DiStiNCT: Synchronizing Nodes with Imprecise Timers in Distributed Wireless Sensor Networks

James A. Boyle, Jeff S. Reeve, *Senior Member, IEEE* and Alex S. Weddell, *Member, IEEE*

***Abstract*—An effective and robust time synchronization scheme is essential for many wireless sensor network (WSN) applications. Conventional synchronization methods assume the use of highly accurate crystal oscillators (10–100 ppm) for synchronization, only correcting for small errors. This paper suggests a novel method for time synchronization in a multi-hop, fully-distributed WSN using imprecise CMOS oscillators (up to 15,000 ppm). The DiStiNCT technique is power-efficient, computationally simple, and robust to packet loss and complex topologies. Effectiveness has been demonstrated in simulations of fully connected, grid and uni-directional ring topologies. The method has been validated in hardware on a grid of nine sensor nodes, synchronizing to within a mean error of 6.6 ms after 40 iterations.**

***Index Terms*—wireless sensor networks, distributed algorithms, synchronization**

I. Introduction and Related Work

TIME SYNCHRONIZATION is essential in wireless sensor networks (WSNs) for tasks including Time Division Multiple Access (TDMA), timestamping, and duty cycling [1]. Crystal oscillators are normally used for precision timing, but microcontroller units (MCUs) are usually equipped with on-chip CMOS oscillators which may be used for timer-based interrupts and the system clock. CMOS oscillators typically have an accuracy in the range 5,000–15,000 ppm [2], but their advantages over crystal oscillators include reduced power requirements, improved ruggedness and the ability to integrate on the same IC. These properties make CMOS oscillators more attractive than crystals for future WSN applications, e.g. Smart Dust [3], which may use thousands of tiny low-cost nodes. Current literature on WSN time synchronization does not consider using imprecise oscillators. Existing schemes make small, infrequent adjustments and are not intended to synchronize when there are large clock offsets and drifts. A scheme using imprecise timers should be distributed i.e. reach a consensus on timing, not rely on a single timer which may be unreliable. A synchronization algorithm for crystal-less, resource-constrained WSNs should be suitable for use with CMOS oscillators, but have low code size and computational complexity.

Manuscript received 4 February 2016; revised 17 August 2016, 16 December 2016, 6 February 2017; accepted 8 February 2017. Date of current version 28 March 2017. The authors are with Electronics and Computer Science, University of Southampton, U.K. (e-mail: jab1g12@ecs.soton.ac.uk, jsr@ecs.soton.ac.uk, asw@ecs.soton.ac.uk)

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

Experimental data used in this paper can be found at DOI:10.5258/SOTON/D0021 (http://doi.org/10.5258/SOTON/D0021)

Established synchronization algorithms include Reference Broadcast Synchronization (RBS), Timing-Sync Protocol for Sensor Networks (TPSN) and Tiny Sync [1]. Important considerations include whether a scheme is peer-to-peer or broadcast based, suited to a stationary or mobile network, and synchronous or asynchronous [4]. A synchronization scheme may be suited to a multi-hop network topology and can be assessed by its scalability and power efficiency. Also, some schemes are fully distributed whereas others require a hierarchy of nodes. In a fully distributed method there is no managing node (i.e. hub or reference node), and sensing and computation tasks are shared between nodes in the network. Distributed networks, where hundreds or thousands of low-cost nodes may be deployed [5], are expected to improve robustness but introduce challenges in data fusion and information processing [6]. There is a small body of work on distributed algorithms, these often use pair-wise communications where messages are exchanged between pairs of nodes. This is an inefficient mode of communication; if nodes broadcast to all neighbors at once, fewer transmissions are required, saving power and time. Suggested synchronization algorithms often make use of a single reference node, or a hierarchy of nodes [7][8].

Schemes such as the Flooding Time Synchronization Protocol (FTSP) rely on a network hierarchy in multi-hop topologies [8]. This requires the election of a reference node from which all other nodes synchronize. In another scheme, which is truly distributed, nodes collectively agree a local time by averaging their time differences [9]. This hardware-based approach, a pulse-coupled synchronization method, has been proposed for time synchronizing sensor nodes in a distributed network. Here, nodes transmit pulses over a radio channel; the phase difference between the pulses and the clock is measured at each receiving node, and phase-locked loops are used to make timing adjustments. Pulse-coupled synchronization has also been shown to be effective for heterogeneous networks with two classes of nodes, where nodes may have different power requirements [10]. Earlier research done on multi-agent consensus using algebraic graph theory mathematically proves that convergence is guaranteed for strongly-connected networks [11]. A synchronization algorithm has been implemented for small, fully-connected distributed networks [12]. Due to the high reliability of the crystal oscillators used, only small corrections are needed (max. 1 clock cycle per frame). There has been little consideration of multi-hop topologies.

Time synchronization methods typically use a software-based clock, described as a 'logical' clock [7][13][14]. When

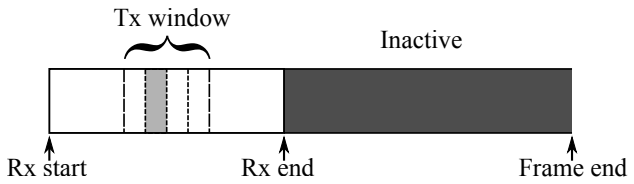


Fig. 1: Frame structure. Dashed lines: TDMA transmit slots—a suggested method for organizing transmits, white blocks: transceiver in Rx mode, light gray block: transceiver in Tx mode, dark gray slot: inactive period in which nodes may enter a low power mode.

maintaining a logical clock, the resolution is limited to the 'tick' interval. If the resolution is fine-grained, the node must wake up frequently to update its clock, consuming power. The Reach-back Firefly Algorithm (RFA) takes inspiration from nature, mimicking the synchronized pulsing of fireflies [15]. RFA is non-linear, requiring relatively expensive computations at each node. It only synchronizes transmissions and does not provide a continuous time reference for interrupts and timestamping. Work on the Proportional-Integral Clock Synchronization algorithm (PISync) suggests that proportional and integral control should be applied to correct both phase and drift errors [14].

In Adaptive Synchronization, nodes estimate the clock drift of their neighbors by measuring timing offsets. The network is synchronized by adjusting clock frequencies to compensate for these drifts [16]. However, this has only been implemented on a pair of nodes and does not consider the effect of different topologies. A Kalman filter may be used to reduce errors from quantization in multi-hop networks with long paths [17]. However, when using imprecise oscillators quantization errors become insignificant. The Try-Once-Discard protocol has been implemented on multi-hop WSN using Black Burst Synchronization (BBS) to perform time synchronization [18]. BBS avoids clashing transmissions by representing bits using a sequence of pulses. This scheme elects a master node so is not fully distributed, and may not work where there is a large uncertainty in ticks.

Least squares linear regression is often used to estimate clock drifts when performing time synchronization [7][8][19]. Least squares estimation is done by measuring the relative error over a number of frames, which is assumed to increase linearly. Using a least squares method, a constant gradient and intercept is fitted to the data. Advantages include increased accuracy and resilience to noise in measurements. However, linear regression requires more computation and space than just estimating drift across consecutive frames. Also, the assumption that error increases linearly may not hold in adverse conditions, for example if the temperature is changing. Alternatively, timer corrections may be separated into phase offset and drift estimations [7]. It is proposed that the drift estimate should be constantly updated using a feedback loop. This provides a drift estimate and accurately tracks any changes in clock drift.

This paper proposes a novel algorithm, DiStiNCT (**Di**stributed **S**ynchroniza**ti**on of **N**odes with **C**MOS **T**imers) for synchronizing nodes in a distributed WSN. In the algorithm, nodes calculate their average timing offset by comparing timestamps included in broadcasts to timer measurements. Nodes converge towards a globally agreed time by correcting for phase offsets and clock drift. A detailed explanation and mathematical analysis of the algorithm has been carried out (Section II). DiStiNCT uses a similar method to a distributed pulse-coupled algorithm, which is mathematically proven to synchronize a connected network [9]. Both algorithms reach consensus on timing by nodes averaging time differences and adjusting their timing periods. However in DiStiNCT, which is packet-based, nodes broadcast timestamps instead of radio pulses. Also, DiStiNCT considers the timing error as a sum of a phase offset and clock drift whereas the pulse-based scheme uses a second-order phase-locked loop.

The proposed algorithm has been modeled and simulated for a range of network sizes and topologies (Section III). The performance of the algorithm has been assessed for fully-connected, grid, and ring networks of various sizes. The effect of packet loss was explored and the performance of different parameter choices measured. The method has been verified in simulation on complete, grid, and ring topologies, on networks of up to 100 nodes. It has been validated in hardware (Section IV) and has low computational and space requirements: an implementation on Texas Instruments eZ430-RF2500 sensor nodes occupied only 1316 bytes of non-volatile memory. Synchronization was achieved to a mean error of $6.6\,\mathrm{ms}$, with a standard deviation of $6.3\,\mathrm{ms}$, after 40 iterations; without synchronization, oscillator drift would have increased the average timing error by $74.6\,\mathrm{ms}$ every frame.

II. Distributed Synchronization Algorithm

We propose a method for synchronization of imprecise, low-frequency hardware timers. The advantage of this method is that no crystal oscillator is required. The scheme is robust to different clock drifts, network topologies and packet losses and has low computational and space overheads.

A. Algorithm Description

Each node divides time into blocks, which are referred to as *network frames*. The frame structure is shown in Fig. 1. Each node transmits only once within each frame. In the proposed method, nodes use timestamps to measure the offsets between neighbors' hardware timers. During each frame these differences are averaged and used to adjust the timing so that it converges towards a local average. Here, we will describe an ideal method which has been shown to be stable for all tested network topologies and sizes. Later, a practical adjustment to the algorithm is suggested which allows the network to update on every frame and converge more quickly.

In this method, all nodes observe for a fixed number of frames between corrections. The algorithm has three stages which are executed every frame:

1) Turn on the transceiver and initialize frame variables e.g. average time differences.
2) a) Measure timings from other nodes.
   b) Broadcast a timestamp once to the network.
3) Make correction to timing offset and turn off transceiver.

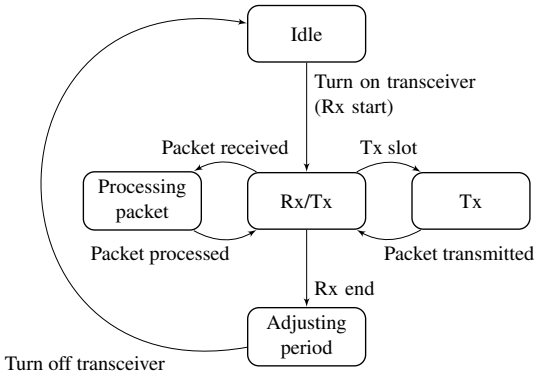


Fig. 2: DiStiNCT's control flow. For efficient implementation, state transitions should be triggered by interrupts.

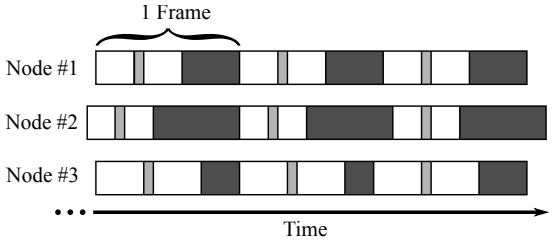


Fig. 3: Synchronization of a connected network of 3 nodes over 3 frames. White: nodes are able to receive broadcasts, light gray: transmit (Tx) window in which each node has a slot to broadcast to the network, dark gray: inactive i.e. transceivers turned off.

The operation of each node may be represented using a state diagram (Fig. 2). Fig. 3 illustrates the algorithm running on a small network of 3 nodes—showing how nodes adjust their inactive period lengths to compensate for clock drift and offsets between each other. After 3 iterations the transmit (Tx) windows are aligned, i.e. all 3 nodes are synchronized.

For each discrete frame, i, node n includes a timestamp, $t_n(i)$, in its transmission packet header; the packet payload may contain useful data related to the network application. When a node n receives a broadcast from node m, it measures its timer value and compares this with the timestamp. Each of the N nodes keeps a running average of these time differences over the frame. At the end of the frame, the average offset (ϵ_n) represents a node's time offset, i.e. error relative to the frame.

$$\epsilon_n(i) = \frac{1}{N-1} \sum_{m \neq n}^{N} t_n(i) - t_m(i) \tag{1}$$

On certain frames, nodes make corrections according to their offset with the network. By correcting for these differences, all nodes converge to a common time. However, instead of adjusting the timer directly, the frame *period* is changed. For example, this may be done by writing to the timer compare register, avoiding changing the phase of the timer. This is analogous to the phase-locked loop (PLL) which synchronizes the phase of an oscillator to a reference by varying the oscillator frequency. A benefit of this method is that the timer count is continuously incrementing, i.e. is monotonically increasing, making it suitable for timestamping events and setting hardware interrupts. If there were discontinuities in the timer count, the system could skip past interrupts, causing scheduled events to be missed. The error correction process may be separated into two components: *phase error correction* and *oscillator drift estimation*.

Phase error correction attempts to match the phase of a node's clock with its neighbors. This correction is proportional to the average error observed by the node: $k_{phase}\epsilon_n(i)$. This simple correction is sufficient for nodes to converge to a steady state error in any strongly connected topology, although with clock drifts this results in a phase offset at each node [9].

Oscillator drift describes a difference in the oscillator frequency from the expected value, causing a long-term 'drift' of the clock from the reference. The drift estimator compensates for this drift, relative to other nodes in the network. To measure drift (d_n), nodes can agree to simultaneously observe for a fixed number of frames in between corrections. While observing, no phase error corrections are made. In the simplest case, nodes correct on every other frame e.g. on even frame numbers. These observation frames allow nodes to measure the change in timer offset over time, i.e. the clock drift. If nodes observe for more than one frame between updates they should use a least-squares estimator with linear regression to fit a gradient and offset to the measured data.

Linear regression assumes that clocks drift apart at a constant rate, however the drift is likely to be time-variant. To track changes, the drift estimate is improved incrementally using a feedback loop. The drift estimate converges towards the actual drift at a rate decided by the parameter k_{drift}.

$$d_n(i) = d_n(i-1) + k_{drift}(\epsilon_n(i) - \epsilon_n(i-1)) \tag{2}$$

Nodes are initialized with a default period length, T. The final equation for the adjustment at each node, x_n is given by Eq. 3, with the period length, T_n given by Eq. 4.

$$x_n(i) = k_{phase}\epsilon_n(i) + d_n(i) \tag{3}$$

$$T_n(i) = T + x_n(i) \tag{4}$$

Effective duty cycling is key to a power efficient system, therefore nodes should turn off transceivers and power down their MCUs wherever possible. Within each frame is a small window for transmitting (Tx), and an inactive period. TDMA is used to organize broadcasts. If nodes are unsynchronized, it is highly likely that nodes will broadcast while other nodes are inactive. Therefore, a *guard interval* (or guard period) must be added either side of the transmit window in which nodes may still receive messages. The guard interval for a given node should be no smaller than the maximum timing error observed at that node. A target of this synchronization scheme is for the timing error to be shorter than one frame period, so that the guard period may be minimized.

The propagation and processing delays result in a constant error offset appearing at every node. To reduce this delay, time stamping should always be done as close as possible to transmission and receiving. However, in some cases the delay error may be significant relative to the network period. If

greater accuracy is required, this delay should be compensated for. When packets are a fixed size it may be possible to measure the propagation delay in clock cycles and subtract it from the final average timer difference. Otherwise, the delay could be estimated based on the measured delay per byte.

Each node initializes in an unsynchronized state. In this state the node does not duty cycle. For a node to enter the synchronized state it must receive a broadcast from a neighboring node. The node then writes the received timestamp directly to its timer and enters the synchronized state (this is the only time where a node can directly set its timer value). If a node fails to receive from any neighbors in a given number of frames, it times out and re-enters the unsynchronized state.

If the network observes for longer time periods between corrections, the drift estimate becomes less affected by noise in measurements. However, this also means nodes must wait longer between correcting errors. Although observing over more frames may increase the drift estimate accuracy, it was found in simulation that nodes converged more slowly when they spent longer observing. Also, this technique performs poorly in time-variant topologies; packet losses destabilized networks which observed between timing corrections more than networks that corrected after each frame.

B. A Practical Adjustment

Nodes can measure their clock drifts if the network does not correct timings for one or more frames. However, correcting less frequently often increases timer uncertainty and instability in time-variant topologies. If nodes correct at the maximum rate, i.e. every frame, the timer uncertainty is minimized and nodes' timers converge more quickly and reliably.

An adjustment may be made to the algorithm so that nodes correct after every frame. Instead of observing the network over one or more frames, nodes *predict* what the error would have been (ϵ'_n) had they and their neighbors not made their previous phase corrections. This requires estimating the unadjusted timing of each node, t'_n.

$$t'_n(i) = t_n(i) - k_{phase}\epsilon_n(i-1) \tag{5}$$

$$\epsilon'_n(i) = \frac{1}{N-1}\sum_{m\neq n}^{N} t'_n(i) - t'_m(i) \tag{6}$$

Using this prediction, along with the previous measured average error, it is possible for nodes to estimate their clock drift d'_n (Eq. 7). To implement this, nodes must store and broadcast their previous error, $\epsilon_n(i-1)$.

$$d'_n(i) = d_n(i-1) + k_{drift}(\epsilon'_n(i) - \epsilon_n(i-1)) \tag{7}$$

Whenever a node shares a timer value with another node, there is an error introduced due to the clock drifts between nodes. For example, consider the scenario where the clock of node #1 is running twice as fast as the clock at node #2. If node #1 transmits an error of 50 clock cycles, this corresponds to $50/2 = 25$ clock cycles at node #2. Unless both nodes can measure their drifts exactly, it is impossible to correctly relate timer values between nodes. This must be considered when using the suggested drift estimation algorithm, as nodes must share their (previous) errors with each other. Although it was relatively insignificant, there is always an error proportional to the drift ratios in the prediction of unadjusted error.

C. Mathematical Analysis

The proposed algorithm is linear, therefore the network may be formulated as a system of difference equations. Spectral analysis decomposes a matrix into a set of eigenvalues and eigenvectors. Eigenvalues of a system matrix are an objective metric for performance; for a discrete-time system to be stable, all eigenvalues must lie in the unit circle (or equivalently $|\lambda| < 1$). The following section analyses the modified version of the algorithm, introduced in Section II-B. The use of difference equations and eigenvalues was inspired by in-depth analysis of a similar synchronization scheme based on transmitting and observing pulses [9].

Considering Eq. 4–6 for the adjusted algorithm, a node's timer value may be described as two first-order difference equations, $t_n(i)$ and $d_n(i)$ (Eq. 2). Note that jitter is ignored, and it is assumed that the drifts are time invariant. However, jitter and other nonlinear effects are considered later.

$$t_n(i+1) = t_n(i) + T_n + k_{phase}\epsilon_n(i) + d_n(i) \tag{8}$$

T_n is the frame period of a given node, which may vary between nodes due to clock drift. By performing some algebraic manipulation (Eq. 9), the two equations for $t_n(i)$ and $d_n(i)$ may be combined into a single second-order equation (Eq. 10).

$$\begin{aligned} t_n(i+1) - t_n(i) =& t_n(i) - t_n(i-1) + \\ & k_{phase}(\epsilon_n(i) - \epsilon_n(i-1)) + \\ & d_n(i) - d_n(i-1) \end{aligned} \tag{9}$$

Rearranging and substituting for $d_n(i) - d_n(i-1)$ yields:

$$\begin{aligned} \implies t_n(i+1) =& 2t_n(i) - t_n(i-1) + \\ & k_{phase}(\epsilon_n(i) - \epsilon_n(i-1)) + \\ & k_{drift}(\epsilon'_n(i) - \epsilon_n(i-1)) \end{aligned} \tag{10}$$

Although the period term (T_n) has been eliminated, it is implied in the initial conditions, i.e. $t_n(0) = 0, \quad t_n(1) = T_n$. Synchronization is equivalent to convergence in this system. As the system is linear it may be represented as a matrix—lending itself to spectral (eigenvalue) decomposition.

The next step closely follows the analysis of the pulsed-based scheme [9]. The first-order and second-order terms of Eq. 10 are separated to create the first-order and second-order system matrices, $\mathbf{A}$ and $\mathbf{B}$ respectively. The system matrix is then given by the partitioned matrix in Eq. 11, in which $\mathbf{I}$ is the identity matrix and $\mathbf{0}$ is the zero matrix. $\mathbf{A}$, $\mathbf{B}$, $\mathbf{I}$ and $\mathbf{0}$ are all $N \times N$ square matrices.

$$\begin{bmatrix} \mathbf{t}(i+1) \\ \mathbf{t}(i) \end{bmatrix} = \begin{bmatrix} \mathbf{A} & \mathbf{B} \\ \mathbf{I} & \mathbf{0} \end{bmatrix} \begin{bmatrix} \mathbf{t}(i) \\ \mathbf{t}(i-1) \end{bmatrix} \tag{11}$$

Without adjustment, this system describes the error relative to the perfect reference time. Without access to this reference the nodes can only converge to a shared concept of time. This

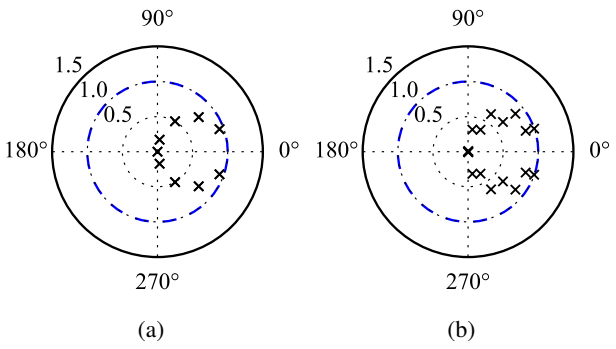


Fig. 4: Eigenvalues from analysis of ring networks of 9 nodes that (a) can perfectly predict their unadjusted error, $k_{phase} = k_{drift} = 0.5$, (b) can *not* perfectly predict their unadjusted errors due to clock drift.

offset between the network and the reference results in at least one eigenvalue at 1. However, the result of interest i.e. the convergence of a node to a shared network time, $\hat{t}_n(i)$, can be obtained by simply subtracting the average timestamp, $\bar{t}(i)$.

$$\hat{t}_n(i) = t_n(i) - \bar{t}(i) = t_n(i) - \frac{1}{N}\sum_{m}^{N} t_m(i) \qquad (12)$$

Wherever the difference between times at two nodes is calculated i.e. $t_n(i) - t_m(i)$, this steady state correction cancels out. Therefore, the matrix adjustments are simply: $\mathbf{A}' = \mathbf{A} - \frac{2}{N}, \quad \mathbf{B}' = \mathbf{B} + \frac{1}{N}$.

As described in Section II-B, whenever nodes share timer values there is an error introduced. When constructing the system matrix this may be simulated by scaling any shared timer values by a random amount. This corresponds to the ratios of clock drifts between nodes. Fig. 4a shows the eigenvalues of a unidirectional ring topology with no drift, Fig. 4b shows the same network but with significant drift present. The error introduced by clock drift scatters the eigenvalues from their ideal positions. The scattering explains the increased instability caused by clock drifts in this algorithm.

In a discrete linear system, the pole with the largest magnitude dominates the response. The poles correspond directly to the eigenvalues of the system matrix. Therefore, the magnitude of the largest eigenvalue, i.e. the spectral radius (ρ) determines whether the system is stable, and if so the speed at which it converges. For a discrete system, the complex argument of a pole corresponds to the oscillatory frequency of that response. Eigenvalues along the positive real axis represent poles with no oscillatory response.

III. Simulation Results

Spectral analysis evaluates performance and stability, but fails to consider the effect of nonlinearities such as clock jitter and propagation delay. It is also difficult to analyze time-variant drifts and topologies e.g. unreliable links between nodes. To address this, a detailed simulation was set up to model the performance of each node and link. Time is measured relative to the perfect reference timer so that 1 corresponds to a frame period. A clock cycle is not generally equivalent for different nodes, this needs to be considered whenever nodes are sharing a timing value e.g. their previous errors. The simulation also considers the effect of phenomena such as propagation time delay and clock jitter.

The simulation may be split into four main components: (1) error measurement, (2) drift estimation, (3) error correction *or* period adjustment, and (4) timer updates.

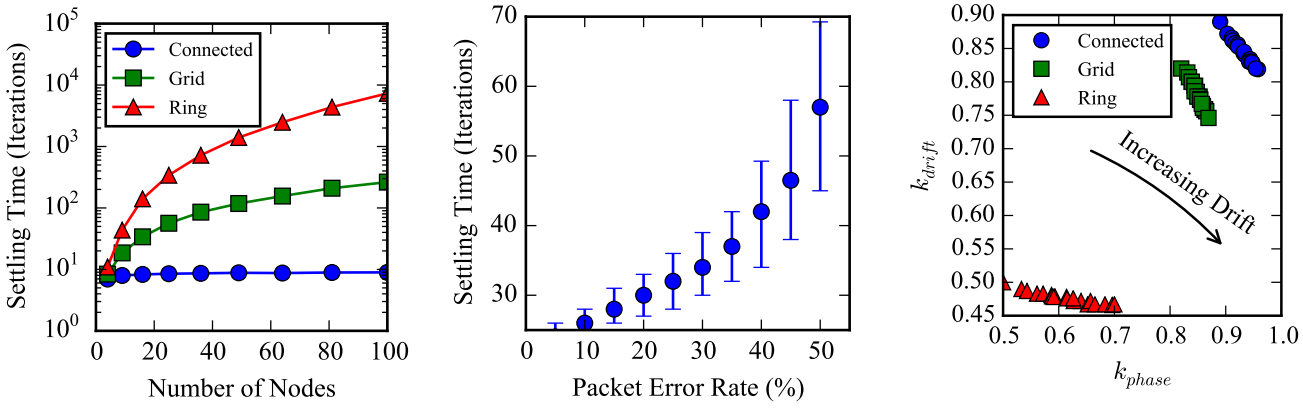
Error measurement is modeled by calculating the errors at every node for each iteration of the algorithm i.e. the error after every frame. The network topology is represented by an adjacency matrix, which may be different on each iteration. Nodes are only able to measure differences between themselves and their neighbors. Error correction is then applied at each node using the current, and possibly previous, errors according to the method.

When updating the timer, a number of factors may be considered. During a frame, a random amount of clock jitter (random error) is introduced. This is modeled using a normally distributed random variable, with a mean of 0 and standard deviation set to a measured value. Each node is assigned a drift value which is added after each frame. This drift is measured as an offset from the reference (perfectly matching frequencies result in a drift of 0 in the simulation). The drifts are randomly generated across the working range of frequencies given by the MSP430 data-sheet [20]. The model may add a fixed propagation delay, taken from empirical measurements.

Three different network topologies were simulated. The fully connected topology represents a network where any node has direct links with all other nodes in the network—this is a best-case scenario. The uni-directional ring network is a worst-case topology, in which each node has only one incoming and outgoing edge. The grid topology is a reasonable middle ground, in which the network is a square lattice of nodes and all links are bidirectional.

The settling times for networks of different sizes and topologies were simulated (Fig. 5a). The results verify the algorithm is stable for larger networks in realistic topologies. The grid network of 100 nodes converges more slowly than smaller networks, taking approximately 200 iterations to read a steady state. For all results the clock jitter was modeled as a normally distributed random variable with standard deviation of 40 ns. As there is no data-sheet value for jitter for the MSP430s CMOS oscillator, a value was taken from measurements by observing the oscillator of a eZ430-MSP430 sensor node at room temperature and $V_cc = 3.3$ V. A histogram of clock timings showed that the period is normally distributed about an average. This means that the majority of clock ticks lie within one standard deviation of the average period, with ticks statistically independent from one another.

Unreliable links were simulated by varying the topology over the course of the simulation. Packet loss was modeled by using a fixed probability of error for each link. The settling time was measured for a 3×3 grid and a range of packet loss rates (Fig. 5b). It was found that with increased packet loss the mean settling time increased. A lost packet may be seen as equivalent to a node dropping out for a single frame. A node that fails to rejoin will not destabilize the network, as adverse topologies were shown to converge and smaller networks are generally more stable than larger ones. A simulation confirms



(a) (b) (c)

Fig. 5: Simulation results (a) network settling time ($k_{phase} = 0.5, k_{drift} = 0.25$), defined as the time required for the timing offset between any two nodes, i.e. error, to reach and stay within 10% of the maximum error; (b) median settling time ($k_{phase} = 0.5, k_{drift} = 0.25$) for a 9-node grid with various link qualities, error bars show interquartile range after 500 repeats; (c) impact of unadjusted timing estimation on the optimum parameters, performed with 9-node networks for different topologies and clock drifts (spectral radius was used to locate the optimum parameter in a grid of k_{phase} and k_{drift} values), optimum parameter choices are plotted for different drift values.

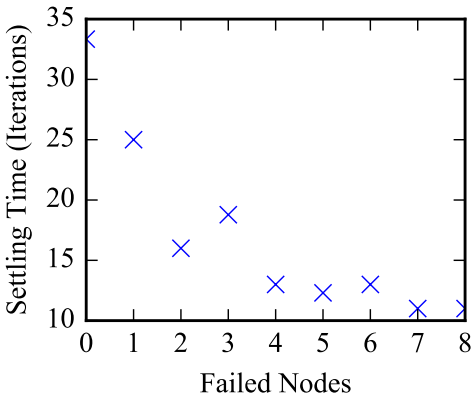


Fig. 6: Effect of node drop-out on synchronization of a 9-node grid. After 10 iterations, a given number of nodes fail and leave the network. The settling time of the remaining nodes was measured.

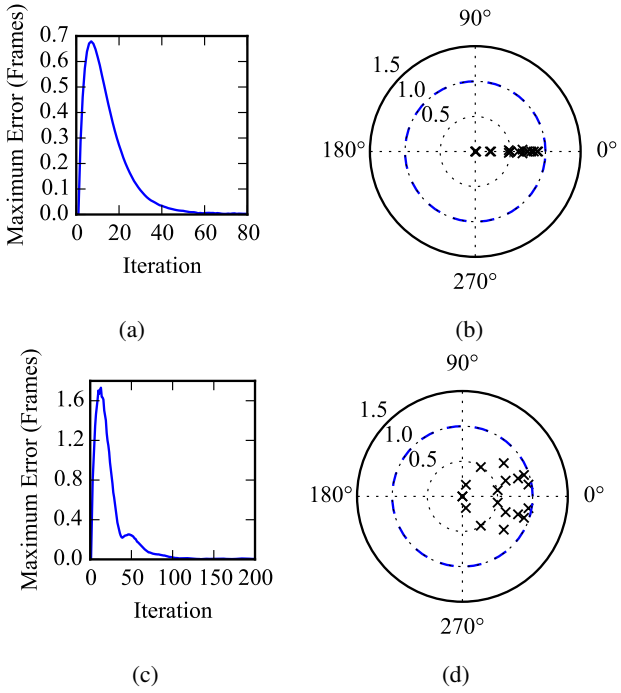


Fig. 7: Simulation results with $k_{phase} = 0.5, k_{drift} = 0.25$ for: a 9-node grid (a) maximum timing error (b) eigenvalues; a 9-node unidirectional ring (c) maximum timing error, (d) eigenvalues.

that drop-out does not adversely affect synchronization of the remaining nodes (Fig. 6).

The errors in a 3×3 grid network, Fig. 7a, and ring network, Fig. 7c, were also simulated. The spectral analysis of the same networks are shown in Fig. 7b and Fig. 7d. The simulations also make some assumptions and simplifications. Transmissions are modeled as occurring synchronously. In reality, nodes will exchange messages over their Rx period and transmit in different time slots. In the simulation, time is represented using floating point precision. A value of '1' corresponds to one frame period. However, in the hardware implementation, time is measured in integer clock cycles. Also, the simulation does not consider duty cycling and the effect that this has on performance.

The effect of parameter choice and amount of drift on time synchronization was explored. It was found that certain parameter choices may be unstable (Fig. 8). Using spectral analysis, the optimum parameters were found for a range of maximum clock drifts (Fig. 5c). As the clock drift was increased, the optimum parameter choice changed. When there is no clock drift, the optimum parameters lie along the line $k_{phase} = k_{drift}$. However, as the clock drift is increased this choice is no longer optimum. Considering these results, there is no optimum choice of parameters for an arbitrary network. Instead, k_{phase} and k_{drift} should be chosen as appropriate for

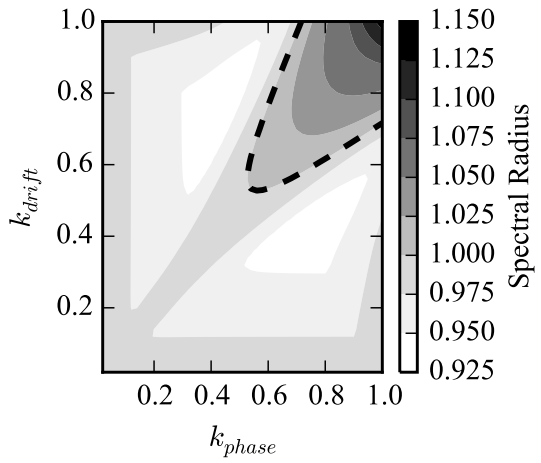


Fig. 8: Spectral radius of the adjusted algorithm for a range of parameters (9-node unidirectional ring network, clock drifts randomly distributed $\pm 66.7\%$). Dashed line outlines region where spectral radius > 1, showing parameter choices which result in an unstable network (implying time synchronization will not be achieved).

the expected network size, topologies, chosen method and the drifts expected. The parameters $k_{phase} = 0.5, k_{drift} = 0.25$ are stable for a wide range of drifts, network sizes and topologies while remaining efficient to implement.

These results show that the performance of the algorithm depends on the choice of the parameters k_{phase} and k_{drift}. If nodes are able to measure their unadjusted error, $\epsilon'_n(i)$, exactly, the network will be stable for all choices of parameters within the bounds $0 < k_{phase} < 1, \quad 0 < k_{drift} < 1$. If the unadjusted error is estimated, there is a small error which may affect stability. It has been shown that the optimum parameters depend on the network size, topology and amount of clock drift. Generally, smaller parameter choices improve stability although the timings converge more slowly. The unidirectional ring may be used with the maximum expected drifts as a worst case test, but the algorithm should be optimized for a realistic topology e.g. grid. Also, if $k_{phase} = 2^{-x}, \quad k_{drift} = 2^{-y}$ then the scaling may be implemented as a bit-shift, which is efficient compared to an integer division.

The algorithm was tested observing for a different number of frames between corrections. It was found that increasing the number of observation frames resulted in slower convergence (Fig. 9a). An improvement in accuracy was seen going from zero to one observation frame (Fig. 9b). However, observing for longer gave no improvement. Also, it was found that observing between corrections may result in instability in time-variant topologies. When there are link errors, a network that observes between corrections may be unstable (Fig. 9c). If the topology changes over consecutive frames, the gradient of these errors may no longer correspond to the clock drift.

IV. Hardware Validation Results

The algorithm was validated in hardware using the eZ430-RF2500 development kit, which uses an MSP430 MCU [20]. The MSP430 supports a 'count up' timer mode, which counts up to a value (held in a register) before resetting to 0 and triggering an interrupt. Each node may broadcast at most once every network frame. A timer is used to generate a periodic interrupt after a desired amount of time, referred to as the frame period. One timer was used with three timer interrupts available. The first timer register was used to set to the frame period. This is the value that the timer counts up to for each frame. A timer value of '0' corresponds to the start of the Tx window i.e. the first TDMA slot.

The second compare register was used to time the broadcast within the transmit window. Each node used a unique identifier to calculate its TDMA slot and therefore interrupt value. For example, if the slot number is 2, and the slot duration is 150 clock cycles, the register would be set to $2 \times 150 = 300$ (clock cycles). Although beyond the scope of this work, in a multi-hop network it will be important to consider how the transmissions are scheduled—for example in a simple case, slots could be randomly self-assigned and clashes ignored.

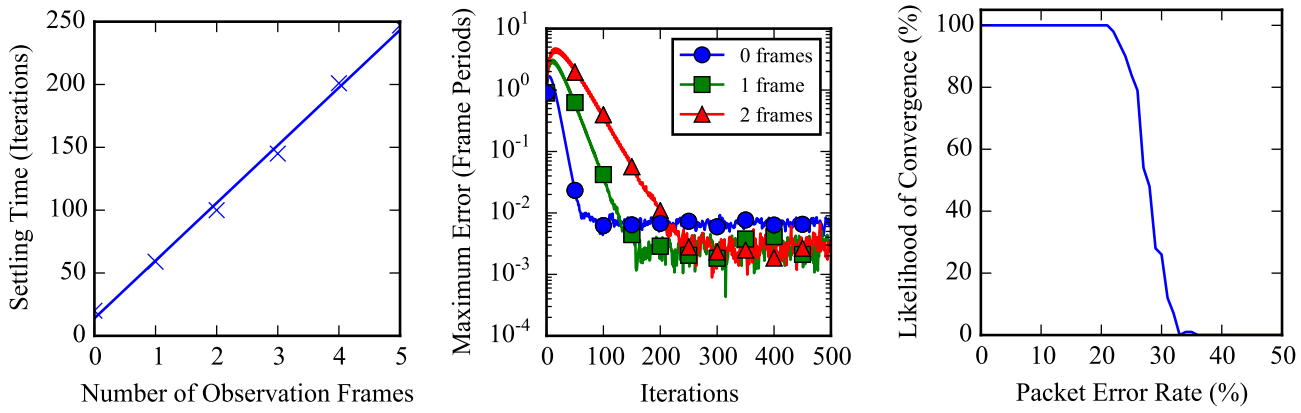
The third compare register (`TACCR2`) was used to manage the duty cycling of the node. The function of this register was alternated between starting the next frame, and ending the current frame. If the node was inactive, this interrupt would wake up the node and start the next frame. For example, if the frame is 36,000 cycles long with a 50% duty cycle, `TACCR2` would be set to a value of $36,000 - \frac{18,000}{2} = 27,000$. This was done to wake up the node before the start of the Tx window.

Once turned on, the same register was reused to turn the node off and end the frame. Using the example above, `TACCR2` would now have a value of 9,000. As the node turned off the register was changed back, ready to wake the node up again. There would then be a period of inactivity before the process repeated and then next frame began[1].

Tests were performed with a 3×3 grid of sensor nodes. The grid topology was created artificially by whitelisting nodes. Fig. 10 shows the algorithm running for 20 frames, after which the network is synchronized to a maximum error of $290\,\mathrm{ms}$ (9.8% of the frame period which is $3\,\mathrm{s}$). After 40 frames, nodes synchronized with a maximum error of $22.2\,\mathrm{ms}$ (0.74% of frame period), and a mean error of $6.6\,\mathrm{ms}$ (0.22% of frame period) with a standard deviation of $6.3\,\mathrm{ms}$. Fig. 11 shows the absolute timing error measured at a single node, showing how nodes track their timings relative to the network. In tests, nodes were sporadically reset or isolated from the network and resynchronized. Rejoining nodes detect that they have dropped out and did not destabilize the network.

Energy consumption has been observed on similar hardware using crystal oscillators and TPSN. Power usage was dominated by the radio transceiver which was active roughly 0.01% of the time [21]. If using DiStiNCT, up to 34% energy saving is possible by using a CMOS oscillator rather than a crystal [22], although owing to its poorer timing accuracy the transceiver would have to be active for longer. To mitigate

[1]The observed clock drifts were small enough for nodes to compare timer register values and not require a coarse timer e.g. a frame counter. If larger drifts were observed it may be possible to have an error larger than 1 frame. To avoid overflow in this case nodes would also need to use coarse synchronization [12].



(a) (b) (c)

Fig. 9: Simulated performance with a 9-node grid (a) settling time for different numbers of observation frames, $k_{phase} = k_{drift} = 0.5$ (when 2 or more observation frames are simulated, linear regression has been used to estimate the drift), (b) maximum timing error where different number of frames are observed between corrections, (c) likelihood of time synchronization for different packet error rates, when 1 observation frame is used between corrections, using parameters $k_{phase} = 0.5, k_{drift} = 0.25$ (showing that a network using linear regression and packet loss >30% is unlikely to synchronize).

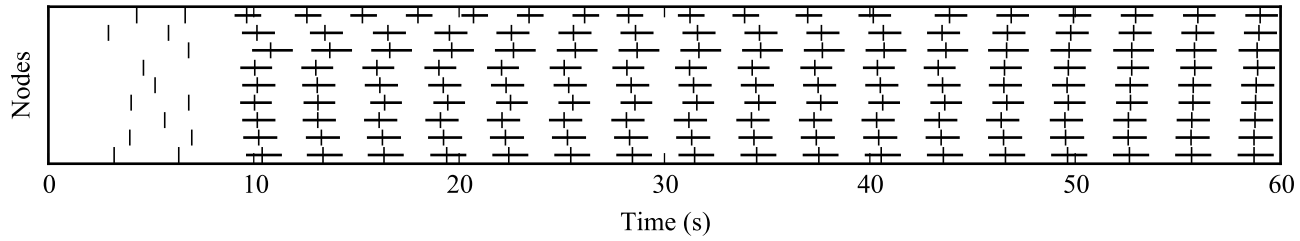


Fig. 10: Results of hardware verification with 3×3 grid. Vertical bars show the start of each frame, horizontal lines indicate when nodes are active, i.e. when transceivers are powered on. Frame period is 36,000 cycles—3 s at 12 kHz, $k_{phase} = k_{drift} = 0.5$.

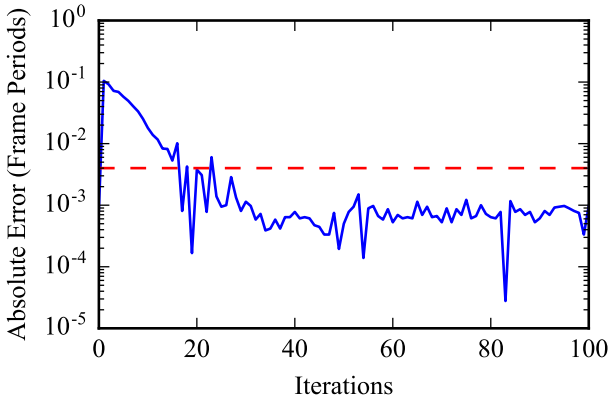


Fig. 11: Results from hardware experiments, showing the absolute average error measured at a single node in a 3×3 grid. The dashed red line is the target error, which is the length of the guard interval.

this issue, the technique proposed in this paper also minimises the on-time of the transceiver. The transceiver duty cycling may be dynamically updated to improve power efficiency. The guard interval for a given node should be no smaller than the maximum timing error observed at that node. Applying dynamic duty cycling to the 3×3 grid resulted in a guard interval that was 0.75% of the frame period after 40 iterations. Including the Tx window, this reduced the duty cycle from 50% to 6.5% according to Eq. 13.

$$\text{Duty Cycle} = \frac{2 \times \text{Guard Interval} + \text{Tx Window}}{\text{Frame Length}} \tag{13}$$

V. Conclusions

A novel time synchronization algorithm has been proposed, which has shown to be effective in all tested topologies and is robust to lost packets. Unlike previously reported works, it is able to synchronize distributed nodes with imprecise CMOS timers. The algorithm performs well in both a fully-connected and grid network and is even able to synchronize a unidirectional ring network. The algorithm is suitable for a distributed WSN with broadcasting, and is space, power and bandwidth efficient. An implementation of the algorithm has demonstrated its effectiveness in hardware, and requires only 1316 bytes in non-volatile memory. The hardware tests on a grid of eZ430-RF2500 sensor nodes prove that the algorithm is effective in a real network.

REFERENCES

[1] S. M. Lasassmeh and J. M. Conrad, "Time synchronization in wireless sensor networks: A survey," *Proc. of the IEEE SoutheastCon 2010 (SoutheastCon)*, pp. 242–245, Mar. 2010.

[2] C. S. Lam, "A review of the recent development of mems and crystal oscillators and their impacts on the frequency control products industry," in *Proc. - IEEE Ultrasonics Symposium*, pp. 694–704, 2008.

[3] B. Warneke, M. Last, B. Liebowitz, and K. Pister, "Smart Dust: communicating with a cubic-millimeter computer," *Computer*, vol. 34, no. 1, pp. 44–51, 2001.

[4] A. R. Swain and R. Hansdah, "A model for the classification and survey of clock synchronization protocols in WSNs," *Ad Hoc Networks*, vol. 27, pp. 219–241, 2015.

[5] I. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci, "Wireless sensor networks: a survey," *Computer Networks*, vol. 38, pp. 393–422, Mar. 2002.

[6] S. Kumar, "Sensor networks: Evolution, opportunities, and challenges," *Proc. of the IEEE*, vol. 91, pp. 1247–1256, Aug. 2003.

[7] 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 Transactions on Vehicular Technology*, vol. 59, no. 6, pp. 2963–2973, 2010.

[8] M. Maroti, B. Kusy, G. Simon, and A. Ledeczi, "The flooding time synchronization protocol," in *SenSys '04: Proc. of the 2nd international conference on Embedded networked sensor systems*, p. 39, 2004.

[9] O. Simeone, U. Spagnolini, and Y. Bar-Ness, "On pulse-coupled discrete-time phase locked loops for wireless networks," in *2007 IEEE 8th Workshop on Signal Processing Advances in Wireless Communications*, no. 1, pp. 1–5, IEEE, June 2007.

[10] O. Simeone and G. Scutari, "Pulse-coupled distributed plls in heterogeneous wireless networks," in *2007 Conference Record of the Forty-First Asilomar Conference on Signals, Systems and Computers*, pp. 1770–1775, Nov 2007.

[11] R. Beard and E. Atkins, "A survey of consensus problems in multi-agent coordination," in *Proc. of the 2005, American Control Conference, 2005.*, pp. 1859–1864, IEEE, 2005.

[12] A. Berger, M. Pichler, J. Klinglmayr, A. Potsch, and A. Springer, "Low-Complex Synchronization Algorithms for Embedded Wireless Sensor Networks," *IEEE Transactions on Instrumentation and Measurement*, pp. 1–1, 2014.

[13] P. Sommer and R. Wattenhofer, "Gradient Clock Synchronization in Wireless Sensor Networks," *Information Processing in Sensor Networks, 2009. IPSN 2009. International Conference on*, pp. 37 – 48, 2009.

[14] R. Carli, E. D'Elia, and S. Zampieri, "A PI controller based on asymmetric gossip communications for clocks synchronization in wireless sensors networks," in *IEEE Conference on Decision and Control and European Control Conference*, pp. 7512–7517, IEEE, Dec. 2011.

[15] G. Werner-Allen, G. Tewari, A. Patel, M. Welsh, and R. Nagpal, "Firefly-inspired sensor network synchronicity with realistic radio effects," in *Proc. of the 3rd international conference on Embedded networked sensor systems - SenSys '05*, (New York, New York, USA), p. 142, ACM Press, 2005.

[16] D. Stanislowski, X. Vilajosana, Q. Wang, T. Watteyne, and K. S. J. Pister, "Adaptive synchronization in IEEE802.15.4e networks," *IEEE Transactions on Industrial Informatics*, vol. 10, no. 1, pp. 795–802, 2014.

[17] X. Xu, Z. Xiong, X. Sheng, J. Wu, and X. Zhu, "A New Time Synchronization Method for Reducing Quantization Error Accumulation Over Real-Time Networks: Theory and Experiments," *Industrial Informatics, IEEE Transactions on*, vol. 9, no. 3, pp. 1659–1669, 2013.

[18] D. Christmann, R. Gotzhein, S. Siegmund, and F. Wirth, "Realization of Try-Once-Discard in Wireless Multihop Networks," *IEEE Transactions on Industrial Informatics*, vol. 10, pp. 17–26, feb 2014.

[19] M. K. Maggs, S. G. O'Keefe, and D. V. Thiel, "Consensus Clock Synchronization for Wireless Sensor Networks," *IEEE Sensors Journal*, vol. 12, pp. 2269–2277, June 2012.

[20] Texas Instruments, "MSP430F22x2 / MSP430F22x4 Mixed Signal Microcontroller (Rev. G) - SLAS504G," 2012.

[21] V. Pratap Singh, N. Chandrachoodan, and A. Prabhakar, "A Time Synchronized Wireless Sensor Tree Network using SimpliciTI," *International Journal of Computer and Communication Engineering*, vol. 2, no. 5, pp. 571–575, 2013.

[22] Texas Instruments, *MSP430F22x2 / MSP430F22x4 Mixed Signal Microcontroller*, July 2006. Rev. G.

James A. Boyle is working towards the M.Eng. (Hons.) degree in Electronic Engineering with Computer Systems at the University of Southampton, UK. He is the recipient of a scholarship from the UK Electronic Skills Foundation (UKESF) and in 2015 held a summer position in the system validation team at ARM Holdings, UK. His research interests include embedded systems, wireless sensor networks, communications systems and computer architecture.

Jeff S. Reeve (M'98–SM'05) received the Ph.D. degree in theoretical physics from the University of Alberta, Canada, in 1976. He has worked as a consultant in the communication and control group of Plessey, Auckland, New Zealand, and in the airspace division of Marconi Radar, Chelmsford, UK. He has over 100 publications in parallel computing, wireless sensor networks and computer architecture. His current research interests include wireless ranging and positioning, and distributed and parallel computing. He is currently a Senior Lecturer in the Department of Electronics and Computer Science at the University of Southampton, UK. He is a Chartered Physicist and a Member of the IoP.

Alex S. Weddell (GSM'06–M'10) received the Ph.D. degree in Electrical and Electronic Engineering from the University of Southampton, UK, in 2010. He has industrial, consultancy, and project experience in energy harvesting, instrumentation, and wireless sensor networks, and has authored over 30 papers in the area. He is now a Lecturer in the Department of Electronics and Computer Science at the University of Southampton, UK. He has special interests in the areas of energy-aware systems and energy harvesting.