FLOPSYNC-QACS: Quantization-Aware Clock Synchronization for Wireless Sensor Networks

Federico Terraneo, Alessandro Vittorio Papadopoulos, Alberto Leva, Maria Prandini

***Abstract*—Distributed real-time systems often relies on clock synchronization. However, the achievement of precise synchronization in Wireless Sensor Networks (WSNs) is hampered by competing design challenges, which finally causes many WSN hardware platforms to rely on low frequency clock crystal for local timebase provision. Although this solution is inexpensive and with a remarkably low energy consumption, it limits the resolution at which time can be measured. The FLOPSYNC synchronization scheme was then introduced to compensate for possible quartz crystal imperfections. The main limitation of FLOPSYNC is that it does not account for the effects of quantization. In this paper we propose a switched control variant of the base FLOPSYNC scheme to address quantization explicitly in the compensator design, providing clock synchronization in cost-sensitive WSN node platforms with a minimal additional overhead. Experimental evidence is given that the approach reaches a synchronization error of at most 1 clock tick in a real WSN.**

***Keywords*-Quantized control; Clock synchronization; Switched control; Wireless Sensor Network.**

I. Introduction

The increasing pace of smart applications and devices to handle complex issues is nowadays growing alongside with the demand for connectivity. Recent studies estimate the Internet of Things (IoT) to count 6.4 billion devices (excluding smartphones, tablets and computers), with a forecast of up to 21 billion by 2020 [1]. Communications are crucial in this *arena*, and in particular, Wireless Sensor Networks (WSNs) proliferate. Tiny, inexpensive, low-power WSN nodes will thus become ubiquitous, as an enabling technology for IoT [2], [3].

For the correct operation of WSNs, a major challenge is accurate time synchronization [4], [5]. This is required to ensure reliable communication links, but also for location/proximity estimation [6], energy efficiency [7], [8], mobility [9], and wherever WSN nodes coordination – possibly with real-time tasks – is required. Moreover, since most WSN nodes operate on battery and synchronization must be guaranteed continuously, energy-efficient solutions are in order [10].

Requirements are not equally tight in any application, however. To reduce costs, when high timing precision is not needed, low-resolution clocks can be a viable choice [11], [12]. In such cases, to reduce the synchronization quality detriment, solutions able to push precision to the limits are needed.

F. Terraneo, A. Leva, and M. Prandini are with the Dipartimento di Elettronica, Informazione e Bioingegneria, Politecnico di Milano, Milano, Italy (e-mail: {federico.terraneo, alberto.leva, maria.prandini}@polimi.it)

A.V. Papadopoulos is with School of Innovation, Design and Engineering (IDT), Mälardalen University, Västerås, Sweden (e-mail: alessandro.papadopoulos@mdh.se)

This work was partially supported by the European Commission under the project UnCoVerCPS with grant number 643921, and by the Swedish Foundation for Strategic Research under the project "Future factories in the cloud (FiC)" with grant number GMT14-0032.

We here present a synchronization mechanism that minimizes the effect of quantization on the synchronization error, with a minimal overhead. The work is cast in the framework of *multi-hop master-slave clock synchronization*, i.e., when the WSN is composed by a master that holds the reference clock, and of a number of slaves that must synchronize their clocks to the master.

A preliminary conference version of this paper has been presented in [13]. The present manuscript extends [13] in several directions. More precisely, a characterization of the behavior of the proposed synchronization mechanism in terms of invariant set is given together with an intuitive explanation and some illustrative plots. Implementation of the scheme on typical WSN hardware is detailed. Finally, a more extensive simulation study is provided.

II. The synchronization problem

In master-slave clock synchronization, timing information is disseminated to the WSN by one master node. In a single-hop WSN this happens by direct communication. In multi-hop WSNs, flooding schemes [14], [15] allow for the said dissemination irrespectively of the network topology, and within a very small amount of time. The disseminated packets may contain a timestamp of the master clock, or timing information can be implicit if packets are flooded periodically over a contention-delay-free MAC [16].

Upon receiving master time information, the slaves can correct their local time, and in the WSN literature this action is called *clock synchronization*. However, the slaves also need to maintain accordance with the master time in between two subsequent arrivals of master information; this is called *skew compensation* [17]–[19], as NTP defines the skew as the derivative of the synchronization error with time. The skew can however vary with time as well, for example owing to temperature variations, and its derivative with time is named drift. It is possible to compensate for drift, see e.g. [16], [20], and also for the packets' radio propagation delay [21], [22] if ultra-high precision synchronization is needed (the delay is about 1μs each 300m traveled by the signal).

Master-slave schemes may have many different features, but invariantly need to timestamp incoming packets with the local clock. Timekeeping in WSN nodes is done by a dedicated hardware timer/counter aboard the node processor, that is read to know the time, and allows to set interrupts for

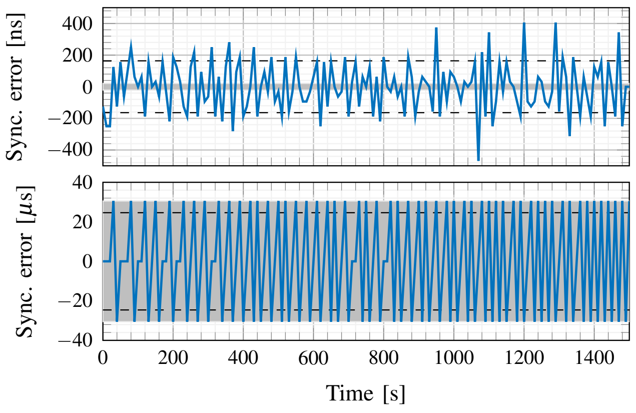

Figure 1: Experimental result showing the synchronization error using the high frequency timer (top graph) and low frequency timer (bottom graph) on the WandStem node. Note the different time units on the vertical axis.

generating events. Incoming packet timestamping can thus be done by reading the hardware counter in the packet reception interrupt handler [14], or by using a hardware input capture module [15], [16] that takes a counter snapshot upon reception. In any cases, the finite frequency of the local hardware counter inevitably introduces quantization in packet timestamping, and the entity of this quantization increases as the counter frequency is decreased.

To limit energy consumption by the counter, its frequency has to be limited with respect to typical CPU clocks—consider that the CPU can be set to "deep sleep" to save power, but the timekeeping timer cannot be stopped to not lose the notion of time. Briefly, in WSN nodes it is common to have the CPU clocked at several megahertz, while the timekeeping timer runs at just 32768Hz [11], a frequency for which inexpensive and ultra low-power quartz crystals are available.

To tackle the tradeoff between timing resolution and consumption, the Virtual High-resolution Time (VHT) algorithm [7] was introduced. This synchronizes a high-frequency timer, turned off in deep sleep, with a low frequency timer always active. This solution has been used to achieve high synchronization precision and ultra-low consumption [16], but requires a hardware support that is not common in WSN nodes.

The main requirements for VHT are a high-frequency timer clocked with a stable oscillator, and hardware support for timestamping an edge of the low-frequency clock with that high-frequency timer. In [7] this problem is solved in hardware, and a VHDL realization is also proposed. Recently, nodes appeared with this support [23], but widely employed nodes [24], such as the TelosB [11], are not VHT-capable.

This is far more than a legacy issue. Cost-sensitive IoT applications may not justify the additional cost of VHT support. Thus, obtaining accurate synchronization using low-frequency clock timers is a relevant research topic, with the potential to enable low-cost, real-time capable WSN platforms.

To evidence that synchronization with a high- and a low-frequency clock are two different *scenarii*, we present a test in which two nodes are synchronized with the FLOPSYNC-2 [16] scheme on the WandStem [23] WSN node platform. This platform has VHT support, which made it possible to compare the synchronization quality, in the two cases with and without VHT, on the same hardware. The tests were done with a 10s synchronization period, and the reported error samples are taken at the end of each period.

The top plot in Figure 1 shows the results with a high-frequency timer, while the bottom one shows the low-frequency timer case. The high-frequency timer has a resolution of 20.8ns (gray area), but the standard deviation of the synchronization error is 164ns (dashed lines), a significantly higher value. The reason is that at high frequencies, the oscillator phase noise jitter and the packet transmission jitter from the radio transceiver, are greater than the quantization-induced error [16]. On the contrary, with the low-frequency timer, the error standard deviation is 24.6μs (dashed lines), i.e., lower than the 30.5μs timer resolution (gray area). Interestingly, the error shows a regular pattern with only three values – 0 and ± 1 timer tick – evidencing that the quantization-induced error magnitude is greater than the noise sources.

The clock synchronization algorithm we propose herein, includes a switching control scheme that minimizes the effect of quantization on the synchronization error. The proposed solution is therefore applicable and useful in all the cases where quantization is the major source of error.

III. The FLOPSYNC synchronization scheme

In this section we formalize the problem, point out the sources of quantization, and review the original FLOPSYNC synchronization scheme as proposed in [18].

A. Problem formalization

Time synchronization in a distributed system is a well known and studied problem in computer science [25]–[28], and has recently gained attention in the control community as well [8], [29]. We here limit the scope to the master-slave case in which the master floods the WSN with synchronization packets at a fixed period T, constant and known network-wide. Furthermore, we assume the presence of a fast flooding scheme like Glossy [15], so that medium access contention introduces no uncertainty in the transmission time. Finally, synchronization is achieved by individual controllers aboard each slave node, that only receive packets from the master.

The synchronization error at the k-th synchronization time kT, $k \in \mathbb{N}$, is

$$e(k) := t(k) - \hat{t}(k),$$

where $t(k)$ denotes the master clock at the k-th synchronization, and $\hat{t}(k)$ the slave estimate of the master clock. As the error accumulates over time, during each time interval $[kT, (k+1)T]$ the synchronization error dynamics is ruled by

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

where $d(k)$ is a disturbance that accounts for different phenomena, briefly discussed later on, and characterized as

$$d(k) = -\int_{kT}^{(k+1)T} \frac{\delta_f(\tau)}{f_o} d\tau, \tag{2}$$

where f_o is the nominal frequency of the slave clock, and $\delta_f(t)$ the (continuous-time) variation of that frequency caused by manufacturing tolerances, aging, thermal stress, and short-term jitter. The minus sign in (2) is because $\delta_f > 0$ makes the local clock advance, while (1) contains $d(k)$ with the plus sign for convenience.

Notice that all the uncertainty is confined in the way the disturbance $d(k)$ is generated. Based on (1), a controller can be designed to reject $d(k)$ with a very little computational overhead, see e.g., [18].

The phenomena in d are briefly listed below, and can be counteracted by considering their different time scales.

- Tolerances due to imperfections in the quartz crystals manufacturing result in a *constant frequency error* $\overline{\delta_f}$.
- Aging is a phenomenon that acts on a time scale of days, while reasonable values for the synchronization period T are seconds or minutes, hence this can be safely thought of as a constant disturbance contribution as well, and eliminated at steady state – like the effect of imperfections – by integral control.
- The temperature dependence of crystals is a major source of variable disturbance [7]. However, in virtually any operating condition, a WSN undergoes either abrupt but sporadic thermal stress episodes like shade-sunlight transitions, or environmental variations that are slow when compared to the thermal dynamics of typical nodes. The controller of [18] can be extended to compensate for abrupt thermal variations [16], but in between such events, this disturbance contribution can be assumed constant as well.
- Short-term jitter acts on the time scale of electronic noise, hence it is too fast to compensate, and provides the ultimate bound for the achievable synchronization quality. However, as anticipated, in this work we are addressing the case where quantization is a greater source of error than jitter.

The above disturbance characterization allows to focus on optimizing the controller for the constant d case, although the proposed controller will obviously still be able to cope with (reasonably) variable disturbances.

B. The FLOPSYNC synchronization scheme with quantizers

In [18], the FLOPSYNC scheme was proposed. FLOPSYNC introduces a corrective action u to compensate for the sources of the synchronization error e:

$$e(k+1) = e(k) + u(k) + d(k), \tag{3}$$

with u is computed with a Proportional Integral (PI) controller.

The control scheme performance is limited by the presence of a quantization on both the synchronization error e (controlled variable that should be driven to zero) and the output of the clock correction algorithm u (corrective action that should drive the synchronization error to zero).

As for the former quantization, the hardware counter is incremented on the active edge of its clock, while asynchronous events – such as packet arrivals – can occur at any time between two edges. The reported timestamp will thus be the

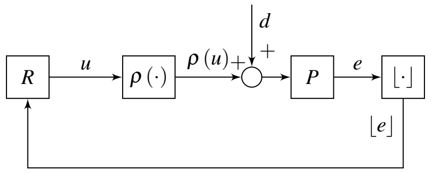

Figure 2: The FLOPSYNC synchronization scheme with quantizers.

value of the counter as last incremented by the edge preceding the event. Thus, hardware timestamping works like the $floor$ operator on the synchronization error e.

As for the latter quantization, it occurs when the output u of the clock correction algorithm, computed using floating point or fixed point numbers, is converted back to the tick resolution. This is done in software, however, hence one can choose the quantization function—for example, the *round* operator.

Summarizing, we have two sources of quantization. One is physically constrained to act as the $floor$ operator, acts on the synchronization error e, and depends on the clock resolution. The other is software-configurable, acts on the corrective action u, and depends on the arithmetic precision of the used architecture.

Without loss of generality, we conduct the following treatise as if the resolution of the clock were the unity. Re-scaling for a different quantum is just the same as changing the time unit. As for the control resolution, it is configurable and here is set equal to the clock resolution. This design choice will be motivated in Section IV (see Remark 1).

At this point, we need to define the required operators. Given a real number z, we denote by $\operatorname{sign}(z)$ the sign function, by $\lfloor z\rfloor$ the $floor$ operator, and by $\rho(z)$ the *round* operator, with $\rho(0.5) = 1$, and $\rho(-0.5) = -1$. We also define the *rounding error* for the real number z as $\Delta_z := z - \rho(z)$. Notice that the rounding error is always bounded as $|\Delta_z| \leq \frac{1}{2}$.

Coming back to FLOPSYNC, since the control action is quantized, (3) becomes

$$e(k+1) = e(k) + \rho(u(k)) + d(k), \tag{4}$$

where u is determined from the quantized measurements of the synchronization error $\lfloor e\rfloor$, by the discrete-time PI controller

$$u(k+1) = u(k) + \lfloor e(k)\rfloor - \alpha \lfloor e(k+1)\rfloor \tag{5}$$

where α is the only design parameter.

Figure 2 shows the FLOPSYNC control scheme, where P is the process (4), and R the controller (5). Substituting (5) into (4) we get

$$\begin{aligned} e(k+1) &= e(k) + d(k) \\ &\quad + \rho\left(u(k-1) + \lfloor e(k-1)\rfloor - \alpha\lfloor e(k)\rfloor\right) \end{aligned}$$

In the original formulation of FLOPSYNC in [18] both quantizers were just neglected in the controller design, and relegated to implementation-related accidents. With no quantization in place, by replacing the expression for $u(k)$ in (4) with that given by (5) with k in place of $k+1$ and then using

$e(k-1)+u(k-1)=-e(k)-d(k-1)$ (derived from (4)), we get:

$$\begin{aligned} e(k+1) &= (1-\alpha)e(k)+e(k-1)+u(k-1)+d(k) \\ &= (2-\alpha)e(k)+d(k)-d(k-1) \end{aligned}$$

which corresponds to an asymptotically stable system if $1 < \alpha < 3$. For a constant disturbance $d(k) = d(k-1) = \overline{d}$, the scheme makes the synchronization error converge to zero, with a rate that depends on α.

When quantizers come into play, the synchronization error still (ideally) converges to zero, but quite intuitively, it is not possible to discriminate from zero errors that are below the clock resolution. Moreover, d is integrated over time according to (4). The integrated residual disturbance is not detectable on the quantized output $\lfloor e \rfloor$ until it exceeds the clock resolution. This makes the controller react whenever the quantization of the integrated residual disturbance switches to 1 or -1. As a result, the controlled system enters a limit cycle of amplitude 2. An example of this effect is illustrated in Figure 3, with $\alpha = 1.4$, $d(k) = \overline{d} = \sqrt{3}$, and the control system initialized with $e(0) = 2$, $u(0) = 0$.

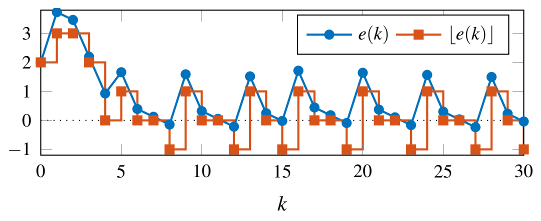

Figure 3: The impact of quantization on the synchronization error in the FLOPSYNC scheme.

This paper proposes a switched control scheme that reduces the just evidenced effect of quantization, steering the system to a limit cycle of an amplitude that is half of that obtained with the bare FLOPSYNC control scheme as proposed in [18]. The solution presented herein has the additional advantage of sticking to simple controllers, easy to implement in an embedded device, with very low computational and memory overhead.

IV. The proposed FLOPSYNC-QACS synchronization scheme

In this section we describe FLOPSYNC-QACS, the variant to FLOPSYNC that we propose to improve its performance in the presence of quantization.

The FLOPSYNC-QACS controller is composed of a linear and a switched component. The linear part is described by

$$\tilde{u}(k+1) = \lfloor e(k) \rfloor - \alpha \lfloor e(k+1) \rfloor \tag{6}$$

and generates signal $\tilde{u}$, which is fed into the switched part that computes the control input u as

$$u(k+1) = \begin{cases} u(k) + \tilde{u}(k+1), & \text{if } \lfloor e(k+1) \rfloor \neq 0 \\ \rho\left(u(k)\right) + \tilde{u}(k+1), & \text{if } \lfloor e(k+1) \rfloor = 0, \end{cases} \tag{7}$$

depending on the quantized synchronization error measurement $\lfloor e \rfloor$. The resulting switched control scheme is represented in Figure 4, where $\tilde{R}$ is the linear component (6), P the synchronization error dynamics (4), and z^{-1} the unitary delay operator.

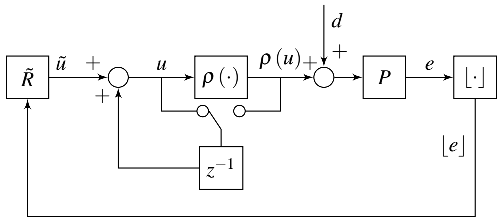

Figure 4: The FLOPSYNC-QACS synchronization scheme.

The switched control system dynamics is characterized in terms of the evolution in time of the variables e and u as

$$\begin{aligned} &\text{if } \lfloor e(k+1) \rfloor = \lfloor e(k) + \rho\left(u(k)\right) + d(k) \rfloor = 0 \\ &\quad \begin{cases} e(k+1) = e(k) + \rho\left(u(k)\right) + d(k) \\ u(k+1) = \rho\left(u(k)\right) + \lfloor e(k) \rfloor \end{cases} \\ &\text{else} \\ &\quad \begin{cases} e(k+1) = e(k) + \rho\left(u(k)\right) + d(k) \\ u(k+1) = u(k) + \lfloor e(k) \rfloor - \alpha \lfloor e(k) + \rho\left(u(k)\right) + d(k) \rfloor, \end{cases} \end{aligned} \tag{8}$$

which are obtained from (4) and (7) with $\tilde{u}$ replaced by its expression in (6). Apparently, the computational complexity of the proposed solution is limited to measuring $\lfloor e(k) + \rho\left(u(k)\right) + d(k) \rfloor = \lfloor e(k+1) \rfloor$ and to computing the control action u as per the applicable alternative in (8), based on the measured quantized value of $\lfloor e(k+1) \rfloor$.

Let the disturbance be constant and equal to $d(k) = \overline{d}$, $k \geq 0$. We define the disturbance rounding error

$$\Delta_d = \overline{d} - \rho\left(\overline{d}\right), \tag{9}$$

and the residual control input signal

$$\overline{u}(k) = u(k) + \rho\left(\overline{d}\right), \tag{10}$$

which is zero when the control input perfectly counteracts the quantized value of the disturbance.

The proposed FLOPSYNC-QACS control scheme is able to reduce the amplitude of the limit cycle for the quantized output $\lfloor e \rfloor$ from 2 to 1. This is actually evident in Figure 5, where an example of possible evolution of the system is shown, for $\alpha = 11/8$, when $\Delta_d = -0.2$ and the switched control system is initialized at $e(0) = 0$, and $\overline{u}(0) = 2$. The left column present the results obtained with FLOPSYNC, while the right column presents the results obtained with FLOPSYNC-QACS. The top graphs in Figure 5 show the phase plot of the system, with the green square indicating the initial condition. The central and bottom graphs represent the time evolution of the state variables e and $\overline{u}$ and their quantized version.

In the case of FLOPSYNC-QACS, after the state enters the red area in the top plot, it ends up in one time step (and keeps evolving forever) in an invariant set where the quantized variables $\lfloor e(k) \rfloor$ and $\rho\left(\overline{u}(k)\right)$ have an excursion of amplitude equal to 1. This same behavior can be observed for other values of α, which are given in Theorem 4.1 together with the characterization of the red area (equation (11)).

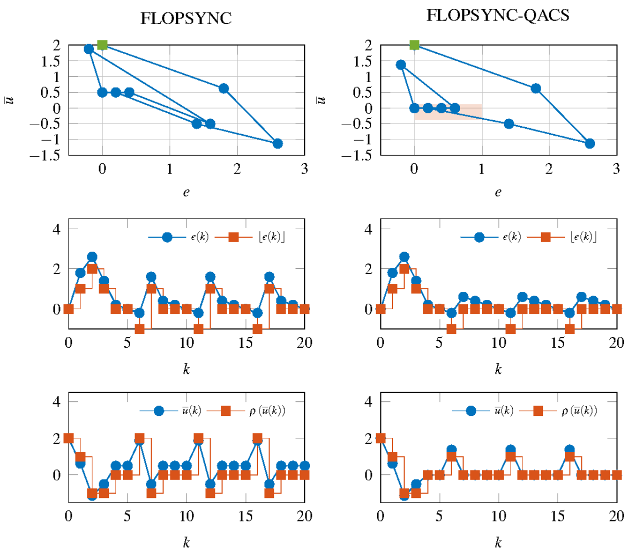

Figure 5: Comparison between FLOPSYNC (left column) and FLOPSYNC-QACS (right column). The top graph shows the phase plot in the state variables e and $\overline{u}$. The lower plots show the time evolution of the state variables and their quantized versions.

Theorem 4.1: Let the design parameter α be chosen within $(1, \frac{3}{2})$. Suppose that at some time h the state variables e and u of system (8) satisfy:

$$\begin{cases} 0 < e(h) < 1 \\ 1 \leq \alpha - \overline{u}(h)\,\mathrm{sign}\left(\Delta_d\right) < \frac{3}{2} \\ -\frac{1}{2} < \overline{u}(h) < \frac{1}{2}, \end{cases} \tag{11}$$

where $\overline{u}(h) = u(h) + \rho\left(\overline{d}\right)$. Then,

$$\left(\lfloor e(k) \rceil, \rho\left(\overline{u}(k)\right)\right) \in \mathscr{I} = \left\{(0,0), \left(\mathrm{sign}\left(\Delta_d\right), -\mathrm{sign}\left(\Delta_d\right)\right)\right\},$$

for all $k > h$. Moreover, $\mathscr{I}$ is the smallest invariant set for $\left(\lfloor e \rceil, \rho\left(\overline{u}\right)\right)$, when system (8) evolves starting from (11).

The proof of Theorem 4.1 (here omitted) can be obtained by adapting the proof in [30] to the case when $\rho(e)$ is replaced with $\lfloor e \rceil$.

In [30], we performed a numerical reachability analysis study and showed that if $5/4 < \alpha < 3/2$ and $|\Delta_d| < 0.5$, then, the invariant set in Theorem 4.1 is globally attractive, i.e., it will be eventually reached from any initial condition. For $|\Delta_d| = 0.5$, global attractiveness does not hold true and the system may end up in an invariant set where the amplitude of the excursion for the quantized state is 2, while for $|\Delta_d| \neq 0.5$ and $1 < \alpha \leq 5/4$ only invariant sets where the excursion amplitude is 1 appear.

We can then conclude that the proposed switched scheme performs better than the bare FLOPSYNC control scheme proposed in [18] for almost all Δ_d values.

Remark 1 (control resolution): Note that in both modes of operation in (8), the synchronization error e is obtained by integrating the signal $\rho(u) + \overline{d}$. Ideally, the control action u should be set so as to compensate exactly for the disturbance but this is hampered by the presence of the round quantizer. The process then integrates the residual disturbance $\rho(u) + \overline{d}$ and the controller realizes that the disturbance is not compensated exactly only when the process output e reaches the quantization threshold, since only at that point the measured synchronization error $\lfloor e \rceil$ will jump to either 1 or -1. $\lfloor e \rceil$ is then brought back to zero, and the same kind of behavior is observed over and over, thus resulting in a cycle for $\lfloor e \rceil$ with a unitary amplitude. Improving the control resolution would not have any impact on the control scheme performance in terms of amplitude of this cycle: The time needed for the process output e to reach the quantization threshold will be larger, but still a cycle of amplitude 1 for $\lfloor e \rceil$ will be observed.

V. Implementation of FLOPSYNC-QACS

In this section we describe how FLOPSYNC-QACS can be efficiently implemented on typical WSN hardware. To do so, we first have to briefly review the FLOPSYNC synchronization scheme, which is composed of two main parts.

The first part is implemented at the MAC (Medium Access Control) level. Its task is to periodically take over the ordinary MAC used for the applications, and switch the radio control to the flooding scheme used to receive and rebroadcast the synchronization packet. The flooding scheme we adopt is Glossy [15], which was extensively proven capable of making synchronization packets reach all the nodes of a realistically-sized, multi-hop WSN in a practically negligible time. From the timestamped arrival time of the synchronization packet, the clock synchronization error is measured, and the controller is run to compute the correction u. This quantity is used both to decide the expected arrival time for the next synchronization packet, and as the input to the second part of FLOPSYNC.

This second part is implemented at the operating system level. Its role is to employ the correction provided by the controller to offer timestamping and clock services to the running applications. FLOPSYNC-QACS does not introduce changes to this part compared to FLOPSYNC, hence the matter will not be explained further.

A simplified version of the aforementioned first part is shown in Listing 1. The code paths to handle packet losses have been omitted, to better focus on the control law. The implementation of the said law is a task that a short time (called the *receiver window*) before the expected arrival time for a synchronization packet, disables the ordinary MAC, sets the radio to receive mode, and timestamps the packet et its arrival. The packet is then rebroadcast, as required by Glossy, and after that the synchronization error is computed as the difference between the expected and the actual arrival time. Due to the limited hardware timer resolution the actual arrival time has a finite resolution as well, hence computing the error introduces the first quantization in the control loop.

The control law is easy to implement using fixed point arithmetic, which is a notable advantage, as most WSN nodes lack hardware floating point support. The computed correction u, still in fixed point form, is then converted to an integer number using the *rounded_division* function, which is necessary as the C/C++ division operator does not perform arithmetic rounding.

```
int uo = 0;             // past control value
int eo = 0;             // past error
int w = wMax;           // receiver window
long long eat = period; // expected arrival time

while (1) {             // each synchronisation k
   disableMacLayer();
   int timeout = (2*w) + packetTime;
   waitForSyncPacket(timeout);
   // get actual arrival time, first quantization (floor)
   long long at = rtc.getValue();
   rebroadcastWithGlossy();
   // compute error
   int e = eat - at;
   // u(k)=u(k-1)+11/8*e(k)-e(k-1)
   // fixed point implementation with 3 bit fractional part
   int u = uo + 11*e - 8*eo;
   // from fixed point to int, second quantization (round)
   int uquant = rounded_division(u,8);
   // switched part, the core of FLOPSYNC-QACS
   if(e==0) {
      uo = 8 * uquant;
   } else {
      uo = u;
   }
   eo = e;
   w = update_receiver_window(e);
   // updating expected arrival time
   eat += period + uquant;
   enableMacLayer();
   rtc.sleepUntil(eat - w);
}
```

Listing 1: FLOPSYNC-QACS controller.

Computing the actually applied control therefore introduces the second quantization.

The switched part of the controller is implemented as a single *if* statement, selecting either the quantized or fixed point correction, depending on the error value.

The last part of the task computes the receiver window based on the error value – the interested reader can find information about this step in [16], suffice here to say that this is motivated by minimizing the radio ON-time for energy efficiency – and then re-enables the ordinary MAC layer. This done, the FLOPSYNC-QACS task is suspended till the next synchronization.

VI. Experimental and simulation results

The performance gains provided by FLOPSYNC-QACS have been tested both in a real WSN node and in simulation. In Section VI-A we show a representative example of the performed experimental tests, to testify the correct operation of the technique in practice. In Section VI-B we then summarize the results of a simulation campaign, in which the operation of FLOPSYNC with and without the proposed switched control scheme is compared. The use of simulation is necessary for this purpose, as it is the only way to compare the two algorithms in the exact same conditions.

A. An experimental test

Experimental testing aims at assessing the performance improvement yielded by the proposed FLOPSYNC extension in a real-world setting. The main point is that in the theoretical analysis the disturbance has been considered constant, whereas real disturbances are actually varying. However, if their time-variability occurs at a timescale that is much longer than the

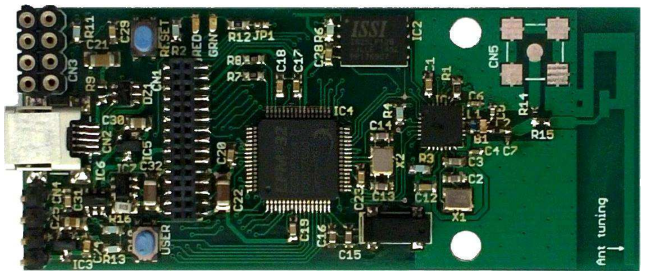

Figure 6: One of the WandStem WSN nodes used for testing.

control sampling time, then, results of our analysis should still be valid. The test presented in this section shows that this is indeed the case.

We implemented the base FLOPSYNC and FLOPSYNC-QACS schemes on a WSN composed of WandStem [23] nodes, that employ ARM Cortex-M3 microcontrollers running at 48MHz, and CC2520 radio transceivers operating in the 2.4GHz band. One of these nodes is shown in Figure 6. The control algorithms are implemented in C++ in the Miosix [31] microcontroller operating system. The synchronization period T is 60 seconds. The hardware timers of the nodes have a measurement and actuation resolution (also called *tick*) of 30.5μs, which is the source of quantization, and it is normalized to 1. For our implementation, we set $\alpha = 11/8$ as in Listing 1, since this value preserves stability of the closed-loop linear dynamics and satisfies the condition on α required for Theorem 4.1 to hold.

In the test, three nodes are used. One plays the role of the master, broadcasting synchronization packets. Out of the other two, one runs the bare FLOPSYNC scheme, and the other the FLOPSYNC-QACS switched variant of the scheme. The nodes are placed in an office environment, and, hence, they are exposed to radio interference from local wireless networks and to temperature variations like those encountered in a typical indoor setting with standard climatization. In order to show the long-term behavior of the system in the face of slowly varying disturbances, the experiment was set to last 20 hours. Results are not specific to the experiment duration. We chose it so as to cover a time window large enough for the WSN to experience a (slow) variation in the disturbance that makes it switch between the two invariance sets in Theorem 4.1 that are associated to the two signs of the disturbance rounding error.

Figure 7 shows both the quantized synchronization error in ticks (top plots), and the quantized control variable (bottom plots) for FLOPSYNC (left column) and its switched version (right column). The horizontal axes report the experiment time in hours. The $[-1,1]$ synchronization error range is highlighted in both top plots with a gray area. The transient leading the error to approach the gray area can be estimated from the data to last approximately 6 periods (i.e., minutes) in both the bare and the FLOPSYNC-QACS case. Notice that in the case of FLOPSYNC, the gray area is practically covered by the quantized synchronization error trajectory. This is because the quantized synchronization error oscillates within the set $\{-1,0,1\}$ with an excursion of amplitude 2.

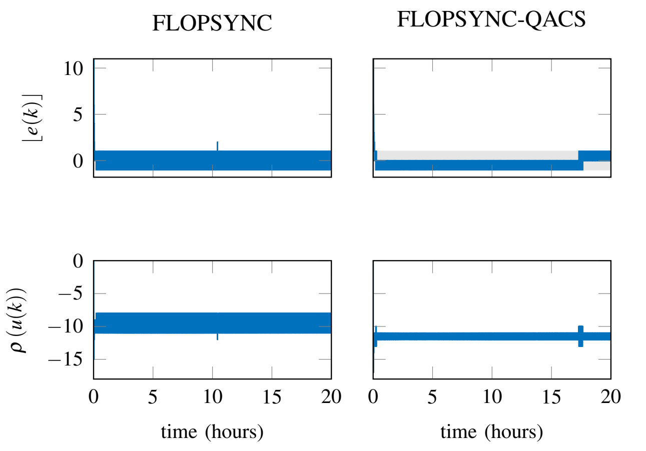

Figure 7: Experimental results comparing bare FLOPSYNC with FLOPSYNC-QACS. Quantized synchronization error $\lfloor e \rceil$ (top plots), and quantized control action $\rho(u)$ (bottom plots).

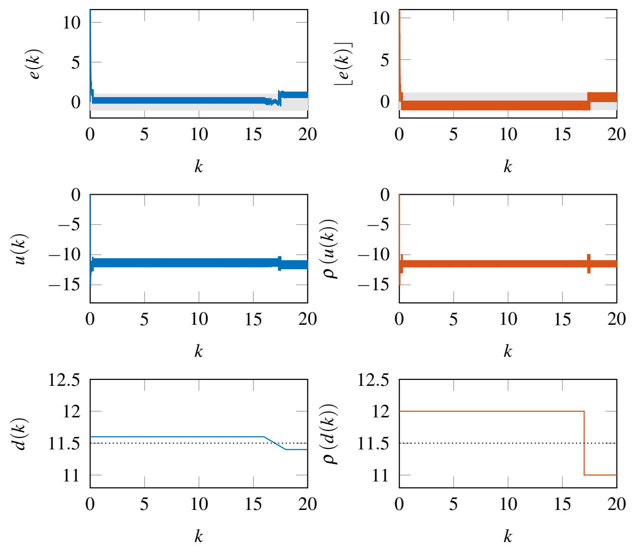

Figure 8: Replication of the experimental results with simulated dynamics.

In the case of FLOPSYNC-QACS, the quantized synchronization error first switches within the set $\{-1,0\}$, then, after a brief transient, it switches within $\{0,1\}$. For practically the whole experiment, the quantized synchronization error has an excursion of amplitude 1. More in details, the error lies in either $\{-1,0\}$ or $\{0,1\}$ for 97% of the time.

We compare the two results by computing the Root Mean Square (RMS) performance index of the quantized synchronization error, that is defined as:

$$RMS(\lfloor e \rceil) = \sqrt{\frac{1}{H}\sum_{i=0}^{H-1} \lfloor e(i) \rceil^2}$$

where H is the number of samples collected in the experiment. The RMS computed in the case of bare FLOPSYNC is 0.946, while the RMS computed for FLOPSYNC-QACS is 0.740, i.e., about 22% less than with bare FLOPSYNC. This means that in FLOPSYNC-QACS the quantized synchronization error $\lfloor e \rceil$ is equal to zero more times than in FLOPSYNC.

Note that the values taken by $\rho(u)$ in the two nodes are different, possibly owing to the manufacturing tolerance of each clock crystal that results each in an unique offset (i.e., a different disturbance value) to compensate. This is, however, not relevant for the purpose of the test.

In summary, we can conclude that the proposed control scheme results in a lower RMS error magnitude in a practical setting, where the disturbance is not rigorously constant.

Relying on the theoretical analysis presented in this paper, it is possible to analyze a bit more in detail the experimental results. In particular, focusing on the FLOPSYNC-QACS case, we see that, after the initial settling, the quantized synchronization error enters the invariant set of Theorem 4.1 and is kept in the set $\{-1,0\}$ for about 17 hours. During this time span, we can guess that the rounding error of the disturbance Δ_d is negative and that it does not change sign—even though it might have varied. In fact, the evolution of $\rho(e)$ is compatible with the invariant set associated with $\Delta_d < 0$, yielding $\rho(\overline{u}) \in \{0,1\}$. Within the same time span, it is possible to observe that $\rho(u)$ switches in the set $\{-12,-11\}$, and since $\rho(u) = -\rho(d) + \rho(\overline{u})$ by equation (10), we can conclude that the quantized value of the disturbance is $\rho(d) = 12$.

After about 17 hours from the beginning of the experiment, the evolution of the quantized synchronization error changes, and it settles to the invariant set of Theorem 4.1 associated with $\Delta_d > 0$, yielding $\rho(\overline{u}) \in \{-1,0\}$. We can therefore conclude that Δ_d changed its sign. Since the value of $\rho(u)$ still switches in the set $\{-12,-11\}$, but now $\rho(\overline{u}) \in \{-1,0\}$, we can conclude that the quantized value of the disturbance is now $\rho(d) = 11$. This entails that the disturbance decreased crossing 11.5.

In order to better investigate what caused the transition between the two invariant sets in the experimental results, we performed a simulation study trying to replicate the same behavior with a slowly changing disturbance. The results of the simulation are shown in Figure 8, where on the left column we reported the synchronization error e, the control signal u and the disturbance d, while on the right column we reported their quantized versions. We initialized the system at $e(0) = 0$, $u(0) = 0$, and we set $\alpha = 11/8$, as in the experimental setting.

We selected a disturbance that starts as a constant $d = \overline{d}_1 = 11.6$, i.e., $\Delta_d = -0.4 < 0$. Then from time $k = 960$ (16 hours) the disturbance slowly decreases linearly up to the value $d = \overline{d}_2 = 11.4$, i.e., $\Delta_d = 0.4 > 0$. Finally, the disturbance keeps constant and equal to $\overline{d}_2$, from time $k = 1080$ (18 hours). Apparently, the abrupt change of sign of Δ_d when the disturbance crosses the threshold of 11.5 at time $k = 1020$ (17 hours) causes a transient, that is reflected in the quantized version only at time $k = 1038$, where the quantized synchronization error oscillates between $[-1,1]$ and correspondingly the quantized control input oscillates between $[-13,-10]$. This is exactly the same behavior that can be observed in the experimental data of Figure 7.

We can thus conclude that the behavior that appeared in the experimental results may have been caused by a disturbance similar to the one presented in the left bottom graph of

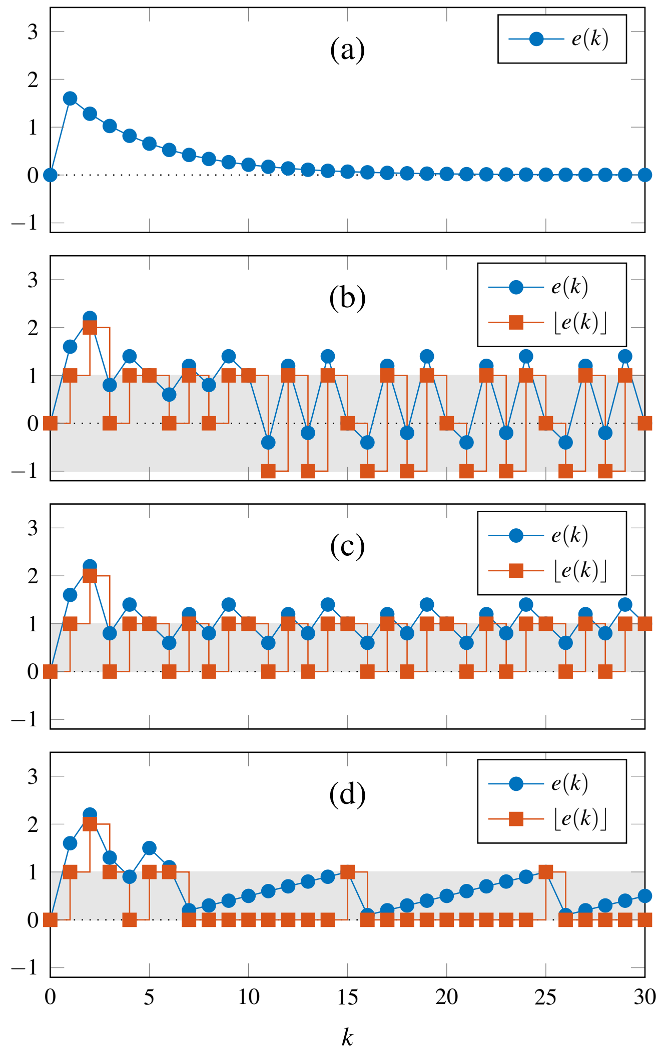

Figure 9: Quantized (red line with squares) and non quantized (blue line with circles) synchronization error in a simulated experiment obtained by adopting (a) FLOPSYNC without quantizers, (b) bare FLOPSYNC, (c) FLOPSYNC-QACS, and (d) FLOPSYNC-QACS with a higher control resolution.

Figure 8.

B. A comparative simulation campaign

We first present some simulation results comparing the cases when no quantization is present in the control scheme, when quantization is present and either FLOPSYNC or its switched extension FLOPSYNC-QACS is implemented. Notice that in the absence of quantization FLOPSYNC and FLOPSYNC-QACS coincide. The three plots on the top of Figure 9 represent the simulation runs for the three cases for a finite horizon of 30 synchronization periods. The bottom plot shows the performance of FLOPSYNC-QACS when a control resolution of 0.5 is adopted instead of 1 (see Remark 1), e.g., 0.6 is approximated as 0.5 instead of 1. In all plots the error is normalized, i.e., a unit clock resolution is assumed. The value used for α is 1.2, and $\Delta_d = 1.6$, while the system state is initialized at $e(0) = 0$, and $u(0) = 0$.

While in the absence of quantization the synchronization error converges to 0 with the designed controller, when quantization is in place it is not possible anymore to guaranteeing convergence to zero. In the case of bare FLOPSYNC, the synchronization error oscillates in the area $[-1, 1]$, while in the case of its switched extension, it ends up oscillating in the region $[0, 1]$ according to Theorem 4.1. As pointed out in Remark 1, the oscillation extent does not improve if a higher control resolution is adopted and only the frequency of the oscillations is affected.

The results presented next refer to a simulation campaign aimed at investigating the effect of the disturbance magnitude on the synchronization quality, with and without the proposed FLOPSYNC extension.

The campaign was carried out by choosing the values of $\overline{d}$ reported in Table I. For each value of $\overline{d}$, the two synchronization schemes, one with FLOPSYNC and the other with FLOPSYNC-QACS, were initialized to $e(0) = 0$ and $u(0) = 0$, and then subjected to a constant disturbance of the selected amplitude. Data were collected over a time horizon of $H = 1000$ synchronization periods. Table I summarizes the results: the proposed extension decreases the RMS by about 30%.

	Synchronization error RMS	
$\overline{d}$	FLOPSYNC	FLOPSYNC-QACS
± 0.01	0.134	0.100
± 0.02	0.195	0.141
± 0.04	0.279	0.200
± 0.05	0.313	0.223
± 0.1	0.444	0.314
± 0.2	0.631	0.447
± 0.4	0.893	0.632
$\pm(\sqrt{2}-1)$	0.908	0.643

Table I: RMS values of the simulation campaign.

VII. Conclusions and future work

A control-based time synchronization mechanism for WSNs, called FLOPSYNC-QACS, was proposed for reducing the degradation effect due to quantization of both corrective actions and synchronization error. FLOPSYNC-QACS was implemented in a real WSN, and experimental results back up the proposed solution.

As a future work, we plan to perform some study and experimental analysis so as to evaluate the power consumption requested by the proposed methodology compared to alternative state-of-the-art approaches.

References

[1] Gartner, 2015, http://www.gartner.com/newsroom/id/3165317.
[2] L. Mainetti, L. Patrono, and A. Vilei, "Evolution of wireless sensor networks towards the internet of things: A survey," in *SoftCOM*, Sept 2011, pp. 1–6.
[3] N. Khalil, M. R. Abid, D. Benhaddou, and M. Gerndt, "Wireless sensors networks for internet of things," in *ISSNIP*, April 2014, pp. 1–6.
[4] Y. C. Wu, Q. Chaudhari, and E. Serpedin, "Clock synchronization of wireless sensor networks," *IEEE Sign. Proc. Mag.*, vol. 28, no. 1, pp. 124–138, Jan 2011.
[5] F. Sivrikaya and B. Yener, "Time synchronization in sensor networks: a survey," *IEEE Network*, vol. 18, no. 4, pp. 45–50, July 2004.

[6] G. Mao, B. Fidan, and B. D. Anderson, "Wireless sensor network localization techniques," *Comp. Netw.*, vol. 51, no. 10, pp. 2529–2553, 2007.

[7] T. Schmid, P. Dutta, and M. B. Srivastava, "High-resolution, low-power time synchronization an oxymoron no more," in *IPSN*, 2010, pp. 151–161. [Online]. Available: http://doi.acm.org/10.1145/1791212.1791231

[8] A. Leva, F. Terraneo, L. Rinaldi, A. V. Papadopoulos, and M. Maggio, "High-precision low-power wireless nodes' synchronization via decentralized control," *IEEE Trans. Cont. Syst. Tech.*, vol. 24, no. 4, pp. 1279–1293, July 2016.

[9] R. Silva, J. S. Silva, and F. Boavida, "Mobility in wireless sensor networks – survey and proposal," *Comp. Comm.*, vol. 52, pp. 1–20, 2014.

[10] N. Aakvaag, M. Mathiesen, and G. Thonet, "Timing and power issues in wireless sensor networks: An industrial test case," in *ICPP*, 2005, pp. 419–426.

[11] J. Polastre, R. Szewczyk, and D. Culler, "Telos: enabling ultra-low power wireless research," in *IPSN*, 2005, pp. 364–369.

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

[13] F. Terraneo, A. Papadopoulos, A. Leva, and M. Prandini, "Flopsync-qacs: Quantization- aware clock synchronization for wireless sensor networks," in *IEEE 4th International Workshop on Real-Time Computing and Distributed systems in Emerging Applications*, November 2016. [Online]. Available: http://www.es.mdh.se/publications/4559-

[14] M. Maróti, B. Kusy, G. Simon, and A. Lédeczi, "The flooding time synchronization protocol," in *SenSys*, 2004, pp. 39–49.

[15] F. Ferrari, M. Zimmerling, L. Thiele, and O. Saukh, "Efficient network flooding and time synchronization with Glossy," in *IPSN*, 2011, pp. 73–84.

[16] F. Terraneo, L. Rinaldi, M. Maggio, A. V. Papadopoulos, and A. Leva, "FLOPSYNC-2: Efficient monotonic clock synchronisation," in *RTSS*, 2014, pp. 11–20.

[17] S. Yoon, C. Veerarittiphan, and M. Sichitiu, "Tiny-sync: Tight time synchronization for wireless sensor networks," *ACM Trans. Sens. Netw.*, 2007.

[18] A. Leva and F. Terraneo, "Low power synchronisation in wireless sensor networks via simple feedback controllers: the FLOPSYNC scheme," in *ACC*, 2013, pp. 5017–5022.

[19] F. Ren, C. Lin, and F. Liu, "Self-correcting time synchronization using reference broadcast in wireless sensor network," *IEEE Wireless Comm.*, 2008.

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

[21] 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 *RTSS*, 2015, pp. 175–184.

[22] R. Lim, B. Maag, and L. Thiele, "Time-of-flight aware time synchronization for wireless embedded systems," in *EWSN*, 2016, pp. 149–158. [Online]. Available: http://dl.acm.org/citation.cfm?id=2893711.2893732

[23] F. Terraneo, A. Leva, and W. Fornaciari, "Demo: A high-performance, energy-efficient node for a wide range of wsn applications," in *EWSN*, 2016, pp. 241–242. [Online]. Available: http://dl.acm.org/citation.cfm?id=2893711.2893753

[24] V. Handziski, A. Köpke, A. Willig, and A. Wolisz, "TWIST: A scalable and reconfigurable testbed for wireless indoor experiments with sensor networks," in *REALMAN*, 2006, pp. 63–70. [Online]. Available: http://doi.acm.org/10.1145/1132983.1132995

[25] L. Lamport, "Time, clocks, and the ordering of events in a distributed system," *Commun. ACM*, vol. 21, no. 7, pp. 558–565, Jul. 1978.

[26] D. L. Mills, "Network time protocol (NTP)," Network Working Group Report, Tech. Rep. RFC-958, 1985.

[27] F. Cristian, "Probabilistic clock synchronization," *Distributed Computing*, vol. 3, no. 3, pp. 146–158, 1989.

[28] R. Gusella and S. Zatti, "The accuracy of the clock synchronization achieved by TEMPO in berkeley UNIX 4.3BSD," *IEEE Trans. Soft. Eng.*, vol. 15, no. 7, pp. 847–853, Jul 1989.

[29] J. He, P. Cheng, L. Shi, J. Chen, and Y. Sun, "Time synchronization in WSNs: A maximum-value-based consensus approach," *IEEE Trans. Aut. Cont.*, vol. 59, no. 3, pp. 660–675, March 2014.

[30] A. V. Papadopoulos, F. Terraneo, A. Leva, and M. Prandini, "Switched control for quantized feedback systems: invariance and limit cycles analysis," 2017, submitted. Available as arXiv:1701.07482.

[31] F. Terraneo, *Miosix embedded OS*, sources available at `http://miosix.org`.