Authors' Accepted Manuscript (AAM): this is the accepted version of the paper. Citation information: DOI: 10.1109/TII.2015.2396003,
IEEE TRANSACTIONS ON INDUSTRIAL INFORMATICS, VOL. 11, NO. 3, JUNE 2015

Implementation and Evaluation of the Reference Broadcast Infrastructure Synchronization Protocol

Gianluca Cena, *Senior Member*, *IEEE*, Stefano Scanzio, *Member*, *IEEE*,
Adriano Valenzano, *Senior Member*, *IEEE*, and Claudio Zunino

***Abstract*—This paper describes Reference Broadcast Infrastructure Synchronization (RBIS), a clock synchronization protocol for IEEE 802.11 infrastructure wireless networks. The protocol is especially tailored for industrial and home automation networks, and in many application contexts it offers several advantages compared with other solutions targeted at similar purposes. RBIS has been conceived to rely on conventional Wi-Fi equipment, and in particular on unmodified access points. It is based on the master/slave approach and follows the receiver/receiver paradigm.**

An implementation of RBIS—carried out completely in software and based on timestamps taken at the interrupt handler level—has been developed, which achieves a synchronization error below $3\,\mu$s. Then, a simple distributed hard real-time control application has been set up, which consists in two PCs running RTAI and connected through Wi-Fi. The actuation error, measured on the generation of synchronous pulses, is strictly below $13\,\mu$s.

***Index Terms*—Clock Synchronization Protocols, Infrastructure Wi-Fi networks, IEEE 802.11, PTP, RBS, RBIS**

I. Introduction

SYNCHRONIZATION among devices and applications is playing an increasingly important role in today's industrial plants. A very effective way to achieve so is using a Clock Synchronization Protocol (CSP), which exploits the underlying communication network to keep the local clocks of the different nodes aligned. Besides factory automation, CSPs [1], [2] are used in other application areas like robotics, home automation, and sensors networks, to cite a few. The most popular CSP is probably the Network Time Protocol (NTP). Since it was mainly conceived for Wide Area Networks (WANs), its accuracy rarely meets the typical requirements of factory automation [1]. The "de facto" standard for synchronization in wired industrial networks is IEEE 1588 [3], also known as Precision Time Protocol (PTP). For this reason, it has been included in many industrial Ethernet solutions.

On the other hand, the widespread availability of Wireless Local Area Networks (WLANs), and particularly IEEE 802.11 [4], whose related equipment (Wi-Fi) can be currently found at quite low price, has led in recent years to the adoption of this technology in hybrid wired-wireless networks for factory automation, and in particular in all the cases where cables are hard-to-place, expensive, or suffer from tear and wear [5]–[8]. Of course, wireless extensions based on other technologies exist as well, e.g., IEEE 802.15.1 for Wireless Personal Area Networks (WPANs) or IEEE 802.15.4 for Wireless Sensor Networks (WSNs). However, their throughput and interoperability with Ethernet are noticeably lower than IEEE 802.11.

Many research activities were addressed in the past years to improve determinism of Wi-Fi [9]. The related approaches are often based on Time Division Multiple Access (TDMA) [10]–[13]. This requires precise synchronization of nodes, which can be easily accomplished through a CSP. In addition, CSPs make it possible to interconnect devices over the air in several industrial applications where determinism is relevant. To this extent, it is sufficient to send commands in advance, so that, despite the unpredictable transmission delays over Wi-Fi, they will reach the target devices before the deadline with high probability. Then, the CSP takes care of ensuring precise actuation times at the application level.

The Time Synchronization Function (TSF) defined in IEEE 802.11 and available on Wi-Fi devices can be seen as a basic CSP. TSF timers (with a resolution of $1\,\mu$s) are kept aligned in the different wireless stations (denoted STAs in the IEEE 802.11 terminology) with an accuracy in the order of ten μs. Unfortunately, they are embedded into adapters. This means that hardly the reference clock—typically located in the access point (AP)—can be bound to an external time source.

Conventional CSPs can also be layered on top of IEEE 802.11. PTP can be effectively applied to infrastructure Basic Service Sets (BSS)—i.e., Wi-Fi networks based on an AP—but modifications are required to the AP firmware (f/w) and, sometimes, hardware (h/w). In [14], a PTP-like protocol was implemented using a *virtual AP* running on an industrial PC equipped with a wireless adapter based on a softMAC chipset—i.e., the MAC sublayer management entity (MLME) is realized completely in software (s/w). In particular, a modified version of PTP was defined where the AP acts as the time master. To reduce network load, *beacon* frames were exploited both as synchronization events and to broadcast the timestamps obtained by the AP. In [15] a similar implementation was analyzed for hybrid wired-wireless networks.

A number of other CSP solutions exist, e.g., the Flooding Time Synchronization Protocol (FTSP) [16] and the Reference Broadcast Synchronization (RBS) mechanism [17], which were conceived for WSNs. In such a kind of networks, the impact of the synchronization protocol on power consumption

This work was partially supported by the Ministry of Education, University, and Research of Italy (MIUR) in the framework of the Project Wi-Fact "WIreless FACTory and beyond" (DM53543). The authors are with the National Research Council of Italy, Istituto di Elettronica e di Ingegneria dell'Informazione e delle Telecomunicazioni (CNR-IEIIT), I-10129 Torino, Italy (e-mail: stefano.scanzio@ieiit.cnr.it).

must be taken into account as well, besides its precision [18]. FTSP relies on the *sender/receiver* paradigm and resembles somehow TSF. RBS, instead, is a quite interesting approach that exploits the *receiver/receiver* paradigm. This means, that every node takes timestamps on specific synchronization events modeled as broadcast messages. These timestamps are then exchanged among nodes, so that they are enabled to determine how much their local clocks are displaced. The biggest advantage of RBS is that all the sender latencies are left out of the *critical synchronization path* (i.e., the path between the source of synchronization events and their recipients), and hence accuracy is noticeably better, even in the case timestamps are taken in s/w. Unfortunately, these solutions are unable to satisfy all the typical requirements of industrial applications (synchronization accuracy, *master/slave* timing hierarchy with external time source, low bandwidth utilization, etc.). Moreover, they were not designed with Wi-Fi infrastructure BSSs in mind, and therefore they do not exploit some peculiar features of these kinds of networks, such as the use of *beacons* in order to save bandwidth.

The most recent Wi-Fi specification (IEEE 802.11-2012) [4] incorporates some concepts related to time synchronization—as previously defined in amendment IEEE 802.11v—namely *timing measurement* and *timing advertisement*. Although effective, these approaches are probably not the best solution in several practical contexts. Moreover, such parts of the standard are not mandatory and currently they are implemented only in a limited number of Wi-Fi chipsets. The *timing measurement* mechanism allows a STA to synchronize to another STA using a pair of messages, the timing measurement frame and the corresponding acknowledgment frame. Timestamps are taken on the transmission of such frames with 10 ns resolution. Propagation delays are taken into account as well. Unfortunately, this mechanism causes non-negligible overheads. In fact, in a typical centralized network where several STAs have to synchronize to a common time source, the time reference must periodically send (unicast) confirmed messages to every single STA. Instead, the *timing advertisement* mechanism is basically a way to flood the difference (expressed in ns) between the TSF of the transmitting STA (in μs) and the Coordinated Universal Time (UTC), so that UTC is made available to all receiving STAs. This mechanism suffers from the typical problems of CSPs based on flooding, i.e., propagation delays and in-node latencies are not compensated.

In [19] a new CSP, denoted Reference Broadcast Infrastructure Synchronization (RBIS), was first introduced and its performance analyzed through simulation. In this paper, a more detailed description is provided for the protocol and a comparison is carried out with other CSPs. Moreover, a real implementation has been developed and its accuracy measured in a variety of experimental conditions. The RBIS protocol is specifically targeted at Wi-Fi networks operating in infrastructure mode. Its main advantage is that, unlike the competing solutions, no modifications at all are required to the AP. Possibly, small changes can be brought to the drivers of network adapters in the synchronized STAs in order to improve timestamp accuracy. As a consequence, RBIS can be profitably adopted to achieve precise synchronization in all the Wi-Fi networks already deployed in factory, process, building, office, and home automation systems. In a world where more than 200 million households use Wi-Fi [20], this certainly represents a cheaper alternative to the replacement of the preexisting wireless network infrastructures.

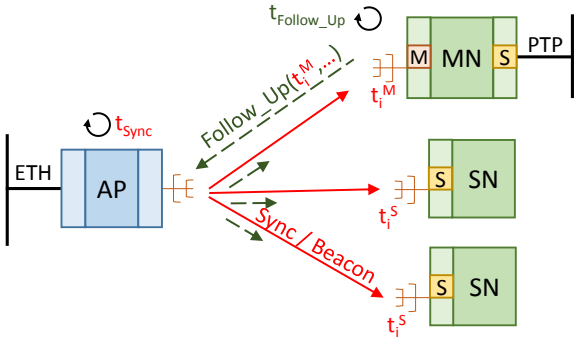


Fig. 1. The RBIS clock synchronization protocol.

Pragmatically, a tradeoff has to be found in many cases between the costs involved by the need to upgrade the existing devices and the required synchronization quality (which depends on the specific application). If compensation of propagation delays is not required (as happens in the several scenarios) the accuracy and precision RBIS provides on conventional Wi-Fi equipment is comparable with approaches that explicitly require purposely modified APs. Besides low implementation complexity and full compatibility with the existing equipment, other features like the *master/slave* timing hierarchy and the low communication overhead make RBIS a good candidate for synchronization in wireless industrial networks.

The paper is organized as follows: Section II describes RBIS in detail whereas Section III highlights its advantages and drawbacks with respect to other synchronization protocols. A real RBIS implementation on conventional PCs is described in Section IV, while experimental results under different operating conditions are reported in Section V.

II. The RBIS protocol

RBIS can be seen as a specialization of RBS to IEEE 802.11 networks operating in *infrastructure* mode (by far the most typical case for Wi-Fi). We decided to name it as RBIS to stress the specific application contexts it was conceived for, and to highlight the peculiar differences compared to RBS. RBIS is a *master/slave* CSP that is based on the *receiver/receiver* paradigm. Two messages are used for node synchronization, namely *Sync* and *Follow_Up*. *Sync* messages are the common events used for synchronization. In the configuration analyzed here (corresponding to the third one in [19]), they coincide with *beacon* frames periodically broadcast by the AP. Hence, their generation period t_{Sync} corresponds to the *beacon interval*, whose default value is typically 100 ms. Each node running RBIS takes a timestamp on the reception of every *Sync* message. Then, these values are used to estimate how much its local clock differs from the *reference time*. As shown in Figure 1, each BSS synchronized through RBIS has to include at least two RBIS-enabled STAs, i.e., exactly one

Master Node (MN) and one or more Slave Nodes (SN). MN is the *reference time* in the BSS (which constitutes a broadcast domain), while each SN uses RBIS to synchronize its local clock to MN. In this master/slave hierarchy—which is typical of industrial CSPs like PTP—MN can have its local clock synchronized to an external time source—either GPS or a higher-level network running, e.g., PTP. In the latter case MN acts as a time slave for PTP and propagates the reference time from the (wired) plant backbone to the wireless extension.

Let t_i^M and t_i^S be the timestamps recorded on the reception of the $i\text{-}th$ *Sync* message by MN and (a generic) SN, respectively. In order to adjust its local clock to the reference time, every SN must know the values of the timestamps acquired by MN, too. To this extent, *Follow_Up* messages are sent by MN to SNs. Depending on the number of SNs, *Follow_Up* messages can be delivered using either broadcast or unicast transmission services. If there are many SNs, it is advantageous to use broadcast transmissions, in order to serve all SNs at the same time with a single message. In Wi-Fi, broadcast messages are not confirmed, and so they have a lower probability to be delivered correctly. For this reason they are typically sent with a more robust modulation scheme that uses a reduced bit rate (as low as $1\,\mathrm{Mb/s}$). Conversely, sending unicast *Follow_Up* messages improves protocol robustness thanks to the acknowledgment mechanism, which permits to increase the transmission speed, but a separate copy of the same message must be sent to each SN. Another non-negligible drawback of unicast *Follow_Up* messages is that MN must know all the SNs in the BSS. This information can be obtained either through offline configuration or by exploiting a sort of PTP *General* message, which must be sent by each SN to MN before synchronization is started.

The transmission of *Follow_Up* messages from MN to SNs can be triggered either by the arrival of a *Sync* message or periodically. In the first case, the *Follow_Up* message should be sent as soon as possible after a *Sync* message has been received. Since clock correction on SNs can be performed only when the timestamp t_i^M of MN becomes available, doing so reduces the synchronization error. This approach has the drawback of increasing the network load, especially when the transmission rate of *Sync* messages is high—as in the case when t_{Sync} equals the beacon interval.

A better solution is sending *Follow_Up* messages periodically, with a period t_{Follow_Up} that is larger than t_{Sync}. In this case, not necessarily *Follow_Up* messages have to phased exactly with *Sync* messages. Moreover, more than one timestamp can be collected in each message. Let $n_{ts}^{min} = \lceil t_{Follow\_Up}/t_{Sync} \rceil$ be the number of timestamps t_i^M acquired by MN in the interval t_{Follow_Up} (n_{ts}^{min} is typically greater than one) and n_{ts} the number of timestamps included in each *Follow_Up* message. In order to exploit all timestamps for clock correction, n_{ts} must not be lower than n_{ts}^{min}, i.e., $n_{ts} \geq n_{ts}^{min}$. Setting n_{ts} to a value strictly greater than n_{ts}^{min} may improve RBIS robustness against packet losses. In fact, (unconfirmed) broadcast messages in Wi-Fi have a non-negligible likelihood to be lost. If n_{ts} is set as a multiple of n_{ts}^{min}, then missing a *Follow_Up* message unlikely causes the definitive loss of all the timestamps it contains. A similar approach was used in [21], where it was shown that doing so may increase synchronization resilience and precision.

A reasonable choice for RBIS, when $t_{Sync} = 100\,\mathrm{ms}$ and s/w-based timestamps are employed (affected by high jitters) is to set $t_{Follow\_Up} = 1\,\mathrm{s}$ and $n_{ts} = 20$. This value for t_{Follow_Up} causes a negligible communication overhead and achieves a synchronization error in the order of some μs. Instead, the value chosen for n_{ts} improves the RBIS ability to tolerate the loss of *Follow_Up* messages without increasing the network load excessively.

The timestamps of MN included in each *Follow_Up* message $(t_i^M, t_{i-1}^M, ..., t_{i-n_{ts}+1}^M)$ must be unambiguously paired by each SN with the timestamps it took on the very same *Sync* messages $(t_i^S, t_{i-1}^S, ..., t_{i-n_{ts}+1}^S)$. To this extent, a unique *identifier* must be included in the *Follow_Up* message for every one of the n_{ts} timestamps it embeds. In the most general case where *Sync* messages can be received from more than one nearby AP, the pair $\langle AP\_MAC, BEAC\_TIME \rangle$ can be used, where AP_MAC is the MAC address of the AP ($6\,\mathrm{bytes}$) and $BEAC\_TIME$ is the *Timestamp* field included in every beacon frame ($8\,\mathrm{bytes}$). Doing so improves synchronization accuracy and also achieves fault tolerance.

In the typical case when only one AP is taken into account (the one to which all SNs are associated), $BEAC\_TIME$ is sufficient to identify timestamps uniquely. Since the payload of *Follow_Up* messages has low information density, compression techniques can be used to reduce their size. For instance, differences between subsequent timestamps could be stored in the place of absolute values. When a *Follow_Up* message is received, each SN runs a *control algorithm* on the paired timestamps, which either tunes the slave clock directly or updates the parameters of a *virtual clock*.

III. Comparison with other solutions

A brief comparison is carried out between RBIS and other wireless CSPs—particularly PTP—from both quantitative and qualitative points of view. Some basic properties and key features of such protocols are also summarized in Table I.

A. Synchronization quality vs. communication delays

The quality of synchronization achieved by a CSP (accuracy and precision) heavily depends on its ability to evaluate and compensate delays on the critical path [17], both inside nodes and over the air. Since traversal times may vary greatly for switches and APs—depending also on message queuing and, for the latter, contentions for the transmission medium—these devices should be left out of the critical path (unless they are implemented as boundary clocks).

Unlike PTP, RBIS is not able to evaluate propagation delays. This is because MN and SNs do not communicate directly over the air, but they exploit the relay function carried out by a conventional, unmodified AP. Considering the target accuracy we wish to achieve (in the order of some μs), this is often not a serious limitation from a practical point of view, for the reasons explained below.

1) Clock offset estimation: In the following text a *master/slave* timing hierarchy is assumed, but the considerations are valid more in general. Although RBIS does not estimate propagation delays, its *receiver/receiver* approach offers some advantages compared to *sender/receiver* CSPs like PTP, FTSP, and so on. In the latter case, MN coincides with the *sender* while SNs are *receivers*, whereas in the former both MN and SNs are considered *receivers*. Whatever the approach, each slave node estimates the instantaneous offset o_i between its local clock and the reference clock as the difference between the timestamps obtained on a common event by itself (t_i^S) and by the time master (t_i^M), i.e., $o_i = t_i^S - t_i^M$. A filtering algorithm can be possibly used on a set of subsequent timestamps to reduce noise by smoothing jitters. Each value o_i is affected by latencies due to message propagation and timestamping, which affect synchronization quality negatively.

In Figure 2, the in-node and communication latencies on the critical path for synchronization are shown for the two kinds of CSP. In the *sender/receiver* case, the latencies due to the sender node (d_{send}^M), the propagation delay on the transmission support between MN and SN (d_{prop}^{MS}, which corresponds to about $3\,\mathrm{ns/m}$), and the receiver node (d_{rec}^S), contribute to the *offset estimation error*. In the absence of any compensation (besides the frame duration, which can be computed) it corresponds to $\epsilon^{send/rec} = d_{send}^M + d_{prop}^{MS} + d_{rec}^S$. If the sender takes timestamps on the interrupt that follows successful transmission, as in [22], a lower $\epsilon^{send/rec}$ is expected. Instead, in the *receiver/receiver* case, the uncompensated offset estimation error is $\epsilon^{rec/rec} = (d_{prop}^S - d_{prop}^M) + (d_{rec}^S - d_{rec}^M)$, which is typically smaller than $\epsilon^{send/rec}$.

2) Path delay compensation: Generally speaking, PTP cannot be satisfactorily implemented using only the *Sync-Follow_Up* mechanism, because $\epsilon^{send/rec}$ leads to a systematic error that likely is excessively high. Similar considerations also apply to other *sender/receiver* CSPs like FTSP and the timing advertisement mechanism of IEEE 802.11-2012. For this reason the *Delay_Req-Delay_Resp* mechanism is foreseen by PTP to estimate propagation delays, so that they can be properly compensated. Its effectiveness depends on node mobility. The components related to in-node latencies (d_{send} and d_{rec}) can be satisfactorily evaluated in the hypothesis of symmetry between master-to-slave and slave-to-master delays. Unfortunately, as shown in [22], transceiver behavior is rarely symmetric, and this leads to a residual systematic error. On the other hand, latencies due to signal propagation over the communication support (d_{prop}) suffer from an uncertainty that depends on the rate at which *Delay_Req-Delay_Resp* frames are sent. When nodes move frequently and quickly, ensuring proper compensation may affect the available bandwidth in a non-negligible way.

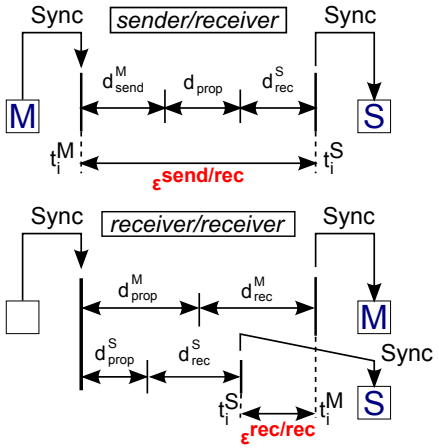


Fig. 2. Offset estimation error in sender/receiver and receiver/receiver CSPs.

TABLE I
MAIN FEATURES OF CSPS FOR WI-FI INFRASTRUCTURES.

	PTP	RBIS	RBS	FTSP (TSF)	IEEE 802.11v
Prop. delays compensation	**Y**	N	N	N	**Y**[1]
Master-slave synchronization	**Y**	**Y**	N	**Y**	**Y**
External time source	**Y**	**Y**	N	N[2]	**Y**[3]
Unmodified legacy APs	N	**Y**	**Y**	**Y**	N[4]

[1] Timing measurement mechanism
[2] If legacy TSF mechanism is used
[3] Timing advertisement mechanism
[4] Modifications required also to STAs

In the case of RBIS, the part of $\epsilon^{rec/rec}$ due to signal propagation corresponds to the difference between the propagation delays from the AP to SN (d_{prop}^S) and from the AP to MN (d_{prop}^M). Since the area covered by real-world BSSs is not very large—the maximum distance allowed between the STAs and the AP is in the order of one hundred meters at most—the error incurred because propagation delays are not compensated (below $1\,\mu\mathrm{s}$) is typically lower than the jitter caused by s/w timestamps. Therefore, it could be neglected in s/w-based implementations. Using h/w timestamps in RBIS decreases both in-node latencies and the related jitters noticeably, but the error due to the uncompensated propagation delay remains.

Although the inability of RBIS to estimate propagation delays is a clear limitation, it can be overcome in several practical cases. For instance, if the position of the nodes is known with good approximation—as happens when limited movements are allowed for them, as in devices mounted on the end effector of a robot arm or fastened to the axes of a numerically controlled machine—such error can be compensated statically via suitable parametrization at system start-up.

B. Advantages and Drawbacks

The most important features of RBIS are described and compared with other CSPs in the case of hybrid wired-wireless networks based on Wi-Fi infrastructures.

1) Timing hierarchy: In the same way as PTP and FTSP—but unlike RBS—RBIS is a *master/slave* synchronization protocol. Although this feature does not affect performance, it is quite desirable in industrial networks because it permits to set up a timing hierarchy that is consistent with the layered architecture of factory automation systems (which foresees, e.g., company, cell, and field levels) and enables synchronization to an *external* time source.

MN can serve as a gateway to other CSPs: for instance, as shown in Figure 1, it can be seen as a "hybrid" *boundary clock* that acts as a PTP slave on the wired backbone (an Ethernet

interface is required for the connection) and as the RBIS master for the BSS (wireless extension). A similar approach was proposed and experimented in [15], where hybrid wired-wireless networks were synchronized through PTP.

2) Backward compatibility: In order to apply PTP to Wi-Fi, the AP ought to: send *Sync* messages (or exploit beacons to this extent [14], [15]) and take timestamps on them; send timestamps using *Follow_Up* messages (or embed them directly inside *Sync* messages); take timestamps on *Delay_Request* messages and reply to them; act as a time slave on the wired backbone (or use an external time source); and, finally, act as the time master for the BSS. Timestamps embedded in PTP messages must have ns resolution (although resolution and precision of local clocks can be lower). For a complete implementation of the related protocol features, modifications are unavoidably required at the f/w level and, sometimes, even at the h/w level. This is true also in the case of the timing advertisement mechanism of IEEE 802.11-2012 (that extends the TSF of Wi-Fi), as well as the time measurement mechanism, which require compliant chipsets.

In RBIS, no modifications at all must be brought to the AP. Only STAs require small changes: besides RBIS protocol implementation, the drivers of Wi-Fi adapters should be enabled to acquire timestamps at the Interrupt Service Routine (ISR) level—this is not needed in the case of h/w timestamps. As a consequence, compatibility with legacy wireless equipment (complying with IEEE 802.11n and earlier) is preserved.

3) Number of synchronized nodes: The minimum number of wireless nodes that are required to take part in synchronization in RBIS—if the AP is not considered—is two (MN and SN), while at least 3 distinct nodes are needed in the original RBS version. This is because, although RBIS follows the *receiver/receiver* paradigm, it exploits beacons as the common synchronization events. Conversely, PTP implementations over Wi-Fi (that must necessarily rely on modified APs) are allowed to include as few as one wireless node, which acts as SN.

The latter configuration (the simplest one in a hybrid network) can indeed be deployed in RBIS as well. To this extent, a streamlined MN can be envisaged, whose *only* purpose is to act as a boundary clock. This particular kind of MN complements a legacy AP and takes care of synchronizing the wireless extension to the wired backbone. Most important, it is "universal", i.e., it is completely independent of the AP type, brand, and model. This trick is unnecessary when two (or more) separate devices (but also subsystems/cells) have to be interconnected and synchronized over the air (for instance, when a robot is connected to a PLC using Wi-Fi).

4) Network overhead: In the typical case when *Follow_Up* messages are sent periodically in broadcast, RBIS is quite efficient because only one additional message is required to be sent, with a relatively high period (1 s for the experiments presented here). Its efficiency is comparable to FTSP (where timestamps are embedded on-the-fly in synchronization messages) and can even be higher in the usual case when several timestamps are embedded in the same *Follow_Up* message.

RBIS does not use as much network bandwidth as PTP, where 4 messages (3 of which mandatory) are defined. In particular, the overhead introduced by *Delay_Req* and *Delay_Resp* messages in PTP is quite low in networks that comprise few devices with limited (or no) mobility, but it may grow noticeably when either of these two conditions is not meet. It is also more efficient than RBS, which requires one message per node (that either represents the network synchronization event or conveys the locally acquired timestamp).

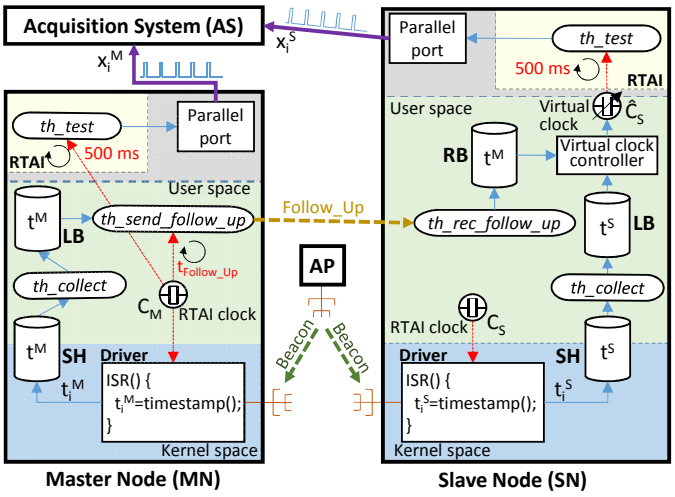


Fig. 3. Testbed for the evaluation of RBIS performance.

As explained in the introduction, the network overhead of RBIS is lower than the timing measurement mechanism of IEEE 802.11-2012 (that uses unicast messages), whereas it is slightly higher (but possibly more precise and accurate) than legacy TSF and the timing advertisement mechanism described in the same standard.

IV. Prototype Implementation

The RBIS protocol has been implemented and evaluated on a testbed made up of two identical PCs and one *acquisition system* (AS). One PC acts as MN and the other as SN. Both PCs are equipped with an Intel Core2Duo E7500 CPU running at 2.93 GHz, with 3 MB of L2 cache and a 1.066 GHz front-side bus. Communication and synchronization take place via wireless network adapters through one AP. Wi-Fi equipment complies to IEEE 802.11n and uses 2 spatial streams. The RTAI [23] hard real-time scheduler version 3.9 was installed and calibrated in a patched Linux Kernel version 2.6.38.8. The RTAI timer is the timing source of RBIS. It is used both to obtain timestamps and to perform synchronized actuations (for evaluating CSP performance). All the main components of the testbed are reported in Figure 3. The timestamps t_i^M and t_i^S are taken at the very beginning of the ISR procedure in the drivers of MN and SN, respectively, so as to be as close as possible to the interrupt raised by the wireless network adapters. Each timestamp is uniquely identified by means of the $BEAC\_TIME$ field in the related beacon. Timestamps (and their identifiers) are moved from kernel to the user space by means of a shared memory area `SH` and collected by the thread `th_collect` in a local buffer `LB`. The `th_send_follow_up` thread of MN cyclically broadcasts *Follow_Up* messages (with period $t_{Follow\_Up} = 1\,\mathrm{s}$), each of

which includes the n_{ts} most recent timestamps (and identifiers) stored in the `LB` of MN. When a *Follow_Up* message is received by SN, the `th_rec_follow_up` thread adds the embedded timestamps (and identifiers) to a second buffer `RB`, removing duplicates. The *virtual clock controller* matches the most recent k timestamps in `LB` and `RB`, and updates the virtual clock using a control algorithm based on Least-Squares Linear Regression (LSLR). It is worth remarking that the RBIS protocol is completely implemented in a non-hard real-time context.

Different types of CSP control algorithms were proposed in the literature. The two most popular ones are based on LSLR [16], [17] and Kalman filter [24], [25]. Algorithms based on Kalman filter are optimal under the hypothesis of gaussian noise in both measurements (i.e., timestamps) and state equations. Unfortunately, noise is usually not normally distributed, because the statistical distribution of jitters on timestamps is often asymmetric. The same holds for state noise, because it depends mostly on the thermal drift, which can be hardly modeled as a gaussian distribution. For these reasons (including simplicity), and since the comparison between different control algorithms is out of the scope of this work, the algorithm described in [17] and based on LSLR has been chosen. The resulting implementation runs in about 34000 clock cycles (corresponding to $\sim 12\,\mu$s in the proposed setup), and its execution time does not depend on the number of points used for computing the regression line. For the sake of truth, further optimizations are still possible. The iterative mechanism used in [17] to discard outliers has been disabled, because experimental results did not show any noticeable improvement in our specific testbed.

In order to evaluate CSP performance, both PCs were programmed to periodically generate pulses on their parallel ports (connected via PCIe) with period 500 ms. Pulses are scheduled at predefined absolute instants by the RTAI hard real-time `th_test` thread, by using the (uncorrected) clock on MN (C_M) and the virtual clock on SN ($\hat{C}_S$, adjusted through RBIS). The `th_test` thread resembles, from every point of view, a task in a distributed real-time control application, which is synchronized through Wi-Fi and is scheduled, for instance, by RTAI. Timestamps are acquired by AS on the *rising edges* of the resulting waveforms (x_i^M and x_i^S for MN and SN, respectively) with a precision of 10 ns and a resolution of 1 ns. They are first compensated as described in Subsection V-A, and then post-processed to evaluate the synchronization error, at both the protocol and the application levels.

V. Experimental Evaluation

The accuracy of the acquisition system, as well as the performance of the compensation technique used to estimate the synchronization error reliably, have been first evaluated. Then, the RBIS synchronization error has been analyzed through a set of experiments based on the above testbed, for different configuration parameters and operating conditions (with respect to both the interfering load inside nodes and the network load). Finally, the RBIS protocol has been uninterruptedly tested for 10 whole days. It is important to stress

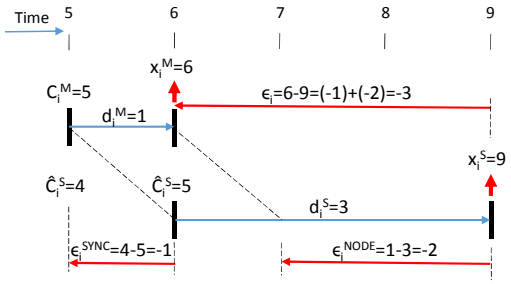


Fig. 4. Characterization of acquisition system timings.

that the reported results concern a specific configuration, based on conventional PCs and open-source s/w. Nevertheless, they provide a clear indication on what one may expect in general from a real RBIS implementation.

A. Characterization of the acquisition system

Let $\epsilon_i = x_i^M - x_i^S$ be the *actuation error*, defined as the difference between the i-th timestamps x_i^M and x_i^S, acquired by AS on corresponding pulses in the synchronized signals output by MN and SN (see Figure 4). It represents the overall system error (i.e., the precision with which nodes perform synchronized actions) and is made up of two contributions, that is, the *synchronization error* ϵ_i^{SYN} and the *in-node error* ϵ_i^{NODE}

$$\epsilon_i = \epsilon_i^{SYN} + \epsilon_i^{NODE} \tag{1}$$

Both contributions are relevant for characterizing the performance of the CSP and its specific implementations as well. The synchronization error is evaluated as the difference between the *slave clock* (corrected by the synchronization protocol) $\hat{C}_i^S$ at the time when MN schedules the i-th synchronized actuation and the related *master clock* C_i^M, $\epsilon_i^{SYN} = \hat{C}_i^S - C_i^M$. Basically, it depends on the CSP, clock discipline algorithm, and timestamp accuracy. Instead, the in-node error is the difference between the in-node *actuation latencies* of the two nodes (d_i^M and d_i^S, respectively), $\epsilon_i^{NODE} = d_i^M - d_i^S$. It is directly related to the real-time properties of the system scheduler and h/w architecture.

Besides ϵ_i^{NODE}, the real-time capabilities of the underlying platform affect clock synchronization accuracy too (particularly if timestamps are acquired in s/w), and hence, ϵ_i^{SYN}. Acquiring timestamps in h/w in the wireless adapter improves synchronization accuracy noticeably (i.e., lower ϵ_i^{SYN} is usually achieved), but leaves open the problem of how the adapter has to be interfaced to the s/w that carries out actuation. For instance, the connection between the clock used to take h/w timestamps in the NIC and the one of the real-time scheduler is typically not a trivial task.

For both MN and SN the actuation latency d_i—responsible of the in-node error ϵ_i^{NODE}—is made up of two contributions, that is, the *s/w latency* d_i^{sw} and the *h/w latency* d_i^{hw}. Thus, $d_i = d_i^{sw} + d_i^{hw}$. In our testbed, d_i^{sw} is mostly due to the real-time scheduler, and can be evaluated approximately as the time elapsing between the scheduled release time of the task that generates the i-th pulse on the parallel port and the instant when its execution is actually started by RTAI. Instead, d_i^{hw} is

the time that elapses between the instant when the application writes the command into the register of the parallel port and the instant when the output signal actually goes high. It is mostly due to the PCIe bus and the rise time of the parallel output ports.

Both d_i^{sw} and d_i^{hw} can be expressed as the sum of a fixed and a random contribution, that is, $d_i^{sw} = \bar{d}^{sw} + j_i^{sw}$ and $d_i^{hw} = \bar{d}^{hw} + j_i^{hw}$, where $\bar{d}^{sw}$ and $\bar{d}^{hw}$ are the *average delays* while j_i^{sw} and j_i^{hw} refer to the offsets specifically related to the $i\text{-}th$ actuation. Because of the above definitions, the sets of j_i^M and j_i^S values have both a null mean value, and hence they represent *jitters*. Let $j_i = j_i^{sw} + j_i^{hw}$ be the overall in-node jitter (h/w plus s/w) of any given node. Since MN and SN in our testbed are based on two identical PCs, $\bar{d}^{sw}$ is about the same for them, and the same holds for $\bar{d}^{hw}$. As a consequence, $\epsilon_i^{NODE} \simeq j_i^M - j_i^S$. This implies that, in our specific experimental setup, the mean value of ϵ_i^{NODE} values is almost null. Hence, the in-node error does not cause any systematic error (that affects accuracy) but only worsens precision.

The most important quantity used to characterize a CSP and how it reacts to specific operating conditions (local system load, network load, configuration parameters, etc.) is certainly the synchronization error ϵ_i^{SYN}. Evaluating ϵ_i^{SYN} is not easy, because what AS actually measures are x_i^M and x_i^S, and not C_i^M and C_i^S. Timestamps x_i^M and x_i^S are taken with the time base of AS, which is not the same as those used by MN and SN. However, this does not lead to any problems since only their difference ϵ_i is taken into account.

A compensation method is proposed here in order to obtain a reliable estimation $\hat{\epsilon}_i^{SYN}$ of ϵ_i^{SYN}, which operates by compensating x_i samples. Whenever AS acquires and stores a sample x_i, each node (PC) also stores the related scheduler latency d_i^{sw} (measured locally by the PC itself). Compensated samples $\tilde{x}_i$ are then obtained from the uncompensated ones x_i as $\tilde{x}_i = x_i - d_i^{sw}$.

Let $\hat{\epsilon}_i^{SYN} = \tilde{x}_i^M - \tilde{x}_i^S$ be the difference between compensated samples. It can be expressed as

$$\hat{\epsilon}_i^{SYN} \quad = \quad \epsilon_i^{SYN} + \epsilon_i^{RES} \tag{2}$$

where $\epsilon_i^{RES} = d_i^{M_{hw}} - d_i^{S_{hw}}$ is the *residual in-node error* after compensation of s/w jitters. Since the h/w of the nodes is identical, $\epsilon_i^{RES} = j_i^{M_{hw}} - j_i^{S_{hw}}$ and, as it will be shown, it is quite low. Hence, $\hat{\epsilon}_i^{SYN}$ is a reliable estimate of ϵ_i^{SYN}.

A set of experiments like those reported in [26] were carried out to characterize ϵ_i^{RES}, which operate by evaluating the actuation precision of each node (either MN or SN) when considered alone. A PC was programmed to generate a train of pulses on its parallel port with period $500\,\mathrm{ms}$. A suitable filter based on LSLR was applied on both measured samples x_i and compensated samples $\tilde{x}_i$, in order to cope with the frequency deviation between the local clock and the AS clock, and to smooth quartz drifts. To this extent, 10 adjacent samples were considered for each sample. LSLR on a subset of samples centered around the $i\text{-}th$ one is a quite effective way to filter the constant components $\bar{d}^{hw}$ and $\bar{d}^{sw}$ out of x_i—as well as the offset between the local and AS clocks. Thus, the difference δ_i between each measured sample x_i and the related regression line can be taken as a satisfactory estimate of jitter j_i. If compensated samples $\tilde{x}_i$ are considered, $\tilde{\delta}_i$ is obtained, which is an estimate of j_i^{hw}. Experimental results are reported in Table II, in the case the PC either is (mostly) idle or experiences a heavy local load condition (due to concurrent disk I/O operations). In the worst case (heavy load and no compensation), the difference between the maximum and the minimum among δ_i values is equal to $19.9\,\mu\mathrm{s} = 12.2\,\mu\mathrm{s} - (-6.7\,\mu\mathrm{s})$, which coincides approximately with the width of the jitter distribution. This value is higher than the one reported in [26] ($9.2\,\mu\mathrm{s}$, obtained with $1\,\mathrm{ms}$ generation period). Performance degradation is due to both dual-core CPUs, which are less deterministic than the single-core ones used in [26], and the fact that the likelihood of cache misses for $500\,\mathrm{ms}$ generation period is much higher.

TABLE II
STATISTICAL CHARACTERIZATION OF THE JITTER ON MEASURED SAMPLES (WITH AND WITHOUT ADDITIONAL LOCAL LOAD ON PCS).

	No Compensation ($\delta_i \approx j_i$)		Compensation ($\tilde{\delta}_i \approx j_i^{hw}$)	
Quantity	No load	Heavy load	No load	Heavy load
μ	0.001	0.001	0.001	0.001
σ	0.350	1.707	0.016	0.037
min	-2.000	**-6.650**	-0.535	**-0.529**
max	6.633	**12.239**	0.160	**0.303**

All values are expressed in μs

In the same heavy load conditions, after compensation of s/w jitters and assuming a symmetric distribution of h/w jitters, $max\left(\left|\tilde{\delta}_i\right|\right) = 0.529\,\mu\mathrm{s}$. This value can be taken as an estimate of the maximum h/w jitter j_{max}^{hw} for both MN and SN. An experimental proof of the effectiveness of the proposed compensation method is that j_{max}^{hw} is (almost) the same as in the no-load condition. This is because, after s/w jitter is compensated, only h/w jitter j^{hw} remains, which does not depend on the PC local load. A proper upper bound on the maximum residual error $\epsilon_{max}^{RES} = max\left(\left|\epsilon_i^{RES}\right|\right)$ can be found as $\left|j_{max}^{M_{hw}}\right| + \left|j_{max}^{S_{hw}}\right| = 2 \cdot max\left(\left|\tilde{\delta}_i\right|\right)$. Thus, it is safe to assume that the value $\hat{\epsilon}_{max}^{SYN}$, obtained by AS after samples are compensated, provides an estimate for the synchronization accuracy ϵ_{max}^{SYN} with a precision within $\pm 1.1\,\mu\mathrm{s}$.

B. Local load

The first set of experiments was aimed at tuning the most important configuration parameter of the LSLR-based control algorithm we chose for RBIS, i.e., the number of samples k used for regression (here, the term "sample" correspond to paired $\left\langle t_i^M, t_i^S \right\rangle$ timestamps). Further, it permitted to evaluate the effects of interfering best-effort tasks (executing on MN and SN) on synchronization accuracy. The clock register, which holds the time inside a node, is updated periodically. Updates to the local time base are driven by a quartz Crystal Oscillator (XO). The XOs used in typical PCs and low cost digital systems are usually of AT-cut type. All types of XOs, but especially AT-cut ones, suffer from frequency deviations. The difference between the actual XO oscillation frequency and the nominal frequency is referred to as *skew*.

TABLE III
SYNCHRONIZATION AND ACTUATION ERRORS VS. LOCAL PC LOAD AND k VALUE (ALL VALUES IN μS).

	No I/O load					I/O load					Periodic I/O load				
k	μ	σ	max	$p_{99.9}$	p_{99}	μ	σ	max	$p_{99.9}$	p_{99}	μ	σ	max	$p_{99.9}$	p_{99}
25	0.454	0.364	5.337	2.671	1.568	0.670	0.584	14.968	5.020	2.472	0.552	0.500	11.876	3.985	2.223
50	0.277	0.221	2.957	1.626	0.952	0.398	0.331	4.628	2.343	1.489	0.332	0.285	3.555	2.176	1.269
100	0.243	0.176	1.492	1.017	0.750	0.357	0.278	3.753	1.603	1.176	0.238	0.198	3.001	1.376	0.881
200	**0.236**	**0.170**	**1.618**	**0.951**	**0.708**	**0.270**	**0.215**	**2.335**	**1.316**	**0.931**	**0.230**	**0.182**	**1.463**	**1.095**	**0.808**
400	0.292	0.254	3.288	2.537	1.279	0.283	0.292	2.576	2.038	1.481	0.457	0.354	2.280	1.913	1.532
600	0.454	0.636	6.643	5.728	3.807	0.352	0.297	2.300	2.031	1.434	1.125	0.884	6.737	6.055	4.217
800	0.544	0.640	4.511	3.978	3.155	0.469	0.325	3.465	2.971	1.376	1.624	1.272	8.374	7.631	5.972

Estimated synchronization error ($\hat{\epsilon}^{SYN}$): scheduler latencies are compensated in order to assess RBIS synchronization quality.

200	**1.159**	**0.896**	**9.010**	**5.670**	**3.560**	**1.881**	**1.606**	**16.060**	**9.880**	**6.310**	**4.597**	**3.513**	**22.700**	**14.400**	**11.150**

Actuation error (ϵ): synchronization quality at the application level, as seen on the parallel ports of PCs (no compensation is carried out).

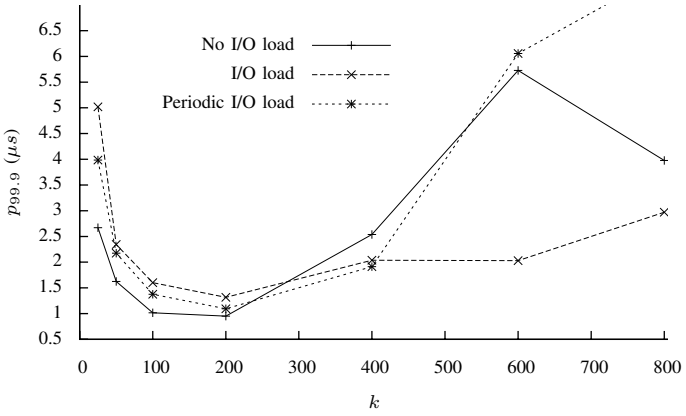


Fig. 5. 99.9-percentile of the estimated synchronization error ($\hat{\epsilon}^{SYN}$) vs. number of samples k used for LSLR, for different PC load conditions.

The variable component of the skew, also known as *drift* (defined as "the systematic change in frequency with time of an oscillator"), is due to a combination of several effects: temperature, supply voltage, vibrations, aging, and so on [27]. While the fixed part of the *skew* can be easily compensated by control algorithms, dynamics of phenomena that contribute to frequency *drifts* lie within a range of frequencies much wider than the rate at which the CSP takes timestamps and performs clock corrections (*Sync* messages in the analyzed RBIS testbed are sent every $100\,\mathrm{ms}$). High-frequency factors, which affect the drift faster than t_{Sync}, such as vibrations and power supply, cannot be compensated. Instead, all factors whose dynamics are slower than t_{Sync} could be, in principle, corrected by the control algorithm perfectly.

Unfortunately, timestamps acquired in s/w—and used by our RBIS implementation to regulate the clock of the nodes—suffer from non-negligible jitter. The standard deviation of such jitter in the proposed testbed was evaluated with separate experiments, and was found to be in the order of $1\,\mu\mathrm{s}$. In order to reduce the related error, every SN applies LSLR on k contiguous timestamps to estimate the current offset and skew of the local clock with respect to the reference clock (MN). LSLR provides the best results when samples in the related interval (whose width is $(k-1)\cdot t_{Sync}$) can be approximated with a straight line, i.e., when variations of the oscillator frequency are negligible in such interval. Thus, when choosing the value for k, a tradeoff has to be found between the *linearity error* due to the frequency drift (that increases as k grows) and the *residual error* due to jitters (that is strictly decreasing for increasing values of k). As a consequence, slow drifts due to, e.g., (graceful) temperature variations or aging, are satisfactorily corrected by the LSLR-based approach.

The optimal value for k depends on the jitter affecting timestamps (which is closely related to the interfering load inside nodes due to best-effort concurrent tasks), but also on environmental factors, often not under control in general and, more specifically, in the experiments we carried out. Table III and Figure 5 report experimental results aimed at finding the optimal value of k for the proposed testbed and for different PC load conditions. All experiments lasted $10\,\mathrm{hours}$ and results are obtained on 72000 samples. The statistical indexes reported for both the synchronization ($\hat{\epsilon}^{SYN}$) and the actuation (ϵ) errors include the expected value (μ) and the standard deviation (σ), as well as the maximum (max) and two percentiles ($p_{99.9}$ and p_{99}) evaluated on the *absolute* values. Among the possible choices for the interfering load, a task invoking a huge number of disk I/O operations was chosen, because it is one of the main sources of interference for real-time operating systems [26].

Three experimental conditions were selected: *No I/O load* (no additional tasks running on PCs, besides the typical load caused by the operating system), *I/O load* (severe disk load, obtained through the `dd` Linux utility), and *Periodic I/O load* (I/O operations are cyclically switched on and off in both MN and SN). In the latter case, the additional in-node load was automatically changed every $300\,\mathrm{s}$ using the following four combinations, in the reported order: no I/O load in both MN and SN, I/O load in both MN and SN, no I/O load in MN and I/O load in SN, and I/O load in MN and no I/O load in SN. The results in Table III and Figure 5 show that, for the specific testbed, the optimal value of k is (around) 200. Consequently, all the following experiments were carried out with such value. As can be seen, for the reported results the optimal k value is mostly independent of the interfering load. When $k < 200$ performance worsen because of timestamp jitters, whereas when $k > 200$ LSLR is able to smooth such jitters but the error due to the oscillator frequency drift becomes predominant. The irregular shape of the curves in Figure 5 when k is in the range from 400 to 800 is due to the fact that, in the experiments,

TABLE IV
SYNCHRONIZATION ERROR VS. NETWORK LOAD AND n_{ts} VALUE (NO PC LOAD, $k = 200$).

Network load	n_{ts}	Synchronization error $\hat{\epsilon}^{SYN}$ (μs)					Percentage of received/matched timestamps (%)				
		μ	σ	max	$p_{99.9}$	p_{99}	$\%_{Sync\_MN}$	$\%_{Sync\_SN}$	$\%_{Follow\_Up}$	$\%_{LSLR^*}$	$\%_{LSLR}$
0 %	20	0.136	0.107	1.044	**0.643**	0.464	94.6	99.5	99.4	94.3	**94.3**
20 %	20	0.137	0.117	1.043	**0.668**	0.500	78.1	82.2	87.4	72.0	**71.2**
40 %	10	0.231	0.190	2.261	1.643	0.800	57.7	74.7	75.6	50.7	44.9
40 %	20	0.229	0.192	1.265	**1.032**	0.810	62.8	78.8	76.7	53.0	**52.3**
40 %	40	0.281	0.193	1.246	0.952	0.769	70.7	67.1	70.0	54.3	54.1

the rate at which the environment temperature varied was not under our control at all. By selecting $k = 200$, the maximum estimated synchronization error ($\hat{\epsilon}^{SYN}_{max}$) is $\sim 2.4\,\mu$s in the *I/O load* condition, and decreases to $\sim 1.6\,\mu$s in the *No I/O load* condition. In all cases, the 99.9 percentile of $\hat{\epsilon}^{SYN}$ stays below $1.4\,\mu$s. The measured worst-case actuation error (ϵ_{max}), which takes into account also the jitter due to the RTAI scheduler, can be up to $\sim 23\,\mu$s in the unrealistic (but possible) *periodic I/O load* condition, and decreases below $10\,\mu$s in the typical *No I/O load* condition. The above results show that, in a distributed application based on RBIS and RTAI, the most part of the error that may affect synchronized actuation is related to the real-time capabilities of the operating system. Conversely, the error due to node synchronization affects the quality of actuation timings marginally.

C. Network load

Network load affects the synchronization error, as it may cause a fraction of the *Sync* and *Follow_Up* messages to be lost. The probability that this event actually takes place is non-negligible, because these messages are sent in broadcast and, hence, they are not confirmed. A number of timestamps n_{ts} greater than n_{ts}^{min} can be included in each *Follow_Up* message, in order to improve robustness against message losses. Although this approach could be effective, it is not worth embedding too many timestamps, as the older ones are hardly useful and they may even worsen the performance of the virtual clock control algorithm in SNs. Unfortunately, nothing can be done to cope with the loss of *Sync* messages, but there are plenty of them since beacons are exploited.

A number of experiments were carried out, aimed at evaluating the impact of the network load and the n_{ts} parameter on both the synchronization error and the number of lost messages. Results are reported in Table IV. In the 20% and 40% network load conditions, packets with payload $1450\,\mathrm{bytes}$ were cyclically broadcast over the air with period $1390\,\mu$s and $695\,\mu$s, respectively. These interfering frames were sent on the same channel as the RBIS testbed, setting the transmitter in *ad-hoc* mode to avoid the double hop of *infrastructure* networks.

The percentage of *Sync* messages correctly received by MN ($\%_{Sync\_MN}$) and SN ($\%_{Sync\_SN}$), as well as of *Follow_Up* messages successfully delivered to SN ($\%_{Follow\_Up}$) were logged. The value $\%_{LSLR^*}$ reported in Table IV is the fraction of timestamps that could be paired by SN if all *Follow_Up* messages were received correctly (hence considering only *Sync* message losses in both MN and SN). Instead, $\%_{LSLR}$ is the fraction of timestamps actually paired by SN. This value is lower than $\%_{LSLR^*}$ since some *Follow_Up* messages can be lost, and depends on the n_{ts} parameter.

Results with no interfering network traffic (first row of Table IV) are slightly different from those reported in Table III in the same experimental conditions. Again, the reason is that the environmental temperature, as well as the traffic caused by nearby Wi-Fi networks operating on the same channel, were not under our control. For instance, worse results were obtained in the experiments where the testbed experienced rapid temperature variations. All experimental results in Table IV were performed between $9\,\mathrm{pm}$ and $7\,\mathrm{am}$, when the room air conditioning system was switched off (i.e., temperature variations were slow).

When $n_{ts} = 20$, in the absence of purposely injected network traffic, almost all timestamps can be paired by SN and exploited by the control algorithm for synchronization ($\%_{LSLR} = 94.3\,\%$). A small performance worsening is experienced when the network load is increased to $20\,\%$ and $40\,\%$, despite only $71.2\,\%$ and $52.3\,\%$ of timestamps can be used for synchronization, respectively (rightmost column of Table IV). In fact, percentile $p_{99.9}$ increased from $643\,\mathrm{ns}$ with no network load to $\sim 1\,\mu$s with $40\,\%$ network load.

When the network load is high ($40\,\%$), the synchronization error increases tangibly if the number of timestamps included in *Follow_Up* messages is lowered, e.g., $n_{ts} = 10$ (see both max and $p_{99.9}$ values). In this experimental condition, if all the timestamps acquired by MN were delivered to SN, the control algorithm could use $\%_{LSLR^*} = 50.7\,\%$ of them to perform clock corrections. Unfortunately, only $\%_{LSLR} = 44.9\,\%$ can be matched by SN due to *Follow_Up* losses. Setting $n_{ts} = 20$ is a good compromise between overheads and performance, because the difference between $\%_{LSLR^*}$ and $\%_{LSLR}$ ($53.0\,\%$ and $52.3\,\%$, respectively) is much smaller. Increasing n_{ts} further (40 and more) is not advantageous, because the percentage of available timestamps increases negligibly (from $54.1\,\%$ to $54.3\,\%$) and there are no clear improvements from the point of view of the synchronization error ($\hat{\epsilon}^{SYN}$), while network overheads increase considerably. Summing up, synchronization quality in RBIS is much more affected by timestamp accuracy and temperature variations than by the network load.

D. Long-term stability

This experiment had the purpose to assess if our RBIS implementation is able to meet the requirements about availability (i.e., uptime) of real distributed application. To this

TABLE V
SYNCHRONIZATION AND ACTUATION ERRORS OVER 10 DAYS.

Error (all values in μs)	μ	σ	max	$p_{99.9}$	p_{99}
Synchronization ($\hat{\epsilon}^{SYN}$)	0.205	0.176	**3.295**	**1.061**	0.761
Actuation (ϵ_i)	1.163	0.981	**12.390**	**7.110**	3.740

aim, the testbed was run (and evaluated) uninterruptedly for a period of 10 days. No additional in-node and network loads were considered—besides the typical operations performed in background on PCs by the operating system and the network traffic on Wi-Fi due to nearby APs operating on the same channel as the RBIS testbed, respectively.

Results, reported in Table V, show that the synchronization error was always bounded to $3.3\,\mu$s, while the worst-case actuation error was $12.4\,\mu$s. The 99-percentile is $\sim 1\,\mu$s for the synchronization error and $\sim 7\,\mu$s at the actuation level. These results confirm that the presented s/w-based implementation of RBIS manages to ensure the intended performance level also over long time periods.

VI. CONCLUSIONS

A new clock synchronization protocol for Wi-Fi infrastructure networks has been presented in this paper, namely RBIS. In many application contexts it could be a valid alternative to others CSPs. In fact, standard solutions like the timing measurement and timing advertisement mechanisms of IEEE 802.11-2012, or PTP, not always satisfy all the application requirements in terms of bandwidth usage, implementation complexity and cost, and backward compatibility. The main advantage of RBIS is that it can be set up using only commercial off-the-shelf equipment: no changes are required to the AP h/w and f/w, and only small modifications are needed to the s/w of end nodes (i.e., wireless adapter drivers). The RBIS *master/slave* scheme also permits wireless extensions to be integrated in the existing backbone of industrial plants. All it is needed is that MN acts as a slave on the wired side (where PTP is typically employed for synchronization).

As pointed out in the paper, the *receiver/receiver* paradigm of RBIS offers some advantage in terms of performance, too. It has been shown that the estimation of propagation delays (not performed by RBIS), which is needed to achieve sub-μs accuracy, could be inaccurate when node mobility is high—unless the rate of messages used for estimation is suitably increased. All these features make RBIS a promising alternative for wireless industrial networks, when synchronization accuracy in the order of a few μs is required. Moreover, it can be used in other contexts as well: for instance, home automation systems can benefit from RBIS to synchronize a TV set with a number of wireless speakers. RBIS is really advantageous in all those contexts where a wireless infrastructure is already deployed and—as happens in the vast majority of cases—the related APs do not embed any support for clock synchronization.

Results, obtained with an experimental setup based on conventional PCs, open-source real-time operating system, and s/w-based timestamps, are quite encouraging. During tests, which overall lasted more than $500\,\text{hours}$—not counting other experiments not reported in the paper—the synchronization error always stayed below $3.3\,\mu$s. Our testbed can be seen as a (simple) distributed control system. The related performance were evaluated directly, by sampling the actuation signals output by two synchronized, RTAI-based, simple test applications. Even when the load of best-effort tasks on PCs was unrealistically high, the actuation error remained below $23\,\mu$s, while in more reasonable conditions it never exceeded $12.4\,\mu$s. It is worth pointing out that the accuracy of RBIS may differ noticeably in other setups and, as for every CSP, it would be much better if timestamps are taken in h/w.

Future work deals primarily on the implementation of new clock regulation disciplines, more robust than LSLR to timestamp jitters and temperature variations [28]. The experimentation of RBIS in heterogeneous network configurations (that include both PCs and embedded systems) is also envisaged.

REFERENCES

[1] G. Cena, I. Cibrario Bertolotti, S. Scanzio, A. Valenzano, and C. Zunino, "Synchronize your watches: Part I: General-purpose solutions for distributed real-time control," *IEEE Industrial Electronics Magazine*, vol. 7, no. 1, pp. 18–29, 2013.

[2] ——, "Synchronize Your Watches: Part II: Special-Purpose Solutions for Distributed Real-Time Control," *IEEE Industrial Electronics Magazine*, vol. 7, no. 2, pp. 27–39, 2013.

[3] IEEE, "IEEE Standard for a Precision Clock Synchronization Protocol for Networked Measurement and Control Systems," *IEEE Std 1588-2008 (Revision of IEEE Std 1588-2002)*, pp. 1–269, 2008.

[4] ——, "IEEE Standard for Information technology–Telecommunications and information exchange between systems Local and metropolitan area networks–Specific requirements Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications," *IEEE Std 802.11-2012 (Revision of IEEE Std 802.11-2007)*, pp. 1–2793, 2012.

[5] A. Flammini, D. Marioli, E. Sisinni, and A. Taroni, "Design and implementation of a wireless fieldbus for plastic machineries," *IEEE Trans. on Industrial Electronics*, vol. 56, no. 3, pp. 747–755, 2009.

[6] A. Willig, "Recent and Emerging Topics in Wireless Industrial Communications: A Selection," *IEEE Trans. on Industrial Informatics*, vol. 4, no. 2, pp. 102–124, 2008.

[7] G. Cena, A. Valenzano, and S. Vitturi, "Hybrid wired/wireless networks for real-time communications," *IEEE Industrial Electronics Magazine*, vol. 2, no. 1, pp. 8–20, 2008.

[8] A. Willig, K. Matheus, and A. Wolisz, "Wireless Technology in Industrial Networks," *Proc. of the IEEE*, vol. 93, no. 6, pp. 1130–1151, 2005.

[9] H. Trsek and J. Jasperneite, "An isochronous medium access for real-time wireless communications in industrial automation systems - A use case for wireless clock synchronization," in *Int. IEEE Symp. on Precision Clock Synchronization for Measurement Control and Communication (ISPCS)*, 2011, pp. 81–86.

[10] P. Djukic and P. Mohapatra, "Soft-TDMAC: A Software-Based 802.11 Overlay TDMA MAC with Microsecond Synchronization," *IEEE Trans. on Mobile Computing*, vol. 11, no. 3, pp. 478–491, 2012.

[11] R. Moraes, F. Vasques, and P. Portugal, "A TDMA-based mechanism to enforce real-time behavior in WiFi networks," in *IEEE Int. Workshop on Factory Communication Systems (WFCS)*, 2008, pp. 109–112.

[12] G. Cena, L. Seno, A. Valenzano, and C. Zunino, "On the Performance of IEEE 802.11e Wireless Infrastructures for Soft-Real-Time Industrial Applications," *IEEE Trans. on Industrial Informatics*, vol. 6, no. 3, pp. 425–437, 2010.

[13] M. Baldi, R. Giacomelli, and G. Marchetto, "Time-Driven Access and Forwarding for Industrial Wireless Multihop Networks," *IEEE Trans. on Industrial Informatics*, vol. 5, no. 2, pp. 99–112, 2009.

[14] A. Mahmood, G. Gaderer, H. Trsek, S. Schwalowsky, and N. Kero, "Towards high accuracy in IEEE 802.11 based clock synchronization using PTP," in *Int. IEEE Symp. on Precision Clock Synchronization for Measurement Control and Communication (ISPCS)*, 2011, pp. 13–18.

[15] A. Mahmood and F. Ring, "Clock synchronization for IEEE 802.11 based wired-wireless hybrid networks using PTP," in *Int. IEEE Symp. on Precision Clock Synchronization for Measurement Control and Communication (ISPCS)*, 2012, pp. 1–6.

[16] M. Maróti, B. Kusy, G. Simon, and A. Lédeczi, "The flooding time synchronization protocol," in *Proc. of the 2nd Int. Conf. on Embedded networked sensor systems*, ser. SenSys '04. New York, NY, USA: ACM, 2004, pp. 39–49.

[17] J. Elson, L. Girod, and D. Estrin, "Fine-grained network time synchronization using reference broadcasts," *SIGOPS Oper. Syst. Rev.*, vol. 36, pp. 147–163, Dec. 2002.

[18] D. Stanislowski, X. Vilajosana, Q. Wang, T. Watteyne, and K. Pister, "Adaptive Synchronization in IEEE802.15.4e Networks," *IEEE Trans. on Industrial Informatics*, vol. 10, no. 1, pp. 795–802, Feb 2014.

[19] G. Cena, S. Scanzio, A. Valenzano, and C. Zunino, "The reference-broadcast infrastructure synchronization protocol," in *IEEE 17th Conf. on Emerging Technologies Factory Automation (ETFA)*, 2012, pp. 1–4.

[20] Cisco and/or its affiliates (2012), "The Future of Hotspots: Making Wi-Fi as Secure and Easy to Use as Cellular." [Online]. Available: http://www.cisco.com/c/en/us/solutions/collateral/service-provider/service-provider-wi-fi/white_paper_c11-649337.pdf

[21] M. Mock, R. Frings, E. Nett, and S. Trikaliotis, "Continuous clock synchronization in wireless real-time applications," in *The 19th IEEE Symp. on Reliable Distributed Systems, 2000. SRDS-2000.*, 2000, pp. 125–132.

[22] A. Mahmood, R. Exel, and T. Sauter, "Delay and Jitter Characterisation for Software-based Clock Synchronization over WLAN using PTP," *IEEE Trans. on Industrial Informatics*, vol. PP, no. 99, pp. 1–1, 2014.

[23] P. Mantegazza, E. L. Dozio, and S. Papacharalambous, "RTAI: Real Time Application Interface," *Linux Journal*, vol. 2000, Apr. 2000.

[24] D. Fontanelli, D. Macii, P. Wolfrum, D. Obradovic, and G. Steindl, "A clock state estimator for PTP time synchronization in harsh environmental conditions," in *Int. IEEE Symp. on Precision Clock Synchronization for Measurement Control and Communication (ISPCS)*, 2011, pp. 99–104.

[25] G. Giorgi and C. Narduzzi, "Performance Analysis of Kalman-Filter-Based Clock Synchronization in IEEE 1588 Networks," *IEEE Trans. on Instrumentation and Measurement*, vol. 60, no. 8, pp. 2902–2909, 2011.

[26] M. Cereia, I. Cibrario Bertolotti, and S. Scanzio, "Performance of a Real-Time EtherCAT Master Under Linux," *IEEE Trans. on Industrial Informatics*, vol. 7, no. 4, pp. 679–687, 2011.

[27] J. R. Vig, "Quartz Crystal Resonators and Oscillators; For Frequency Control and Timing Applications - A Tutorial," Tutorial, US Army Communications-Electronics Research, Development & Engineering Center Fort Monmouth, NJ, USA, Rev. 8.5.2.2, 2004.

[28] M. Mongelli and S. Scanzio, "Approximating Optimal Estimation of Time Offset Synchronization With Temperature Variations," *IEEE Trans. on Instrumentation and Measurement*, vol. 63, no. 12, pp. 2872–2881, Dec 2014.

Gianluca Cena (SM'09) received the Laurea degree in electronic engineering and the Ph.D. degree in information and system engineering from the Politecnico di Torino, Turin, Italy, in 1991 and 1996, respectively.

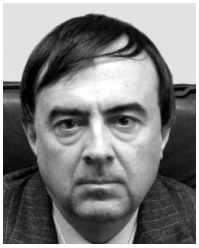
In 1995, he became an Assistant Professor in the Department of Computer Engineering, Politecnico di Torino. Since 2005 he has been a Director of Research with the National Research Council of Italy (CNR), and in particular with the Institute of Electronics, Computer, and Telecommunication Engineering (IEIIT), Turin. His research interests include industrial communication systems and real-time networks. In these areas, he has coauthored more than 100 technical papers.

Dr. Cena served as Program Co-Chairman for the 2006 and 2008 editions of the IEEE Workshop on Factory Communication Systems, and has been an Associate Editor of the IEEE TRANSACTIONS ON INDUSTRIAL INFORMATICS since 2009.

Stefano Scanzio (S'06-M'12) received the Laurea and Ph.D. degrees in computer science from Politecnico di Torino, Torino, Italy, in 2004 and 2008, respectively.

He was with the Department of Computer Engineering, Politecnico di Torino, from 2004 to 2009, where he was involved in research on speech recognition and, in particular, he has been active in classification methods and algorithms. Since 2009, he has been with the National Research Council of Italy (CNR), where he is a tenured technical Researcher with the Institute of Electronics, Computer and Telecommunication Engineering (IEIIT), Turin.

Dr. Scanzio teaches several courses on computer science at Politecnico di Torino. He has authored and co-authored several papers in international journals and conferences in the area of industrial communication systems and real-time networks.



Adriano Valenzano (SM'09) received the Laurea degree in electronic engineering from Politecnico di Torino, Torino, Italy, in 1980. He is Director of Research with the National Research Council of Italy (CNR). He is currently with the Institute of Electronics, Computer and Telecommunication Engineering (IEIIT), Torino, Italy, where he is responsible for research concerning distributed computer systems, local area networks, and communication protocols. He has coauthored approximately 200 refereed journal and conference papers in the area of computer engineering.

Dr. Valenzano is the recipient of the 2013 IEEE IES and ABB Lifetime Contribution to Factory Automation Award. He also received, as a coauthor, the Best Paper Award presented at the Fifth and Eighth IEEE Workshops on Factory Communication Systems (WFCS 2004 and WFCS 2010).

He has served as a technical referee for several international journals and conferences, also taking part in the program committees of international events of primary importance. Since 2007, he has been serving as an Associate Editor for the IEEE TRANSACTIONS ON INDUSTRIAL INFORMATICS.

Claudio Zunino received the degree in computer engineering and the Ph.D. degree in software engineering from the Politecnico di Torino, Turin, Italy, in 2000 and 2005, respectively.

Since 2006, he has been a Researcher with the National Research Council of Italy (CNR). He is currently with the Institute of Electronics, Computer and Telecommunications Engineering (IEIIT).

He has authored and coauthored several papers in international journals and conferences in the area of wireless communication, Industrial Ethernet protocols, computer graphics, parallel and distributed computing, and scientific visualization. He serves as reviewer for a number of international conferences and journals.