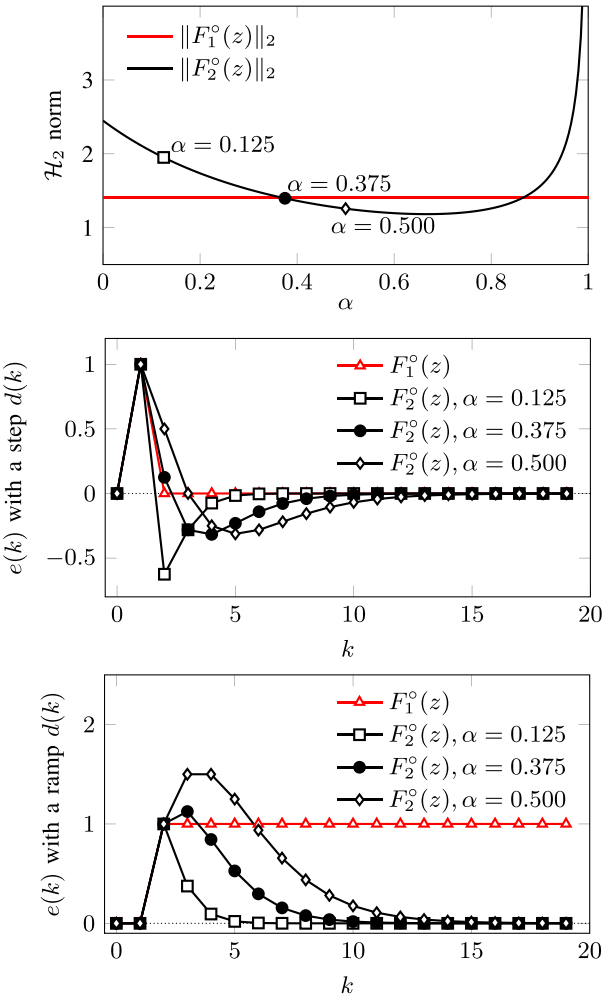


Fig. 2. $\mathcal{H}_2$ norms and time responses of $F_1^o(z)$ and $F_2^o(z)$.

which according to (11), respectively, correspond to

$$R_1(z) = \frac{2z-1}{z-1}$$
$$R_2(z) = \frac{3(1-\alpha)z^2 - 3(1-\alpha^2)z + 1 - \alpha^3}{(z-1)^2}. \tag{17}$$

The impulse response of $F_2^o(z)$, omitting lengthy computations, is

$$y_{s2}(k) = \begin{cases} 0, & k = 0 \\ \alpha^{k-3}\left(1 + \dfrac{(\alpha-1)(\alpha+3)}{2}k + \dfrac{(\alpha-1)^2}{2}k^2\right), & k > 0 \end{cases} \tag{18}$$

corresponding to the $\mathcal{H}_2$ norm

$$\|F_2^o(z)\|_2 = \sqrt{\sum_{k=0}^{\infty} y_{s2}(k)^2} = \sqrt{\frac{6}{(1-\alpha)(1+\alpha)^5}} \tag{19}$$

that monotonically decreases from $\sqrt{6}$ down to $27/25\sqrt{6/5}$—i.e., from 2.45 to 1.18 approximately—for α going from 0 to 2/3. Analogously, one obtains $\|F_1^o(z)\|_2 = \sqrt{2}$.

Fig. 2 shows the $\mathcal{H}_2$ norms of $F_1^o(z)$ and $F_2^o(z)$, and their error responses for a unit step and a unit-slope ramp $d(k)$. When considering $F_2^o(z)$, time responses were obtained with

three representative values of α. The topmost plot shows that $\|F_1^o(z)\|_2$ has an acceptable value, as is true for $\|F_2^o(z)\|_2$ in a wide span of α. Thus, both controllers satisfactorily perform as far as jitter is concerned. The other two plots conversely show that $R_1(z)$ better counteracts a skew step (center plot), but clearly cannot achieve zero steady-state error in the ramp case (bottom plot). With $R_2(z)$, a ramplike skew is effectively counteracted, although large values of α can deteriorate the error convergence time.

As a result, we employ $R_1(z)$ only for the first two periods after the slave initialization, to rapidly compensate for a skew that at that point is totally unknown, and then switch completely to $R_2(z)$, which can steer the error toward zero *before* the temperature settles following a heat rate step. Moreover, as a default, we set $\alpha = 3/8 = 0.375$—a value that eases computations to the advantage of speed—as this gives more or less the same $\mathcal{H}_2$ norm as $R_1(z)$, as marked in the top plot of Fig. 2, and only slightly affects the ramp response peak and settling.

C. Monotonic Virtual Clock

To estimate the reference time between the kth and the $(k+1)$th synchronization events, a slave needs an estimate $\widehat{s}(k)$ of its local clock skew. If this is available, denoting by c_{loc} the reading of the local clock, the reference time estimate can be obtained as

$$\widehat{t}(t_{\text{loc}}) = t_{\text{loc}}(k) + \frac{c_{\text{loc}} - t_{\text{loc}}(k)}{1+\widehat{s}(k)}, \quad t_{\text{loc}}(k) \le t_{\text{loc}} < t_{\text{loc}}(k+1) \tag{20}$$

which, *thanks to the absence of clock overwriting at synchronization instants*, naturally yields a monotonic local time. Of course, the better the $\widehat{s}(k)$, the more accurate the $\widehat{t}(t_{\text{loc}})$.

Although FLOPSYNC-2 does not operate by explicitly estimating the skew, it does provide $\widehat{s}(k)$ somehow implicitly, as $d(k)/T$ is readily interpreted as the average skew over a period, and therefore $\widehat{s}(k) = -u(k)/T$. The transfer function from the real (average) skew to its estimate takes with FLOPSYNC-2 the form

$$\Sigma(z) := \frac{\widehat{S}(z)}{S(z)} = \frac{-U(z)/T}{D(z)/T} = \frac{R(z)P(z)}{1+R(z)P(z)} \tag{21}$$

that with $R_2(z)$—we do not analyze $R_1(z)$ in this respect as it is just used for two periods—becomes

$$\Sigma_2(z) = \frac{(1-\alpha)(3z^2 - 3\alpha z - 3z + \alpha^2 + \alpha + 1)}{(z-\alpha)^3} \tag{22}$$

where α, in the range $[0, 1)$, is the only design parameter.

D. Accuracy Considerations

To end this Section IV, some words are in order—prior to describing simulations and experiments that back up our statement—on the technically achievable accuracy. We limit the scope to $R_2(z)$ for the reason just mentioned and distinguish two basic situations: 1) transient behavior in the face of low-/mid-frequency disturbances, where errors significantly larger than the measurement resolution have to be temporarily accepted, and 2) steady-state behavior, where on the contrary, the absence of disturbances causes the error to approach zero if not for measurement issues.

Considering transient behavior, and in particular thermal stress, disturbances of realistic amplitude and rate result in peak errors well above the resolution of typical off-the-shelf WSN node timers (which is around 30 μs for a 32-kHz crystal). As such, the said resolution is of scarce relevance, and the (T, α) couple can be chosen to optimize the performance index of choice in a hardware-abstracted manner. We built on this idea to create the parameterization tool presented in Section VI, to which the details are deferred.

Coming to the steady-state behavior, the dominant (and very small) effect is conversely that of the short-term oscillator jitter, and measurement quantization effects may have a relevant role. For example, we observed that with a timer based on the typical 32-kHz crystal, the said effects can lead the error to enter a (quasi-)deterministic behavior—not necessarily a limit cycle. If the required accuracy is of the order of tens of microseconds, the resulting error fluctuations can be accepted. If, on the contrary, a finer control of the steady-state error is required, the measurement resolution has to be fine enough for the error to maintain an essentially stochastic character—i.e., to evidence just the jitter effect, and not some accumulation of it. We observed this to happen when the virtual high-resolution timer (VHT) technique [5] is applied, and although the matter needs further investigations, we can state that with such an approach, the claimed (steady-state) precision can actually be achieved in practice.

V. Simulation Assessment

To assess the behavior of FLOPSYNC-2, also in comparison with the state-of-the art alternatives FTSP [25] and FBS [28], we created a simulation model reflecting the system structuring of Section III, and the disturbance generation analysis of Section IV-A, but more detailed than the one used for the first-cut analysis that in the same section led to the controller structure chosen in Section IV-B. The model is based on the following assumptions.

1) In all the involved materials—the node casing and its printed circuit board (PCB)—heat conduction can be neglected in all directions but that orthogonal to the material layer.
2) The thermal resistance of the casing in the direction orthogonal to its surface is dominated by the contact resistances with the internal and external air.
3) The temperature of the air contained in the casing can be considered spatially uniform.

Also, along the same reasoning of Section IV-A, we assumed that only a part of the casing is subject to radiation, while all its surface is involved in convection.

Fig. 3 shows the casing and highlights some of the involved quantities. The surface of the casing that is exposed to radiation is denoted by S_{rc}, while the part not exposed to radiation is indicated by S_{nrc}. The surface that represents the quartz crystal area is indicated by S_x, while the radio transmitter surface is denoted by S_r. Finally, the CPU surface is indicated by S_p.

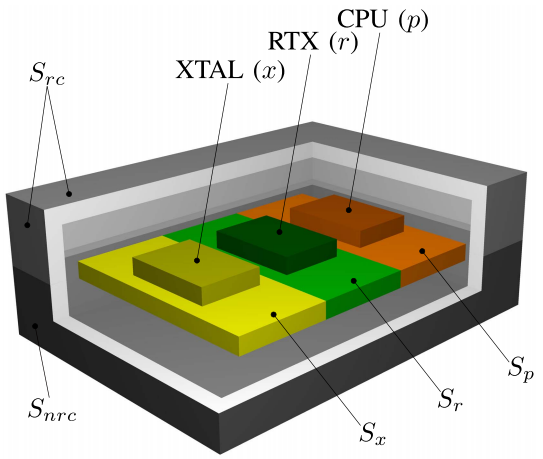


Fig. 3. Node casing and involved quantities.

The relevant heat storage, generation, and exchange phenomena are those listed in the following.

1) A radiated casing thermal capacity C_{rc} receives a thermal power $\Phi_r S_{rc}$, where Φ_r is the radiative flux and S_{rc} is the surface exposed to it. This corresponds, for example, to saying that only the top part of the casing is exposed to the Sun.
2) The same capacity exchanges heat with the contained air and with the external environment via the thermal conductances $\gamma_{ci} S_{rc}$ and $\gamma_{ce} S_{rc}$, respectively, where γ_{ci} and γ_{ce} are the internal and external heat exchange coefficients for the casing.
3) A thermal capacity C_{nrc}, accounting for the part of the casing not exposed to radiation, exchanges heat with the contained air and with the external environment via the thermal conductances $\gamma_{ci} S_{nrc}$ and $\gamma_{ce} S_{nrc}$, where S_{nrc} is the involved surface. This implies, for example, that there is a part of the casing that is not directly exposed to the Sun because of the current conditions.
4) The node PCB is divided into three thermal capacities C_p, C_r, and C_x, corresponding, respectively, to the processor, radio transceiver, and crystal oscillator sections. All of them exchange heat convectively with the air contained in the casing, associated with the thermal capacity C_a, respectively, via the three thermal conductances computed for simplicity as the single heat exchange coefficient γ_{ci} times the three surfaces S_p, S_x, and S_r.
5) Within the p, x, and r sections of the node PCB, three thermal power generation sources are present, each one—in the practical impossibility of describing in detail facts that are heavily software dependent—characterized by an average value of P_p, P_x, and P_r.

Based on the considerations above, the model does not include any strictly physical description of short-term jitter. If necessary, this could easily be provided as a random additive disturbance. Also crystal imprecisions and aging (see the discussion in Section IV-A) are not necessary in a model aimed at control assessment, since they are too slow to be relevant.

Summing up, the model contains the continuous-time thermal part

$$\begin{cases} C_{rc}\dfrac{d\theta_{rc}(t)}{dt} = S_{rc}\Phi_r(t) - \gamma_{ci}S_{rc}(\theta_{rc}(t)-\theta_a(t)) \\ \qquad\qquad - \gamma_{ce}S_{rc}(\theta_{rc}(t)-\theta_e(t)) \\ C_{nrc}\dfrac{d\theta_{nrc}(t)}{dt} = -\gamma_{ci}S_{nrc}(\theta_{nrc}(t)-\theta_a(t)) \\ \qquad\qquad - \gamma_{ce}S_{nrc}(\theta_{nrc}(t)-\theta_e(t)) \\ C_a\dfrac{d\theta_a(t)}{dt} = \gamma_{ci}S_{rc}(\theta_{rc}(t)-\theta_a(t)) \\ \qquad + \gamma_{ci}S_{nrc}(\theta_{nrc}(t)-\theta_a(t)) \\ \qquad + \gamma_{ci}S_p(\theta_p(t)-\theta_a(t)) \\ \qquad + \gamma_{ci}S_r(\theta_r(t)-\theta_a(t)) \\ \qquad + \gamma_{ci}S_x(\theta_x(t)-\theta_a(t)) \\ C_p\dfrac{d\theta_p(t)}{dt} = P_p(t) - \gamma_{ci}S_p(\theta_p(t)-\theta_a(t)) \\ C_r\dfrac{d\theta_r(t)}{dt} = P_r(t) - \gamma_{ci}S_r(\theta_r(t)-\theta_a(t)) \\ C_x\dfrac{d\theta_x(t)}{dt} = P_x(t) - \gamma_{ci}S_x(\theta_x(t)-\theta_a(t)) \end{cases} \tag{23}$$

where $\theta_{rc}(t)$ and $\theta_{nrc}(t)$ are, respectively, the temperatures of the radiated and nonradiated casing parts, $\theta_a(t)$ is the temperature of the contained air, $\theta_p(t)$ is the temperature of the processor, $\theta_r(t)$ is the temperature of the radio transceiver, and $\theta_x(t)$ is the temperature of crystal oscillator sections, while—like in the simplified model (12)—the external temperature $\theta_e(t)$ and the radiative flux $\Phi_r(t)$ are the exogenous inputs. Cascaded to (23) is a temperature–frequency algebraic description in the form (14), providing the instantaneous frequency $f(t)$ as a function of the crystal temperature $\theta_x(t)$, and finally, $d(k)$ is produced as per (8), with $\delta_f(t) = f(t) - f_o$. Strictly speaking, the sampling involved in (8) may be subject to small period fluctuations since it is done in the local slave time, but such a higher order effect proved to be negligible in practice.

The simulation model just described was implemented in the Modelica language[1] [36], [37]. Thanks to the ability of Modelica to mix equation- and algorithm-based modeling, the required synchronization algorithms (FLOPSYNC-2, FTSP, and FBS) were realized in the form of code *replica*, and others can be seamlessly added. The simulator was used for many purposes, of which two examples follow.

A. Skew Estimation Assessment

A test demonstrating the quality of the FLOPSYNC-2 skew estimation discussed in Section IV-C, thus of the virtual clock, is presented.

The test refers to a typical shade–sunlight transient, and the results are shown in Fig. 4. The crystal has a nominal frequency of 32 kHz at 25 °C, and $\beta = -0.035$ ppm/°C^2; its temperature transient is depicted in the topmost plot. The real skew is shown by the dotted line in the center plot. The red lines, in the same plot, represent regression-based skew estimates with $T = 1$ min and a window of 4, 8, and 16 past samples. The blue dashed line is conversely the skew estimated

[1] Modelica. https://modelica.org/

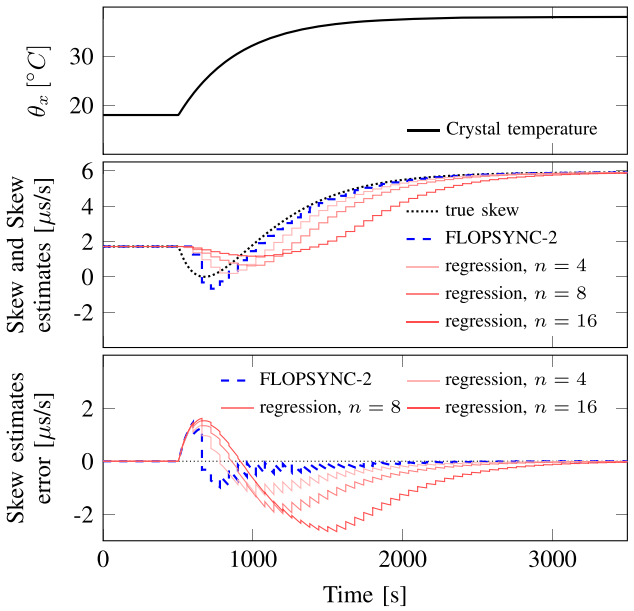


Fig. 4. Skew estimates during a temperature transient.

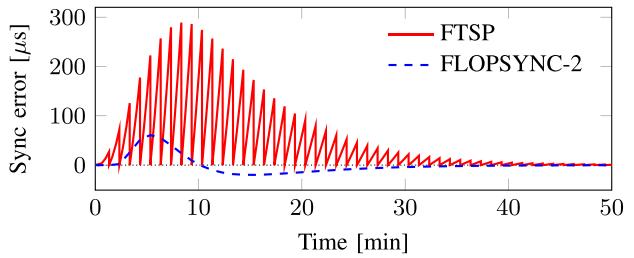


Fig. 5. Example of simulation results—FLOPSYNC-2/FTSP comparison.

by FLOPSYNC-2, as per (22) with $\alpha = 3/8$. The bottom plot finally shows the skew estimate errors.

Not only FLOPSYNC-2 is faster at recovering a good estimate, but there is a more general and important remark. In the real life of a WSN node, as confirmed by this test, there is hardly ever such thing as a true and constant skew to be estimated. Skew comes from a dynamic system subject to inputs (in this case, radiative heat) that can vary with largely unpredictable amplitude and rate. This makes synchronization techniques using regression-based skew estimation structurally weak, whence the workarounds proposed in [38].

Synchronization based on *error feedback* is far more robust, and if a skew estimate is necessary to realize a virtual clock, a solution is to obtain that estimate from the *control* signal, which is naturally led to match the skew dynamics.

B. Scheme Comparison

Fig. 5 shows a test in which FLOPSYNC-2 is compared with FTSP, using the OpenModelica tool. The regression window size in FTSP is set to eight samples, as in the TinyOS implementation.[2]

The test refers to the same shade–sunlight transition of Section V-A, and the reported signals are the synchronization

[2]https://github.com/tinyos/tinyos-main/blob/master/tos/lib/ftsp/TimeSyncP.nc, line 76

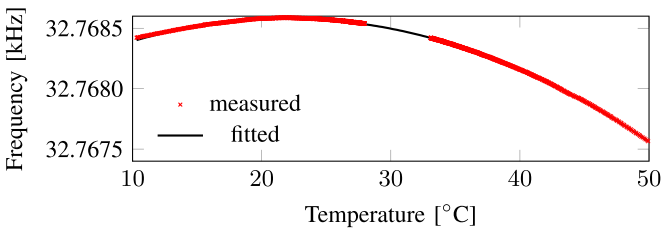


Fig. 6. Temperature–frequency curve for the tuning fork crystal used in one of the sensor nodes.

errors in seconds. Note the effect of the double integrator in the FLOPSYNC-2 controller $R_2(z)$ that steers the error toward zero before the temperature settles (see the topmost plot of Fig. 4). This is not obtained by FTSP, resulting in a larger and longer transient. Moreover, the error jumps exhibited by FTSP indicate a nonmonotonic local time. On the contrary, FLOPSYNC-2 guarantees a continuous and monotonic virtual clock—its error has no jumps at all.

VI. Parameterization

With the designed controller, the only open issue is the choice of the parameter α and the synchronization period T. The synchronization period has a tremendous impact on the node's power consumption and should be optimized for the specific WSN deployment. If the synchronization period is too short, the timing properties of the nodes are better than needed, but this unduly reduces the life of batteries. On the contrary, a too large period can cause nodes' malfunctions, in case the synchronization requirements are strict.

One of the main advantages of the proposed methodology is that the optimization of the synchronization period T and the corresponding choice of the parameter α can be done offline, before the nodes are deployed, based on the guarantees to be obtained and on the operating conditions (therefore avoiding heuristics).

FLOPSYNC-2 naturally provides a straightforward solution for the selection of the couple (T, α), based on nominal information on the crystal oscillator, operating condition data, and error bounds. The tuning procedure is based on an offline design space exploration that uses the simplified model proposed in Section IV-A. The use of the more accurate model presented in Section V is not encouraged, since knowing thermal capacities and conductances may be cumbersome. On the contrary, worst case data on the ambient temperature are not difficult to obtain, based on meteorological information on the deployment site, while characterizing the crystal is quite straightforward in a laboratory setting. We verified this assumption using a frequency counter with a rubidium timebase and a temperature sensor placed on the crystal, both connected to a computer to log temperature and frequency measurements. The temperature was swept by placing the node in a controlled chamber, and the logged data were used to determine the temperature–frequency curve. An example of the obtained results, to show the achievable accuracy, is reported in Fig. 6.

The exploration yields the couples that satisfy the synchronization error constraints under thermal stress, and computes

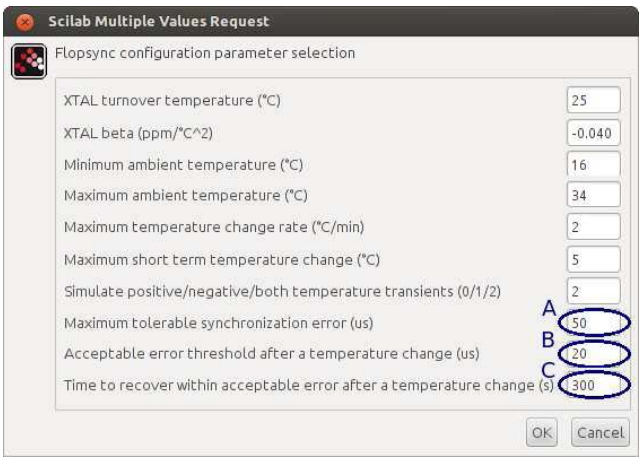


Fig. 7. FLOPSYNC-2 configuration tool user interface. The design requirements A, B, and C are specified as in Fig. 9.

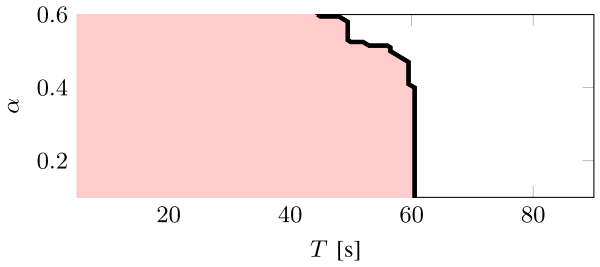


Fig. 8. Feasible (T, α) couples and Pareto frontier.

the Pareto frontier to target a WSN toward battery life, or minimum residual jitter.

We created a configuration tool to easily perform the exploration: by filling out the form shown in Fig. 7, the user receives an output similar to Fig. 8. The filled region contains the feasible (T, α) couples, and the thick black line is the Pareto frontier (the observed quantization is due to the synchronization period granularity).

The input data for the configuration tool of Fig. 7 can be divided into three categories. The first category holds the nominal crystal parameters, which can be obtained from the node data sheet. The second refers to the assumed worst case thermal stress, specified in detail as follows:

1) a minimum and a maximum external temperature $\theta_{e,\min}$ and $\theta_{e,\max}$;
2) a maximum magnitude $r_{\theta,\max}$ for the temperature variation rate;
3) a maximum magnitude $\Delta\theta_{\max}$ for the temperature swing in a single thermal event, as it is very unlikely that the said swing spans the entire $(\theta_{e,\min}, \theta_{e,\max})$ range.

The last category contains the *synchronization requirements*. Given the shape of a typical error trace during a temperature transient with FLOPSYNC-2, as shown in Fig. 9, the tool allows us to select the maximum magnitude for the tolerable error peak value after a thermal event (A in Fig. 9) and the maximum time C for the error to fall within a prescribed zero-centered range of amplitude B after such an event.

Once the form is filled, the tool simulates a set of thermal transients, according to the provided data, and in

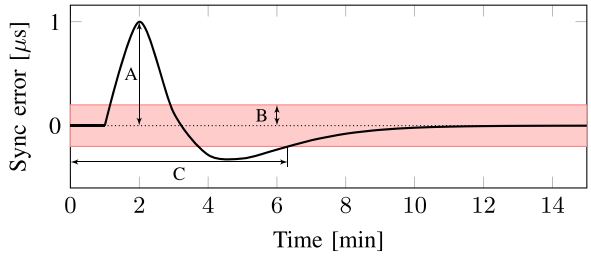


Fig. 9. Typical synchronization error of FLOPSYNC-2 under a thermal transient, showing the meaning of the design constraints for the configuration tool of Fig. 7.

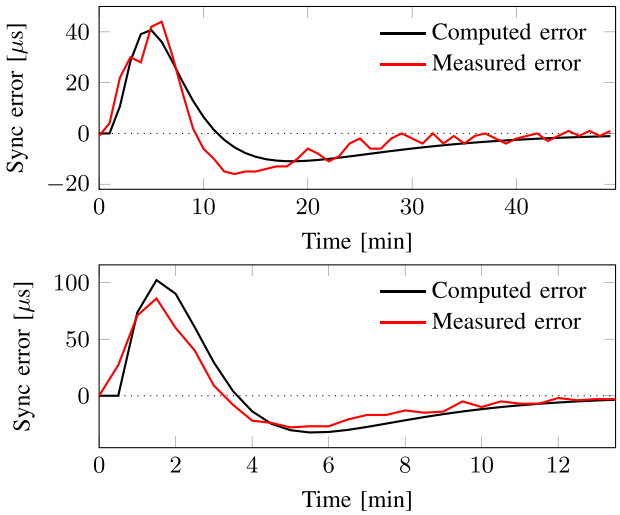


Fig. 10. Comparison between the measured synchronization error (red line) and the simulated one (black line) for two temperature transients.

both directions. This is important, because the parabolic crystal characteristic causes the same temperature variation to produce different errors depending on whether the direction is from lower to higher temperature or vice versa.

The tool was validated by comparing its results with several experiments made with real WSN nodes recording the synchronization error during thermal transients of various amplitudes and rates. To witness the ability of the configuration tool to estimate the synchronization error of a particular temperature transient, we report a sample of the validation experiments carried out to assess the tool. Fig. 10 reports in red line the synchronization error measured during two temperature transients, the top one being a temperature change from 24 °C to 47 °C with a maximum temperature change rate of 2.25 °C/min, while the bottom one refers to an experiment that consists in putting a WSN node inside a fridge, which resulted in a change in temperature from 22 °C to −5 °C with a maximum temperature change rate of 10 °C/min. The black line shows instead the expected synchronization error under the aforementioned conditions produced by the model underlying the configuration tool. The validation has shown that the tool can estimate the maximum synchronization error during a temperature transient with less than 20% error. It is worth noting that this result was obtained despite using only the *nominal* data for the clock crystal, instead of the accurate values obtained in Fig. 6, to evidence that the tool can be used

without the need to thoroughly characterize the manufacturing tolerances of every single WSN node.

Despite being useful for the choice of the involved parameters T and α, the use of the tool is optional. In fact, even when no tuning is conducted, FLOPSYNC-2 *structurally* guarantees stability and zero steady-state error by design. The consequences of not applying the mentioned tuning procedure are only a larger error peak and/or a slower convergence. In standard applications, the default parameters can be used safely, while in critical cases, the sole knowledge needed is application requirements, nodes' nominal parameters, and operational conditions. It is worth stressing that a correct use of the tuning interface also guarantees that the nodes will have good performance and will not waste energy due to excessive conservatism in the parameterization.

VII. Implementation

FLOPSYNC-2 is composed of four main components: 1) a flooding scheme to rapidly disseminate the synchronization packets; 2) the controllers $R_1(z)$ and $R_2(z)$; 3) the virtual clock as per (20); and 4) a resynchronization procedure.

The first two components are periodic tasks with period T. The virtual clock is invoked whenever the reference time is demanded by the node or its running applications, while resynchronization is executed when a new node joins the network, or after a slave node failure.

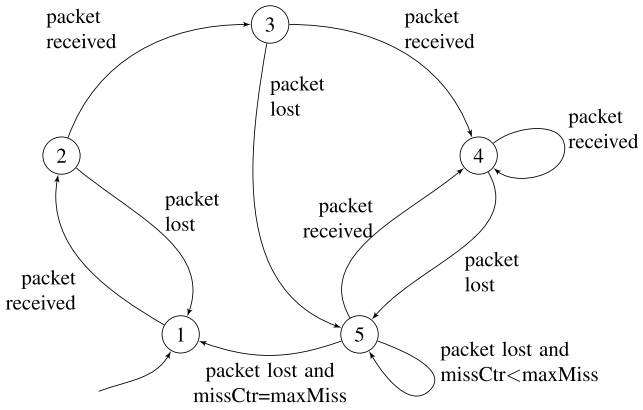
To minimize idle listening, FLOPSYNC-2 adapts the time advance w of the radio activation with respect to the expected synchronization packet arrival. This is very important for low-power operation. For this procedure, the synchronization error standard deviation σ_e is computed in batches of N_σ samples, and w is set to $3\sigma_e$, clamped for safety between a minimum $w_{\min}$ and a maximum $w_{\max}$.

When a synchronization packet is received, the radio goes OFF immediately. On the contrary, in case the packet is not received, the radio is kept active for a maximum of $2w + p$, where p is the time necessary for one packet transmission. After that, the packet is considered lost, w is doubled, and the control variable u of the previous period is also used for the following one. Finally, a counter `missCtr` is incremented every time a synchronization packet is lost, and reset to zero when one is received. If the counter exceeds a threshold value `maxMiss`, the slave triggers the resynchronization procedure. The operation of FLOPSYNC-2 is described by the Moore machine in Fig. 11. State 1 is the initial one and comprises the single timestamp request needed for the initial synchronization (or resynchronization).

FLOPSYNC-2 was implemented on a WSN node platform employing an ARM Cortex-M3 microcontroller running at 24 MHz and a CC2520 transceiver. The software is written in C++ as an application for a microcontroller operating system named Miosix[3] and released as free software.[4]

VIII. Experimental Results

This section discusses experimental evidence that FLOPSYNC-2 improves nodes' synchronization with respect to state-of-the-art solutions. It provides two experiments to support the claims of low synchronization error and correctness even when the nodes are subject to thermal stress.



States	Actions
1	wait for sync packet
2	initialise $R_1(z)$
3	run one step with $R_1(z)$, init $R_2(z)$, set missCtr=0
4	run one step with $R_2(z)$, set missCtr=0
5	increment missCtr

Fig. 11. State machine for FLOPSYNC-2.

The configuration of the testing infrastructure is the following. The control parameters of the synchronization procedure are the synchronization period T, set to 60 s, and the parameter α, equal to $3/8$. This value was selected as a compromise, since the rejection of thermal stress calls for values of α lower than the optimum for the $\mathcal{H}_2$ norm. Moreover, the implementation contains three additional parameters. The number of samples N_σ —used to compute the standard deviation and correspondingly the window of radio activity for the synchronization procedure—is set to 8. The maximum and minimum values for the window duration $w_{\min}$ and $w_{\max}$ are chosen to be 30 μs and 5 ms, respectively.

To measure the error between synchronization instants and to test the virtual clock behavior, the slave nodes send additional packets every 1.5 s to the master. These packets are not needed in a real deployment and are here used only to evaluate the performance of the synchronization scheme.

A. Synchronization Error Statistical Distribution

The first test is aimed at testing the statistical distribution of the synchronization error in a multihop WSN. To this end, we performed an experiment with nodes distributed in an office building, forming a nine-hop network. These nodes were exposed to interference like the presence of Wi-Fi networks, to provide a realistic setting. Also, to simulate a real scenario, the experiment lasted for six days.

Fig. 12 shows the synchronization error average and standard deviation as a function of the hop count. Fig. 12 refers to the entire data set, i.e., includes the error measurements taken in between synchronization events, to give a complete picture of the achieved accuracy.

Observe that the average is extremely low, down to just a few tens of *nanoseconds*. For small hop counts, this is even less than the tick resolution of the employed timer, which is 42 ns. Also, both the average error and its standard deviation

[3]Miosix embedded OS (http://miosix.org)

[4]FLOPSYNC-2. http://miosix.org/flopsync.html

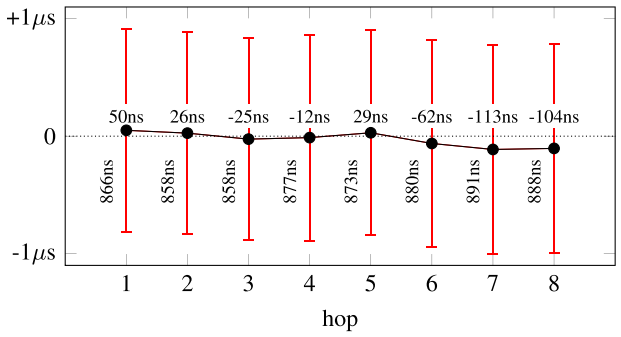


Fig. 12. FLOPSYNC-2 average synchronization error and standard deviation as a function of the hop count.

TABLE I
FLOPSYNC-2 SYNCHRONIZATION ERROR IN A TWO-HOP EXPERIMENT WITH AND WITHOUT VHT. THESE STATISTICS ARE OBTAINED CONSIDERING ONLY THE ERROR SAMPLES IMMEDIATELY BEFORE A SYNCHRONIZATION

	With VHT	
	Average	Standard Deviation
First hop	298 ps	995 ns
Second hop	−817 ps	988 ns

	Without VHT	
	Average	Standard Deviation
First hop	−11.7 μs	24.7 μs
Second hop	16.7 μs	47.9 μs

show a significantly weak dependence on the hop count: this is a merit of the hardware flooding scheme that practically eliminates jitter accumulation from one hop to the next one.

During the entire experiment duration, it never happened that a slave node was desynchronized, although the presence of interference caused some synchronization packet loss. This confirms that the design assumption that resynchronizations are infrequent is not a limitation in real-life deployment. In this case, the one-shot timestamp request used by FLOPSYNC-2 at join is beneficial in terms of limited information exchange and protocol efficiency.

Table I compares the results achieved in a two-hop network with and without VHT, and has two purposes. The first one is to back up the statements of Section IV-D. The second is to show, by considering in the statistics only the error samples immediately before a synchronization event, that the worst case accuracy is also improved. More specifically, comparing the results of the first two hops of Fig. 12 with Table I, it can be observed that when only the error samples immediately before a synchronization are considered, the average error is significantly lower, reaching values less than a nanosecond. This is because of the integral action of the controller that directly acts on these values, while the average error during the period suffers from additional errors due to the linear interpolation used for skew compensation. This remains true also for the case without VHT, which reaches an error that is less than one (low frequency) timer tick, that is, 61 μs. On the contrary, the variance of the error samples immediately before a synchronization tends to be higher as this is the most distant point in time from the last clock correction, and thus the one showing the most variability due to the accumulated skew and drift.

Finally, we logged the time spent by each node in idle listening during the six-day experiment. On average, this time resulted to be 21 μs per period. This value is used to estimate the power consumption overhead imposed on the node for the synchronization procedure in Section IX.

B. Behavior Under Thermal Stress

This second test is intended to be a stress test for the system. In fact, one of the most extreme tests for a clock synchronization scheme is the frequency drift caused by a shade–sunlight transition.

We compare the drift compensation capabilities of FLOPSYNC-2 with two state-of-the-art alternatives, namely, FTSP [25] and FBS [28]. Both adhere to the mainstream scheme with clock synchronization plus multiplicative skew compensation (see Section II). The former estimates the skew via linear regression, as in Fig. 4, while the latter uses a PI controller (thus, a *single* integrator). During the test, three nodes running the synchronization schemes were placed in an enclosure and exposed to the Sun. The enclosure has the double purpose of providing a realistic setting for an outdoor application and keeping the temperature of the nodes as uniform as possible, so as to better compare their response. FTSP was configured with a window of 8 data samples, as in the widely used TinyOS implementation [25], while FBS was configured with $K_i = K_p = 0.7847$, as suggested in [28].

The resulting synchronization error is shown in Fig. 13. The bottom plot of Fig. 13 shows the temperature variation during the experiment. Compared with FTSP, FLOPSYNC-2 performs significantly better: error feedback quickly compensates for clock drift, achieving a maximum error of 45 μs, while with FTSP, the error rises up to 293 μs. Also, FLOPSYNC-2 requires just 7 min—only seven controller interventions—to steer the error into a ± 20 μs range, while FTSP requires 38 min.

Compared with FBS, the more advanced controller structure of FLOPSYNC-2 comes into play: within 7 min, the error reduces to ± 20 μs and, most notably, stays in that range while the temperature is still rising. On the contrary, the PI controller of FBS exhibits a higher maximum error of 94 μs, but, more importantly, does not adequately compensate for the error until the temperature has settled. As a result, FBS takes 28 min to bring the error in the same ± 20 μs range.

Moreover, FTSP and FBS rely on timestamp transmission and instantaneous clock corrections. This can accommodate for a quasi-constant skew when the temperature is almost constant, but temperature variations yield evident clock jumps. The FLOPSYNC-2 clock shows continuity and monotonicity, which are achieved by design.

IX. POWER CONSUMPTION MODEL

This section discusses the power overhead for synchronization in nodes with FLOPSYNC-2, by providing a consumption model. The model is the result of a profiling phase on the nodes, to extract the operating state transitions for the CPU and

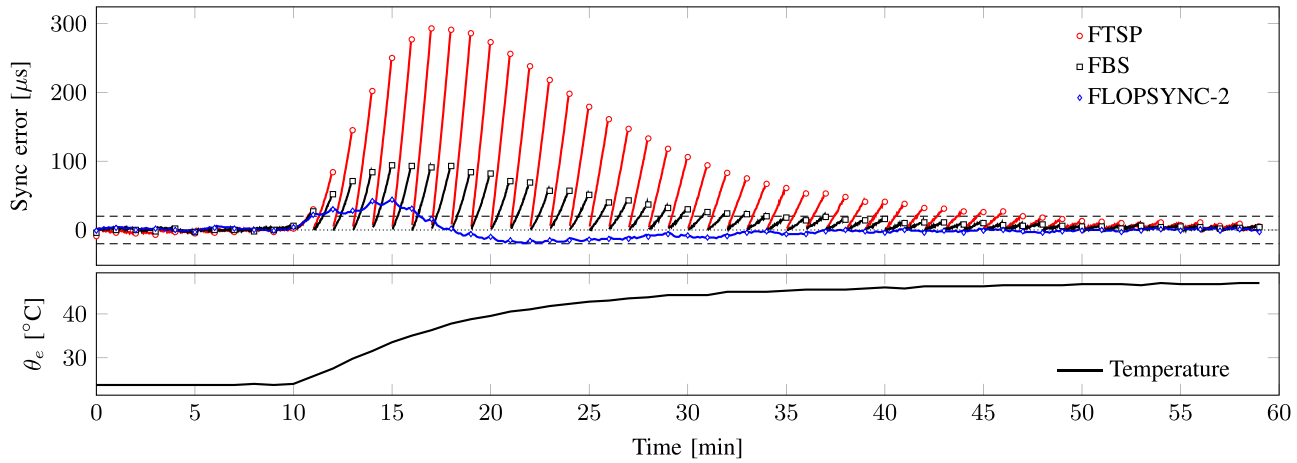


Fig. 13. Synchronization error during the temperature transient caused by a shade–sunlight transition. The dots highlight the error immediately before synchronizing, while the line shows the time error between synchronizations. Note how clock jumps occur in FTSP and FBS causing clock continuity and monotonicity issues. The dashed lines mark the ±20-μs error values. Nodes were exposed to the Sun at minute 10.

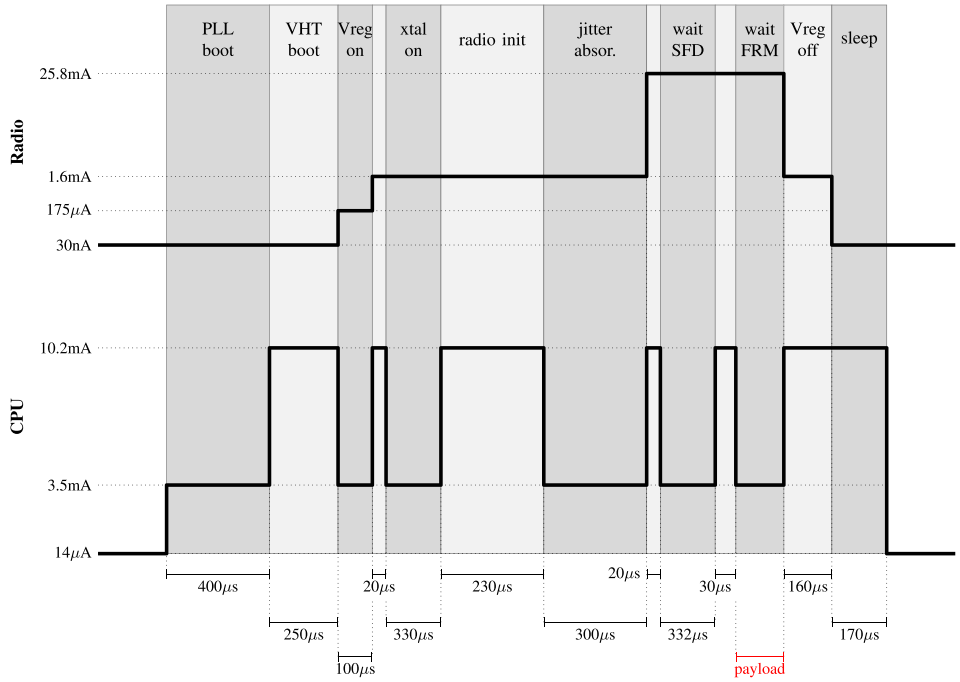


Fig. 14. Current consumption trace for the master node (not in scale). Top: radio. Bottom: CPU.

the transceiver. The timing information is combined with current consumption data from data sheets to obtain reliable consumption estimates.

Fig. 14 shows the current consumption profile for the master node.[5] At the beginning of each synchronization period, the CPU wakes out of deep sleep, starts its PLL, and enters the run state at the full clock frequency. It then performs internal timer (VHT) resynchronization and wakes up the transceiver, waiting for its voltage regulator (Vreg) and oscillator (xtal) to start. The transceiver is configured (radio init), and both radio and CPU remain in the sleep state for some slack time, to absorb the jitter of the PLL and oscillator startup. The CPU then wakes up, prepares the synchronization packet, sends it to the transceiver, and waits for the start frame delimiter interrupt, staying in the sleep (not deep sleep) state. Once the packet is transmitted, which is signaled by the frame (FRM) interrupt, the CPU wakes up, puts the transceiver into deep sleep by turning OFF its voltage regulator, performs some bookkeeping, and goes back to deep sleep. The uncommented 20-μs time spans in Figs. 14 and 15 are to send commands to the radio transceiver.

Fig. 15 conversely shows the consumption profile for a slave node. The CPU and radio wakeup sequence is identical to that of the master. After the slack time, however, the

[5]The overhead is traditionally given as current because this eases the estimation of the battery lifetime. The ultimate reason is that the node electronics draws an almost constant current as a function of the operating state, while along the battery discharge, the voltage—thus the power—varies.

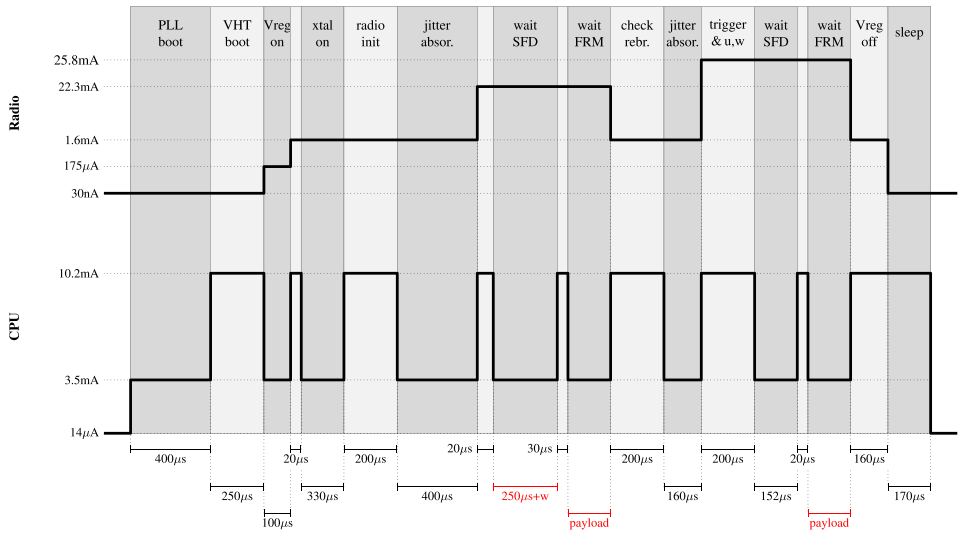


Fig. 15. Current consumption trace for a slave node (not in scale). Top: radio. Bottom: CPU.

CPU instructs the transceiver to start receiving, sets a timeout interrupt, and goes to sleep. When the synchronization packet is received, the CPU is awakened, and after a fixed time, controlled by hardware, the packet is rebroadcast. While the packet is being sent, the CPU runs the controller, and then goes to sleep. When the packet is sent, the transceiver enters deep sleep, followed after some bookkeeping by the CPU.

Thanks to its algorithmic simplicity, the code paths of FLOPSYNC-2 are highly predictable. The average consumption overhead can thus be obtained by integrating the area below the current profile, dividing by the period T, and subtracting the consumption of the node in deep sleep. The resulting equations, parametric in the packet payload p_b, the window w, and the period T, are

$$I_{\text{master}} = \frac{25.6\ \mu\text{C} + 0.94\ \mu\text{C} \cdot p_b}{T} \tag{24}$$

for the master node and

$$I_{\text{slave}} = \frac{37.8\ \mu\text{C} + 1.76\ \mu\text{C} \cdot p_b + 25.8\ \text{mA} \cdot w}{T} \tag{25}$$

for a generic slave node.

Summarizing, with a 2-byte payload (glossy retransmission counter and checksum), a period T of 60 s, and an average w of 21 μs as obtained in the experiment described in Section VIII-A, the current consumption overhead of FLOPSYNC-2 is 458 nA for the master node and 698 nA for a generic slave.

X. Conclusion

This paper discussed a synchronization scheme for the nodes of WSNs. The synchronization problem is seen as a control problem, with a perspective shift with respect to the related literature. This allows the proposed solution to differ from state-of-the-art ones (that are inherently depending on timestamps sent with synchronization packets) and to design and realize a completely decentralized control system that relies only on the reception time of the synchronization packets, reducing the transmission overhead. The resulting discrete-time linear time-invariant control system is guaranteed stable and robust, and the uncertainty is relegated to disturbance signals to be rejected.

The proposed solution was simulated, implemented, and tested on real hardware. The experimental results show that the decentralized control scheme has very good performance and low power overhead on both the master and slave nodes' part. Also, the scheme can be easily tailored to extreme ambient conditions, by just modifying two parameters. The implementation of FLOPSYNC-2 was released as free software, together with a configuration tool to select those two parameters based on nominal data only.

While we have shown the benefits over standard clock synchronization strategies, the proposed scheme does not explicitly take into account network propagation delays. This is not a problem when the nodes' distances do not vary and when the nodes are close. When these conditions are not satisfied, a proper compensation of the propagation delay may become critical. To address this issue, we have developed a delay compensation block that can complement any synchronization scheme based on flooding and enhance the performance of FLOPSYNC-2 [39].

Future work will address the creation of a complete communication stack based on the presented scheme and its use in time-critical applications. The proposed models will also form the basis for a rigorous determination of the ultimate synchronization limit of a given architecture, which would be of great use for design assessment purposes.

References

[1] G. Werner-Allen *et al.*, "Deploying a wireless sensor network on an active volcano," *IEEE Internet Comput.*, vol. 10, no. 2, pp. 18–25, Mar./Apr. 2006.

[2] L. Mottola, T. Voigt, I. G. Silva, and R. Karoumi, "From the desk to the field: Recent trends in deploying wireless sensor networks for monitoring civil structures," in *Proc. IEEE Sensors*, Oct. 2011, pp. 62–65.

[3] I.-K. Rhee, J. Lee, J. Kim, E. Serpedin, and Y.-C. Wu, "Clock synchronization in wireless sensor networks: An overview," *Sensors*, vol. 9, no. 1, pp. 56–85, 2009.

[4] T. Schmid, R. Shea, Z. Charbiwala, J. Friedman, M. B. Srivastava, and Y. H. Cho, "On the interaction of clocks, power, and synchronization in duty-cycled embedded sensor nodes," *ACM Trans. Sensor Netw.*, vol. 7, no. 3, 2010, Art. ID 24.

[5] T. Schmid, P. Dutta, and M. B. Srivastava, "High-resolution, low-power time synchronization an oxymoron no more," in *Proc. 9th ACM/IEEE Int. Conf. Inf. Process. Sensor Netw. (IPSN)*, Apr. 2010, pp. 151–161.

[6] Y. Chen, Q. Wang, M. Chang, and A. Terzis, "Ultra-low power time synchronization using passive radio receivers," in *Proc. 10th Int. Conf. Inf. Process. Sensor Netw. (IPSN)*, Apr. 2011, pp. 235–245.

[7] Y.-C. Wu, Q. Chaudhari, and E. Serpedin, "Clock synchronization of wireless sensor networks," *IEEE Signal Process. Mag.*, vol. 28, no. 1, pp. 124–138, Jan. 2011.

[8] P. Briff, A. Lutenberg, L. R. Vega, and F. Vargas, "On the trade-off of power consumption and time synchronization quality in wireless sensor networks," in *Proc. IEEE Sensors*, Oct. 2012, pp. 1–4.

[9] S. El Khediri, N. Nasri, M. Samet, A. Wei, and A. Kachouri, "Analysis study of time synchronization protocols in wireless sensor networks," *Int. J. Distrib. Parallel Syst.*, vol. 3, no. 3, pp. 155–165, 2012.

[10] F. Heidarian, J. Schmaltz, and F. Vaandrager, "Analysis of a clock synchronization protocol for wireless sensor networks," *Theoretical Comput. Sci.*, vol. 413, no. 1, pp. 87–105, 2012.

[11] P. Sommer and R. Wattenhofer, "Gradient clock synchronization in wireless sensor networks," in *Proc. Int. Conf. Inf. Process. Sensor Netw. (IPSN)*, 2009, pp. 37–48.

[12] A. Leva and F. Terraneo, "Low power synchronisation in wireless sensor networks via simple feedback controllers: The FLOPSYNC scheme," in *Proc. Amer. Control Conf. (ACC)*, Jun. 2013, pp. 5017–5022.

[13] A. Leva and F. Terraneo, "High-precision synchronisation in wireless sensor networks with no tuning in the field," in *Proc. 19th IFAC World Congr.*, 2014, pp. 707–712.

[14] A. Leva, M. Maggio, A. V. Papadopoulos, and F. Terraneo, *Control-Based Operating System Design*. London, U.K.: IET, 2013.

[15] F. Terraneo, L. Rinaldi, M. Maggio, A. V. Papadopoulos, and A. Leva, "FLOPSYNC-2: Efficient monotonic clock synchronisation," in *Proc. 35th IEEE Real-Time Syst. Symp. (RTSS)*, Dec. 2014, pp. 11–20.

[16] F. Ferrari, M. Zimmerling, L. Thiele, and O. Saukh, "Efficient network flooding and time synchronization with Glossy," in *Proc. 10th Int. Conf. Inf. Process. Sensor Netw. (IPSN)*, Apr. 2011, pp. 73–84.

[17] R. Carli and S. Zampieri, "Network clock synchronization based on the second-order linear consensus algorithm," *IEEE Trans. Autom. Control*, vol. 59, no. 2, pp. 409–422, Feb. 2014.

[18] O. Simeone and U. Spagnolini, "Distributed time synchronization in wireless sensor networks with coupled discrete-time oscillators," *EURASIP J. Wireless Commun. Netw.*, vol. 2007, no. 1, pp. 057054-1–057054-13, 2007.

[19] R. Carli, A. Chiuso, L. Schenato, and S. Zampieri, "A PI consensus controller for networked clocks synchronization," in *Proc. 17th IFAC World Congr.*, vol. 17. 2008, pp. 10289–10294.

[20] D. L. Mills, "Internet time synchronization: The network time protocol," *IEEE Trans. Commun.*, vol. 39, no. 10, pp. 1482–1493, Oct. 1991.

[21] J. Elson and D. Estrin, "Time synchronization for wireless sensor networks," in *Proc. 15th Int. Parallel Distrib. Process. Symp. (IPDPS)*, Apr. 2001, p. 186.

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

[23] S. Ping, "Delay measurement time synchronization for wireless sensor networks," Intel Res. Berkeley Lab, Berkeley, CA, USA, Tech. Rep. IRB-TR-03-013, 2003.

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

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

[26] S. Yoon, C. Veerarittiphan, and M. L. Sichitiu, "Tiny-sync: Tight time synchronization for wireless sensor networks," *ACM Trans. Sensor Netw.*, vol. 3, no. 2, 2007, Art. ID 8.

[27] Q. M. Chaudhari, E. Serpedin, and K. Qaraqe, "On maximum likelihood estimation of clock offset and skew in networks with exponential delays," *IEEE Trans. Signal Process.*, vol. 56, no. 4, pp. 1685–1697, Apr. 2008.

[28] J. Chen, Q. Yu, Y. Zhang, H.-H. Chen, and Y. Sun, "Feedback-based clock synchronization in wireless sensor networks: A control theoretic approach," *IEEE Trans. Veh. Technol.*, vol. 59, no. 6, pp. 2963–2973, Jul. 2010.

[29] F. Ren, C. Lin, and F. Liu, "Self-correcting time synchronization using reference broadcast in wireless sensor network," *IEEE Wireless Commun.*, vol. 15, no. 4, pp. 79–85, Aug. 2008.

[30] J. Song, S. Han, A. K. Mok, D. Chen, M. Lucas, and M. Nixon, "WirelessHART: Applying wireless technology in real-time industrial process control," in *Proc. IEEE Real-Time Embedded Technol. Appl. Symp. (RTAS)*, Apr. 2008, pp. 377–386.

[31] A. Demir, "Phase noise and timing jitter in oscillators with colored-noise sources," *IEEE Trans. Circuits Syst. I, Fundam. Theory Appl.*, vol. 49, no. 12, pp. 1782–1791, Dec. 2002. [Online]. Available: http://dx.doi.org/10.1109/TCSI.2002.805707

[32] D. W. Allan, "Statistics of atomic frequency standards," *Proc. IEEE*, vol. 54, no. 2, pp. 221–230, Feb. 1966.

[33] M. Lombardi, "The accuracy and stability of quartz watches," *Horological J.*, vol. 150, no. 2, pp. 57–59, 2008.

[34] M. Nakazawa, Y. Nakamura, and S. Miyashita, "Frequency-temperature characteristics of quartz crystal flexure bars and quartz crystal tuning forks," *IEEE Trans. Sonics Ultrason.*, vol. 26, no. 5, pp. 369–376, Sep. 1979.

[35] P. Marchetto, A. Strickhart, R. Mack, and H. Cheyne, "Temperature compensation of a quartz tuning-fork clock crystal via post-processing," in *Proc. IEEE Int. Freq. Control Symp. (FCS)*, May 2012, pp. 1–4.

[36] S. E. Mattsson, H. Elmqvist, and M. Otter, "Physical system modeling with Modelica," *Control Eng. Pract.*, vol. 6, no. 4, pp. 501–510, 1998.

[37] P. Fritzson, *Principles of Object-Oriented Modeling and Simulation With Modelica 3.3: A Cyber-Physical Approach*. New York, NY, USA: Wiley, 2014.

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

[39] F. Terraneo, A. Leva, S. Seva, M. Maggio, and A. V. Papadopoulos, "Reverse flooding: Exploiting radio interference for efficient propagation delay compensation in WSN clock synchronization," in *Proc. 36th IEEE Real-Time Syst. Symp.*, Dec. 2015, pp. 1–10.